ДЖ. ФОРСАЙТ
М. МАЛЬКОЛЬМ
К. МОУЛЕР

МАШИННЫЕ МЕТОДЫ МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ

COMPUTER METHODS FOR MATHEMATICAL COMPUTATIONS

GEORGE E. FORSYTHE
MICHAEL A. MALCOLM
Department of Computer Science
University of Waterloo

CLEVE B. MOLER
Department of Mathematics and Statistics
University of New Mexico

PRENTICE-HALL, INC.
ENGLEWOOD CLIFFS, N.J. 07632
1977

Дж. ФОРСАЙТ
М. МАЛЬКОЛЬМ
К. МОУЛЕР

МАШИННЫЕ МЕТОДЫ МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ

ПЕРЕВОД С АНГЛИЙСКОГО
Х. Д. ИКРАМОВА

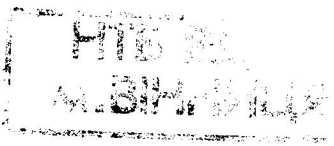
ИЗДАТЕЛЬСТВО «МИР»
МОСКВА 1980

УДК 518.5

Введение в численные методы, ставящее своей целью научить читателя ориентироваться в программах решения типовых задач. Особую ценность придают книге программы на языке Фортран для решения основных задач численного анализа. Авторы книги — видные специалисты; Дж. Форсайт известен советскому читателю по переводам его книг «Разностные методы решения дифференциальных уравнений в частных производных» (совместно с В. Вазовым, М.: ИЛ, 1963) и «Численное решение систем линейных алгебраических уравнений» (совместно с К. Молером, М.: Мир, 1969).

Книга будет полезна специалистам по вычислительной и прикладной математике, а также студентам, изучающим численные методы.

Редакция литературы по математическим наукам



1702070000

Ф $\frac{20204\text{-}035}{041\ (01)\text{-}80}$ 35-80

ОТ ПЕРЕВОДЧИКА

Эта книга является элементарным введением в курс численного анализа. Поэтому она содержит сравнительно небольшое число методов решения типовых математических задач, таких, как решение линейных и нелинейных уравнений, численные квадратуры, интегрирование обыкновенных дифференциальных уравнений и др. Среди включенных в книгу методов лишь немногие разобраны достаточно подробно; знакомство с остальными происходит на «идейном» уровне. Во всех этих отношениях книга заметно отличается от известных советскому читателю учебников по численным методам, которые почти без исключения являются изданиями большого объема. Отмеченные особенности книги являются следствием принятого авторами своеобразного подхода. Своей целью авторы поставили научить читателя не столько методам, сколько правильному обращению с существующими программами, реализующими эти методы.

Обычно в учебниках по численному анализу изложение метода доводится до рабочих формул, иногда с обсуждением тонкостей и подвохов, чтобы читатель при необходимости сам мог составить программу нужного ему метода. Авторы, напротив, считают, что высококачественные программы, т. е. программы, обходящие возможные машинные нули и переполнения, учитывающие нерегулярности вычислительной схемы метода, оценивающие достигнутую точность, обрабатывающие аварийные ситуации с выдачей диагностики,— такие программы могут и должны писать только специалисты. Интересно в связи с этим обсуждение в конце гл. 2 того, какой должна быть программа решения . . . квадратного уравнения.

Итак, составление стандартных программ — дело специалистов; массовому же пользователю ЭВМ знакомство с методом, как правило, необходимо лишь в той мере, чтобы суметь правильно обратиться к соответствующей подпрограмме, правильно интерпретировать выданные ею результаты, верно реагировать на нестандартные исходы и т. д.

В соответствии с изложенными принципами все главы книги, начиная с третьей, построены по одному плану. Рассматривается очередная типовая численная задача, дается краткий обзор основных подходов к ее решению и выделяется определенный метод для более подробного описания, за которым следует реализующая этот метод фортран-подпрограмма. Работа подпрограммы иллюстрируется примером, а глава заканчивается списком содержательных задач, решение которых требует реального счета по этой подпрограмме. Отметим, кстати, нетрадиционное содержание двух последних глав книги, посвященных сингулярному разложению матрицы и датчикам псевдослучайных чисел.

Особо следует сказать о качестве подпрограмм, включенных в книгу. Они отнюдь не являются учебными иллюстрациями, а входят, каждая в своей категории, в число лучших, существующих в настоящее время. Таким образом, наш читатель получает не только оригинальный учебник, но и небольшую библиотеку высококачественных программ по численным методам.

Авторы книги — видные специалисты в области вычислительной математики. В частности, профессор Форсайт хорошо известен советским читателям по переводам его книг «Разностные методы решения дифференциальных уравнений в частных производных» (совместно с В. Вазовым) и «Численное решение систем линейных алгебраических уравнений» (совместно с К. Молером).

В качестве переводчика этой книги я пользовался консультациями ряда товарищей. Всем им, и в особенности О. Б. Арушаняну, С. Ф. Залёткину, Н. Н. Колесникову, Н. В. Петри, Н. Е. Плигиной, мне хотелось бы высказать свою признательность.

Х. Икрамов

В математической стране, образно говоря,
есть пропасти и ущелья, куда можно свалиться;
но это не должно удерживать нас от прогулок.

Л. Ф. Ричардсон.

ПРЕДИСЛОВИЕ

Эта книга возникла из лекционных записей, использовавшихся во вводных курсах численных методов в университетах Станфорда, Нью-Мексико и Ватерлоо. В этих университетах имеется два подобных курса, называемых приблизительно так: «Применение ЭВМ для расчетов» [1]) и «Численный анализ». Курс «Применение ЭВМ для расчетов» предназначается для студентов естественнонаучных и инженерных специализаций, желающих использовать в своей работе численные методы. Он рассчитан на один семестр и опирается на минимальную математическую подготовку. Курс численного анализа охватывает по существу те же методы, но длится целый год. Дополнительное время отводится более глубокому математическому анализу методов. Эта книга задумана главным образом для первого из названных курсов, но ее можно использовать и как дополнение к более абстрактному учебнику по численному анализу.

В книгу включены десять фортранных подпрограмм. Мы рассчитываем, что они будут интенсивно использоваться как во время обучения, так и после него. Программы составлялись тщательно, и с небольшими изменениями или вовсе без изменений они пригодны для широкого круга машин.

Один наш коллега сказал, что «написать две хороших подпрограммы на порядок легче, чем решить, какая из них лучше.» При выборе между различными подпрограммами, имеющимися для данной задачи, мы большое значение отводили ясности и стилю программирования. Если несколько подпрограмм имели сравнимую точность, надежность и эффективность, то мы выбирали ту, которую легче читать и использовать.

В последние годы все больше признается важность создания высококачественного математического обеспечения, а также его тестирования и распространения. Поэтому, хотя мы и считаем, что в период написания этой книги наши подпрограммы входили в число лучших, все же ожидаем и надеемся, что в конечном счете они будут превзойдены. Одной из наших целей при написании книги было научить читателя разыскивать лучшие подпрограммы, распознавать и оценивать их, когда они появляются.

---

[1]) В оригинале — «Numerical Computing».— *Прим. перев.*

У авторов можно получить копии подпрограмм, написанных на языке ANSI Standard Fortran (1966). Версии с обычной и с двойной точностью записываются на одной ленте. Возможны два формата ленты: 7 дорожек, BCD, 556 BPI, и 9 дорожек, EBCDIC, 880 BPI. По специальному запросу может быть предоставлена и колода, содержащая около 2000 перфокарт. Обращаться нужно по адресу: Prof. Cleve Moler, Department of Mathematics, University of New Mexico, Albuquerque, N. M. 87131.

Мы признательны множеству лиц, оказавших нам помощь и поддержку в процессе подготовки рукописи. Вот перечень фамилий некоторых из них: Ричард Аллен, Мэри Бодли, Джон Болстэд, Эрин Брент, Ричард Брент, Франсена Брамбо, Энди Кон, Карл де Бур, Фрэд Дорр, Уильям С. Гир, Кит Геддес, Эйлен Джордж, Джин Голуб, Денис Есперсен, Рондэл Джонс, Уильям Кахан, Дэвид Каханер, Шэрон Кент, Фрэд Крог, Дог Лоусон, Джеймс Линес, Кэтрин Моулер, Тереза Моулер, Уолтер Мюррей, Лэрри Назарет, Джо Олиджер, Виктор Перейра, Стив Пруэс, Джордж Реймос, Лори Роджерс, Майкл Сондерс, Ларри Шампэнь, Лаура Стэггерс, Дж. У. Стьюарт, Майк Штойервальт, Дэвид Стаутмайер, Ричард Андервуд, Г. Э. Уотс и Майкл Уэстер. Особенно мы благодарны многочисленным студентам, страдавшим во время чтения ранних вариантов рукописи, а также помогавшим вылавливать ошибки в подпрограммах.

Еще до завершения рукописи, 9 апреля 1972 г., безвременно, в возрасте 55 лет скончался профессор Джордж Э. Форсайт. Профессор Форсайт был основателем и деканом факультета «Компьютер сайенс» [1]) в Станфордском университете. Дань его памяти отдали академические и научные сообщества, а мы, друзья, коллеги и студенты Джорджа, ощутили глубоко личную утрату. Выступая на траурном собрании, профессор Эдвард А. Файгенбаум выразил чувства многих из нас, когда сказал о профессоре Форсайте: «Это был один из самых кротких и человечных людей, и мы любили его за это... Духу этого человека было чуждо всякое насилие...»

Дух этой книги — дух Джорджа Форсайта. Мы хотели бы, чтобы она нашла мирные применения, достойные его памяти.

Майкл А. Малькольм
Университет Ватерлоо
Клив Б. Моулер
Университет Нью-Мексико

[1]) Computer Science — американский аналог советских факультетов вычислительной и прикладной математики.— *Прим. перев.*

1. ВВЕДЕНИЕ

В настоящей книге рассматривается решение математических задач с использованием автоматических цифровых вычислительных машин. Предполагается, что читатель имеет за плечами два года обучения высшей математики. Сюда должно входить дифференциальное и интегральное исчисление и кое-что из теории матриц и дифференциальных уравнений. Предполагается также, что читатель имеет доступ к средней или мощной машине для научных расчетов и сможет программировать для этой машины на местном диалекте ФОРТРАНа.

Очень важную часть книги составляет набор фортранных подпрограмм. В сущности, книгу вполне можно было бы рассматривать как пространный «путеводитель» по этим подпрограммам. Они отнюдь *не* являются простыми иллюстративными программами, которые можно найти во многих учебниках; напротив, они адекватно отражают уровень, достигнутый в современных научных вычислениях. Мы рассчитываем, что эти подпрограммы будут читать не только машины, но и люди; вот почему они включены в текст. Мы рассчитываем также на то, что эти программы будут использованы для решения задач, приведенных в книге.

Здесь принят довольно нетрадиционный подход к изложению материала. Большинство инженеров и научных работников не имеют ни времени, ни склонности следить за новинками текущей литературы по численному анализу. Замечено, что многие люди, решающие практические численные задачи, остаются навсегда верными раз использованным методам, если не программам. Поэтому мы решили описать самые современные подпрограммы для решения стандартных математических задач.

Многие из этих подпрограмм весьма сложны, и для того, чтобы описать во всех деталях, как и почему они работают, потребовалось бы гораздо больше времени, чем может затратить на материал этого типа большинство студентов. Следовательно, необходимо рассматривать подобные подпрограммы как

черные ящики. Оперирование подпрограммами как черными ящиками является обычной практикой. Аппаратно реализованные подпрограммы для сложения, умножения и т. п. рассматриваются как черные ящики большинством программистов. Лишь немногие люди действительно понимают, как они работают. За последние годы установился обычай трактовать как черные ящики и подпрограммы вычисления стандартных функций анализа (например, синуса, косинуса и т. д.). Мало кто из программистов знает эти подпрограммы во всех деталях. Они принимаются как нечто заданное и заведомо работающее. По мнению авторов, теперь настало время, чтобы программисты приняли как нечто заданное и (обычно) работающее и подпрограммы, решающие линейные системы, обыкновенные дифференциальные уравнения и многие другие стандартные математические задачи.

Однако недостаточно просто объяснить, как обращаться к каждой подпрограмме, и сказать, что она «обычно работает». В численных расчетах всегда имеется бездна ловушек. Студента нужно предупредить о них. Он должен научиться распознавать симптомы численного «нездоровья» и правильно диагностировать задачу (задачи). Это требует определенного уровня понимания численных методов, реализованных подпрограммами. У программиста может также возникнуть надобность в переделке программы или использовании варианта данного метода для решения родственной задачи. Обычно это требует более доскональных познаний.

Мы пытались придерживаться компромиссного подхода к степени подробности описания каждого метода. Возможно, что в результате студент будет рассматривать подпрограммы как «серые», а не черные ящики. Там, где это уместно, помещены ссылки на литературу, содержащую более глубокую трактовку конкретных вопросов.

Краткое обсуждение стиля и различных решений, принятых нами в фортран-подпрограммах, дано в § 1.2 этой главы.

Поскольку мы смогли осветить лишь малую часть своего предмета, большая надежда возлагается на ссылки к другим книгам и статьям. Поэтому начинаем свое изложение с библиографии.

1.1. Библиография

Две вводные книги, предназначенные главным образом для тех, кто считает на настольных машинах, однако в значительной мере сохраняющие свою ценность и для пользователей автоматических вычислительных машин:

Ланцош (Lanczos C.)
(1957) Applied analysis. Englewood Cliffs, N. J.: Prentice-Hall. (Русский перевод: Ланцош К. Практические методы прикладного анализа. Пер. с англ.— М.: Физматгиз, 1961)

Хильдебранд (Hildebrand F. B.)
(1974) Introduction to numerical analysis, 2nd ed. New York: McGraw-Hill.

Некоторые книги общего характера, рассчитанные на пользователей ЭВМ:

Березин И. С., Жидков Н. П.
(1962) Методы вычислений. Том 2. Изд. 2-е.—М.: Наука.
(1965) Методы вычислений. Том 1. Изд. 3-е.— М.: Наука.

Вендрофф (Wendroff B.)
(1969) First principles of numerical analysis. An undergraduate text. Reading, Mass.: Addison-Wesley.

Дальквист, Бьорк (Dahlquist G., Björck Å.)
(1974) Numerical methods, Englewood Cliffs, N.J.: Prentice-Hall. (Полный и современный учебник по численному анализу.)

Контэ, де Бур (Conte S. D., deBoor C.)
(1972) Elementary numerical analysis; an algorithmic approach, 2nd ed., New York: McGraw-Hill.

Мак-Кракен, Дорн (McCraken D. D., Dorn W. S.)
(1964) Numerical methods and Fortran programming. New York: Wiley. (Русский перевод: Мак-Кракен Д. Д., Дорн У. С. Численные методы и программирование на ФОРТРАНе. Изд. 2-е. Пер. с англ.— М.: Мир, 1977.)

Рэлстон (Ralston A.)
(1965) A first course in numerical analysis. New York: McGraw-Hill. (В противоположность книгам Хенричи и Хемминга в учебнике Рэлстона значительное внимание уделено вычислительным методам линейной алгебры.)

Рэлстон, Уилф (Ralston A., Wilf H. S.)
(1960, 1967) Mathematical methods for digital computers, vol. 1 and vol. 2. New York: Wiley. (Математические описания и блок-схемы алгоритмов различного качества. Те, что в томе 2, вообще говоря, много лучше.)

Тодд (ред.) (Тодд J.)
(1962) Survey of numerical analysis. New York: McGraw-Hill.

Фокс, Майерс (Fox L., Mayers D. F.)
(1968) Computing methods for scientists and engineers. Oxford: Clarendon Press.

Хемминг (Hamming R. W.)
(1971) Introduction to applied numerical analysis. New York: McGraw-Hill.
(1973) Numerical methods for scientists and engineers, 2nd ed. New York: McGraw-Hill.
(Русский перевод: Хемминг Р. В. Численные методы для научных работников и инженеров. Изд. 2-е. Пер. с англ.— М.: Мир, 1977.) (Хемминг хочет, чтобы вы научились при решении своих задач соединять математическую теорию, эвристический анализ и численные методы: «Цель вычислений — не числа, а постижение сути».)

Хенричи (Henrici P.)
(1964) Elements of numerical analysis. New York: Wiley. (Тщательно написанный элементарный учебник. О матричных задачах нет ничего.)

Шампэнь, Аллен (Shampine L., Allen R.)
(1973) Numerical computing. Philadelphia: Saunders. (Учебник, схожий с данной книгой. Содержит несколько хороших фортран-подпрограмм.)

Эймс (Ames W. F.)
(1969) Numerical methods for partial differential equations. New York: Barnes and Noble.

Эктон (Acton F. S.)
(1970) Numerical methods that (usually) work. New York: Harper and Row. (Особенно хороши две главы об устранении особенностей.)

Библиография таблиц:

Флетчер, Миллер, Розенхед (Fletcher A., Miller J.C.P., Rosenhead L.)
(1962) Index of mathematical tables, 2nd ed., 2 vols. Oxford: Blackwell's. (Здесь перечислены почти все таблицы математических функций, опубликованные до 1958 или 1959 г., а их было очень много. Эта книга необходима.)

Справочники по аппроксимации трансцендентных функций элементарными:

Гастингс и др. (Hastings C., Jr., et al.)
(1955) Approximations for digital computers. Princeton, N.J.: Princeton University Press. (Формулы с графиками кривых ошибок. Большинство машинных

библиотечных программ использует эти формулы либо их уточнения.)

Люстерник Л. А., Червоненкис О. А., Янпольский А. Р. (1963) Математический анализ. Вычисление элементарных функций.— М.: Физматгиз.

Харт и др. (Hart J. F., et al.) (1968) Computer approximations. New York: Wiley.

Бестселлер:

Абрамовиц, Стиган (ред.) (Abramowitz M., Stegun I.) (1964) Handbook of mathematical functions with formulas, graphs, and mathematical tables. Washington, D.C.: Government Printing Office. (Русский перевод: Абрамовиц М., Стиган И. Справочник по специальным функциям. Пер. с англ.—М.: Наука, 1979.) (Издательство Dover выпустило переиздание в мягкой обложке. Здесь есть все для того, кто лишь от случая к случаю занимается вычислением и использованием специальных функций. Продано свыше 150 000 экземпляров).

Некоторые специализированные книги и книги, расчитанные на высокий уровень подготовки по численным методам:

Алберг, Нильсон, Уолш (Ahlberg J. H., Nilson E. N., Walsh J. L.) (1967) The theory of splines and their applications. New York: Academic Press. (Русский перевод: Алберг Дж., Нильсон Э., Уолш Дж. Теория сплайнов и ее приложения. Пер. с англ.— М.: Мир, 1972.)

Бабушка, Витасек, Прагер (Babuška I., Prager M., Vitásek E.) (1966) Numerical processes in differential equations. New York: Wiley-Interscience. (Русский перевод: Бабушка И., Витасек Э., Прагер М. Численные процессы решения дифференциальных уравнений. Пер. с англ.— М.: Мир, 1969.)

Блэкман, Тьюки (Blackman R. B., Tukey J. W.) (1958) The measurement of power spectra. New York: Dover.

Брент (Brent R. P.) (1972) Algorithms for minimization without derivatives. Englewood Cliffs, N.J.: Prentice-Hall.

Варга (Varga R. S.) (1962) Matrix iterative analysis. Englewood Cliffs, N.J.: Prentice-Hall. (Эта книга посвящена главным образом большим линейным системам, возникающим при решении уравнений в частных производных такого типа, как, например, в расчете реакторов; особо обсуждаются неявные методы.)

Гилл, Мюррей (Gill P., Murray W.)
(1975) Constrained optimization. New York: Academic Press.
Гир (Gear C. W.)
(1971) Numerical initial value problems in ordinary differential equations. Englewood Cliffs, N.J.: Prentice-Hall.
Грегори, Карни (Gregory R. T., Karney L. K.)
(1969) A collection of matrices for testing computational algorithms. New York: Wiley-Interscience.
Гринспэн (Greenspan D.)
(1968) Lectures on the numerical solution of linear, singular and nonlinear differential equations. Englewood Cliffs, N. J.: Prentice-Hall.
Данциг (Dantzig G. B.)
(1963) Linear programming and extensions. Princeton, N.J.: Princeton University Press. (Русский перевод: Данциг Дж. Б. Линейное программирование, его применения и обобщения. Пер. с англ.— М.: Прогресс, 1966.)
Деккер (Dekker T. J.)
(1968) Algol 60 procedures in numerical algebra, Part 1; Part 2 (with W. Hoffmann), Amsterdam: Mathematisch Centrum.
Дэвис (Davis P. J.)
(1963) Interpolation and approximation. Waltham, Mass.: Ginn/Blaisdell.
Дэвис, Рабинович (Davis P. J., Rabinowitz P.)
(1975) Methods of numerical integration. New York: Academic Press.
Исааксон, Келлер (Isaacson E., Keller H. B.)
(1966) Analysis of numerical methods. New York: Wiley. (Учебник по численному анализу на уровне начинающего аспиранта.)
Келлер (Keller H. B.)
(1968) Numerical methods for two-point boundary value problems. Waltham, Mass.: Ginn/Blaisdell.
Кнут (Knuth D. E.)
(1969) The art of computer programming, vol. 2: «Seminumerical algorithms». Reading, Mass.: Addison-Wesley. (Русский перевод: Кнут Д. Е. Искусство программирования. Том 2. Получисленные алгоритмы. Пер. с англ.— М.: Мир, 1977.) (См. гл. 3 о случайных числах и гл. 4 об арифметике с плавающей точкой.)
Ковалик, Осборн (Kowalik J., Osborne M. R.)
(1968) Methods for unconstrained optimization problems. New York: American Elsevier.

Коллатц (Collatz L.)
(1966) The numerical treatment of differential equations, 3rd ed. Berlin: Springer.
(1966) Functional analysis and numerical mathematics. New York: Academic Press. (Русский перевод: Коллатц Л. Функциональный анализ и вычислительная математика. Пер. с нем.— М.: Мир, 1969.) (Применения функционального анализа к вычислительным задачам.)
Крылов В. И.
(1967) Приближенное вычисление интегралов. Изд. 2-е.— М.: Наука.
Лоусон, Хэнсон (Lawson C. L., Hanson R. J.)
(1974) Solving least squares problems. Englewood Cliffs. N.J.: Prentice-Hall.
Лэсдон (Lasdon L. S.)
(1970) Optimization theory for large systems. New York: Macmillan. (Русский перевод: Лэсдон Л. С. Оптимизация больших систем. Пер. с англ.— М.: Наука, 1975.)
Мур (Moore R. E.)
(1966) Interval analysis. Englewood Cliffs, N.J.: Prentice-Hall. (Теория замены арифметики чисел арифметикой действительных интервалов; цель — автоматизировать анализ ошибок.)
Мюррэй (Murray W.)
(1972) Numerical methods for unconstrained optimization. New York: Academic Press.
Ортега (Ortega J. M.)
(1972) Numerical analysis, a second course. New York: Academic Press.
Ортега, Рейнболдт (Ortega J. M., Rheinboldt W. C.)
(1960) Iterative solution of nonlinear equations in several variables. New York: Academic Press. (Русский перевод: Ортега Дж., Рейнболдт В. Итерационные методы решения нелинейных систем уравнений со многими неизвестными. Пер. с англ.— М.: Мир, 1975.)
Райс (Rice J. R.)
(1964) The approximation of functions, vol. 1: «Linear theory»; vol. 2: «Nonlinear and multivariate theory». Reading, Mass.: Addison-Wesley.
(1971) Mathematical software. New York: Academic Press.
Рейд (ред.) (Reid J. K.)
(1971) Large sparse sets of linear equations. New York: Academic Press.

Рейнболдт (Rheinboldt W. C.)
(1974) Methods for solving systems of nonlinear equations. Philadelphia: SIAM.
Ривлин (Rivlin T. J.)
(1969) An introduction to the approximation of functions. Waltham, Mass: Ginn/Blaisdell.
Рихтмайер, Мортон (Richtmyer R. D., Morton K. W.)
(1967) Difference methods for initial-value problems, 2nd ed. New York: Wiley-Interscience. (Русский перевод: Рихтмайер Р. Д., Мортон К. Разностные методы решения краевых задач. Пер. с англ.— М.: Мир, 1972.)
Роуз, Уиллогби (ред.) (Rose D. J., Willoughby R. A.)
(1972) Sparse matrices and their applications. New York: Plenum.
Саульев В. К.
(1960) Интегрирование уравнений параболического типа методом сеток.— М.: Физматгиз.
Смит, Бойл, Гарбоу, Икебе, Клема, Моулер (Smith B. T., Boyle J. M., Garbow B. S., Ikebe Y., Klema V. C., Moler C. B.)
(1974) Matrix eigensystem routines — EISPACK guide. Berlin: Springer.
Снайдер (Snyder M. A.)
(1966) Chebyshev methods in numerical approximation. Englewood Cliffs, N.J.: Prentice-Hall.
Строуд (Stroud A. H.)
(1971) Approximate calculation of multiple integrals. Englewood Cliffs, N.J.: Prentice-Hall.
Строуд, Сикрест (Stroud A. H., Secrest D.)
(1966) Gaussian quadrature formulas. Englewood Cliffs, N.J.: Prentice-Hall.
Стренг, Фикс (Strang W. G., Fix G. J.)
(1973) An analysis of the finite element method. Englewood Cliffs, N.J.: Prentice-Hall. (Русский перевод Стренг Г., Фикс Дж. Теория метода конечных элементов. Пер. с англ.— М.: Мир, 1977.)
Стьюарт (Stewart G. W.)
(1973) Introduction to matrix computations. New York: Academic Press.
Трауб (Traub J. F.)
(1964) Iterative methods for the solution of equations. Englewood Cliffs, N.J.: Prentice-Hall.
Уилкинсон (Wilkinson J. H.)
(1963) Rounding errors in algebraic processes. Englewood Cliffs, N.J.: Prentice-Hall. (Обратный анализ ошибок.)

(1965) The algebraic eigenvalue problem. New York: Oxford University Press. (Русский перевод: Уилкинсон Дж. Алгебраическая проблема собственных значений. Пер. с англ.— М.: Наука, 1970.) (Это важный научный трактат.)

Уилкинсон, Райнш (Wilkinson J. H., Reinsch C.)

(1971) Linear algebra, handbook for automatic computation, vol. 2. Berlin: Springer. (Русский перевод: Уилкинсон Дж., Райнш. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. Пер. с англ.— М.: Машиностроение, 1976.)

Уолш (ред.) (Walsh J.)

(1967) Numerical analysis, an introduction. Washington, D.C.: Thompson Book Co.

Фаддеев Д. К., Фаддеева В. Н.

(1963) Вычислительные методы линейной алгебры. Изд. 2-е.— М.— Л.: Физматгиз.

Фокс (Fox L.)

(1957) The numerical solution of two-point problems. Oxford: Clarendon Press.

Форсайт, Вазов (Forsythe G. E., Wassow W. R.)

(1960) Finite-difference methods for partial differential equations. New York: Wiley. (Русский перевод: Вазов В. Р., Форсайт Дж. Разностные методы решения дифференциальных уравнений в частных производных. Пер. с англ.— М.: ИЛ, 1963.)

Форсайт, Молер (Forsythe G. E., Moler C. B.)

(1967) Computer solution of linear algebraic systems. Englewood Cliffs, N.J.: Prentice-Hall. (Русский перевод: Форсайт Дж. Е., Молер К. Численное решение систем линейных алгебраических уравнений. Пер. с англ.— М.: Мир. 1969.) (Программы для решения систем линейных уравнений на языках АЛГОЛ, ФОРТРАН и PL/1 вместе с соответствующей теорией.)

Хаммерсли, Хэндскомб (Hammersley J. M., Handscomb D. C.)

(1964) Monte Carlo methods. London: Methuen. (Метод псевдослучайного выбора для оценки числовых величин.)

Хаусхолдер (Householder A. S.)

(1964) The theory of matrices in numerical analysis. Waltham, Mass.: Ginn/Blaisdell.

(1970) The numerical treatment of a single nonlinear equation. New York: McGraw-Hill.

Хенричи (Henrici P.)

(1962) Discrete-variable methods in ordinary differential equations. New York: Wiley.

Хэндскомб (ред.) (Handscomb D. C.)
(1966) Methods of numerical approximation. Elmsford, N.Y.: Pergamon. (Весьма современно. Великолепная библиография.)
Шампэнь, Гордон (Shampine L. F., Gordon M. K.)
(1975) Computer solution of ordinary differential equations. San Francisco: W. H. Freeman.
Шульц (Schultz M.)
(1973) Spline analysis. Englewood Cliffs, N.J.: Prentice-Hall.
Янг (Young D. M.)
(1971) Iterative solution of large linear systems. New York: Academic Press.

Избранные журналы:

1. *ACM Transactions on Mathematical Software.*
2. *BIT* (Скандинавский, бóльшая часть статей на английском языке.)
3. *Communications of the ACM* (ACM — Ассоциация по вычислительной технике.)
4. *The Computer Journal.* (Издается Британским обществом (специалистов) по вычислительной технике.)
5. *International Journal for Numerical Methods in Engineering.*
6. *Journal of Computational Physics.*
7. *Journal of the Institute for Mathematics and Its Applications* (Английский.)
8. *Mathematics of Computation* (Американское математическое общество.)
9. *Numerische Mathematik.* (Немецкий, бóльшая часть статей на английском языке.)
10. *SIAM Journal on Numerical Analysis.*
11. *SIAM Review* (SIAM — Общество по использованию прикладной математики в промышленности.)
12. Журнал вычислительной математики и математической физики, СССР.

1.2. О программах, включенных в книгу

Подпрограммы из этой книги являются результатом многих лет исследовательской работы и опыта ряда специалистов по численному анализу. Решение о том, какие программы включить в книгу, а какие указать в списке литературы, было принято после долгих размышлений. Кроме того, мы старались придерживаться более или менее единообразного стиля в программировании и комментариях. Читатель сможет лучше ис-

пользовать эти программы, если будет знать некоторые наши программистские установки. Таков предмет этого параграфа.

Главная проблема, стоящая сейчас перед программированием для научных расчетов,— это транспортабельность [1]) машинных программ. В самом деле, одной из мотивировок при создании языков высокого уровня, таких, как ФОРТРАН или АЛГОЛ, была потребность в программах, которые можно пропускать на различных машинах. Несмотря на значительные усилия по стандартизации языков типа ФОРТРАНа, многие проблемы транспортабельности сохраняются. Каждый производитель вычислительных машин имеет свой собственный вариант ФОРТРАНа. Некоторые из них содержат в качестве подмножества стандартный ANSI FORTRAN, другие нет. Среди фортран-трансляторов, которые допускают ANSI FORTRAN, лишь немногие дают предупредительную диагностику для операторов, не входящих в стандарт ФОРТРАНа. Программы этой книги являются транспортабельными программами языка ANSI FORTRAN, что было проверено посредством программы PFORT Verifier, составленной сотрудником фирмы Bell Telephone Райдером (Райдер (1974)).

Другая важная проблема транспортабельности связана с тем, что различные машины используют разное представление чисел и разную арифметику. Например, представление с плавающей точкой обычной точности в машине CDC 6600 имеет 48 бит, а в IBM 360 — менее 24. В результате большинство вычислений с плавающей точкой в IBM 360 приходится выполнять, используя удвоенную точность — около 56 бит. Но когда той же программой пользуются в CDC 6600, то естественно проводить счет с обычной точностью. Поэтому при поступлении заказа на программу мы поставляли две ее версии: с обычной и с двойной точностью. (Однако в тексте книги приводится лишь вариант с обычной точностью.) Чтобы гарантировать идентичность обеих версий, программы составлялись в двойной точности; преобразование в обычную точность осуществлялось автоматически посредством программы-редактора текстов.

Задача написания транспортабельных программ еще более усложняется тем, что многие программы используют машинно-зависимые константы. Наиболее распространенным в математическом обеспечении машинно-зависимым параметром является *машинное эпсилон*. Это число, которое мы точно определим в гл. 2, связано с разрядностью плавающей арифметики. Машинно-зависимые параметры типа машинного эпсилон традиционно включались в программы в качестве констант. При транспортировке подобной программы на другую машину не-

[1]) В оригинале — portability. *Прим. перев.*

обходимо найти и изменить каждую машинно-зависимую константу. Мы обошли эту трудность, предусмотрев вычисление всех таких параметров при исполнении программы. Эти вычисления требуют лишь ничтожного времени и памяти.

Датчик случайных чисел URAND, описанный в гл. 10, автоматически определяет некоторые характеристики машинной целочисленной арифметики и, следовательно, также транспортабелен. Однако, URAND выполнен лишь в обычной точности, и его точность не может быть изменена на двойную.

УПРАЖНЕНИЯ

1.1. Во многих областях прикладной математики важную роль играет так называемая *функция ошибок*. Она определяется интегральным представлением:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt.$$

Этот интеграл нельзя выразить через более элементарные функции.

Каждая глава книги содержит задачу, связывающую материал этой главы с тем или иным аспектом функции ошибок. Мы начнем с таких заданий:

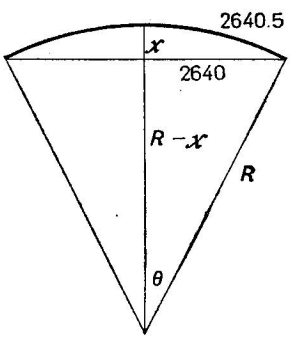
(а) Достаньте таблицу функции erf (x) и найдите значение erf (1.0).

(б) Имеется ли на вашей машине функция или подпрограмма, вычисляющая erf (x)? Если да, то выясните, как пользоваться ею, вычислите значение erf (1.0) и сравните со значением, найденным в таблице.

1.2. Составьте краткий обзор различных трудностей, которые могут встретиться при обращении фортран-программы из двойной точности в обычную или обратно.

1.3. Какова 9000-я десятичная цифра числа π? Постарайтесь «запротоколировать» ход ваших рассуждений, который привел к ответу.

1.4. (Ф. Эктон) Трудность этой задачи не содержательная, а вычислительная. Возьмите настольную или ручную счетную машинку не более чем с десятью десятичными разрядами или же просто карандаш и бумагу. Постарайтесь получить как можно более точный результат. Попробуйте добиться хотя бы пяти значащих десятичных разрядов и оценить точность своего ответа. Объясните подробно свое решение.



Железнодорожный рельс длиной в одну милю жестко закреплен на концах и лежит на плоскости. Некий шутник разрезает рельс и приваривает к нему дополнительный участок в один фут, что вынуждает рельс выгнуться в дугу окружности. Определить максимальную высоту d, которой достигает теперь рельс по отношению к своему прежнему положению.

Указания: На диаграмме x — неизвестная высота, R — неизвестный радиус окружности и θ — угол в радианах, стягиваемый половиной удлиненного рельса. Установите следующие соотношения, которые в дальнейшем нужно использовать для вычисления R, θ и x:

$$R \sin \theta = 2640,$$

$$R\theta = 2640.5,$$

$$R \cos \theta = R - x,$$

$$\sin \theta = \theta - \frac{\theta^3}{6} + \ldots,$$

$$\cos \theta = 1 - \frac{\theta^2}{2} + \ldots.$$

2. ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ТОЧКОЙ

В этой главе мы введем некоторые основные понятия, относящиеся к вычислениям с плавающей точкой. Вначале обсудим обычный способ аппроксимации системы действительных чисел в вычислительной машине, т. е. числа с плавающей точкой. Остальная часть главы посвящена источникам ошибок в вычислениях с плавающей точкой.

2.1. Числа с плавающей точкой

Система действительных чисел с ее неудачным названием столь прочно утвердилась в фундаменте математического анализа, что легко позабыть, что все действительные числа невозможно представить в действительном мире конечных вычислительных машин. Однако, как бы ни упрощала анализ система действительных чисел, практические вычисления вынуждены обходиться без нее.

Много разных методов было предложено для аппроксимации действительных чисел посредством конечных машинных представлений. Метод, принятый в настоящее время почти на всех машинах,— это *числа с плавающей точкой*. Множество F чисел с плавающей точкой характеризуется четырьмя параметрами: основанием β, точностью t и интервалом показателей $[L, U]$. Каждое число x с плавающей точкой, принадлежащее F, имеет значение

$$x = \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_t}{\beta^t} \right) \cdot \beta^e,$$

где целые числа $d_1, \ldots, d_t$ удовлетворяют неравенствам

$$0 \leqslant d_i \leqslant \beta - 1 \quad (i = 1, \ldots, t)$$

и $L \leqslant e \leqslant U$. Если для каждого ненулевого x из F справедливо $d_1 \neq 0$, то плавающая система [1]) F называется *нормализованной*.

[1]) Начиная отсюда, мы позволяем себе переводить словосочетания типа floating-point system, floating-point computation и т. д. как плавающая система, плавающее вычисление и т. д.— *Прим. перев.*

Целое число e называется *показателем*, а число $f=(d_1/\beta+\ldots+d_t/\beta^t)$ — *дробной частью*. Обычно целое число $\beta^t \cdot f$ хранится по той или иной схеме представления, принятой для целых чисел, вроде величины со знаком, дополнения до единицы или дополнения до двух.

Действительная машинная реализация плавающих представлений может отличаться в деталях от идеальной, обсуждаемой здесь, однако различия несущественны, и их почти всегда можно игнорировать, говоря об основных проблемах ошибок округления.

Следующая таблица дает некоторые примеры систем с плавающей точкой. Величина β^{1-t} является оценкой относительной точности арифметики. Мы не приводим точного значения машинного эпсилон [1]), поскольку оно зависит от некоторых тонких деталей, таких, как способ округления.

Если число t не является целым, это означает, что $\beta=2^k$ и для двоичного представления дробной части отводится $k \cdot t$ бит.

Вычислительная машина	β	t	L	U	β^{1-t}
Univac 1108	2	27	−128	127	1.49×10^{-8}
Honeywell 6000	2	27	−128	127	1.49×10^{-8}
PDP-11	2	24	−128	127	1.19×10^{-7}
Control Data 6600	2	48	−976	1 070	7.11×10^{-15}
Cray-1	2	48	−16 384	8 191	7.11×10^{-15}
Illiac-IV	2	48	−16 384	16 383	7.11×10^{-15}
Сетунь	3	18	−121	121	7.74×10^{-9}
Burroughs B5500	8	13	−51	77	1.46×10^{-11}
Hewlett Packard HP-45	10	10	−98	100	1.00×10^{-9}
Texas Instruments SR-5x	10	12	−98	100	1.00×10^{-11}
IBM 360 и 370	16	6	−64	63	9.54×10^{-7}
IBM 360 и 370	16	14	−64	63	2.22×10^{-16}
Telefunken TR440	16	$9\frac{1}{2}$	−127	127	5.84×10^{-11}
Maniac II	65536	$2\frac{11}{16}$	−7	7	7.25×10^{-9}

На некоторых вычислительных машинах используется несколько систем чисел с плавающей точкой. Например, IBM 360 имеет две системы с основанием 16, указанные в таблице. Эти две разные системы называются *обычной точностью* и *удвоенной точностью*.

Множество F не является континуумом или даже бесконечным множеством. В нем ровно $2(\beta-1)\beta^{t-1}(U-L+1)+1$ чисел. Они расположены неравномерно; равномерность расположения

[1]) См. определение в 2.2.— *Прим. перев.*

имеет место лишь при фиксированном показателе. На рис. 2.1 показано 33-точечное множество F для небольшой иллюстративной системы с параметрами $=2$, $t=3$, $L=-1$, $U=2$.

Поскольку F — конечное множество, невозможно сколько-нибудь детально отобразить континуум действительных чисел. Более того, действительные числа с модулем, бо́льшим максимального элемента из F, не могут быть отображены вообще. По ряду причин то же самое верно в отношении ненулевых

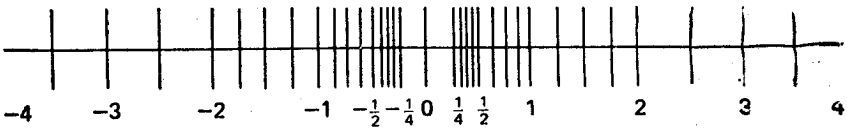


Рис. 2.1. Система чисел с плавающей точкой для $\beta=2$, $t=3$, $L=-1$, $U=2$

действительных чисел, меньших по абсолютной величине по сравнению с наименьшим положительным числом из F. Наконец, каждое число из F должно представлять целый интервал действительных чисел. Если x — действительное число, не выходящее за границы множества F, то будем обозначать через $fl(x)$ элемент из F, ближайший к x. [Заметим, что если x равноудалено от двух чисел из F, то $fl(x)$ может принимать любое из двух соседних с x значений.] Легко доказать, что для числа x, находящегося в границах множества F, справедливо неравенство

$$\left|\frac{fl(x)-x}{x}\right| \leqslant \frac{1}{2}\beta^{1-t}.$$

Поучительный пример дает десятичное число 0.1, которое часто берется в качестве величины шага во многих алгоритмах, обсуждаемых в дальнейшем. Составят ли десять шагов длины 0.1 то же самое, что один шаг длины 1.0? Нет, это не так в плавающей системе, у которой β является степенью 2, поскольку 1/10 не имеет конечного разложения по степеням 1/2. В самом деле,

$$\frac{1}{10} = \frac{0}{2^1} + \frac{0}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \ldots .$$

Используя индексы для обозначения основания системы, можем написать

$$\begin{aligned}(0.1)_{10} &= (0.000110011001100\ldots)_2\\ &= (0.012121212\ldots)_4\\ &= (0.063146314\ldots)_8\\ &= (0.199999999\ldots)_{16}.\end{aligned}$$

В дробях правой части нужно ограничиться t разрядами. Когда складывается десять таких дробей, результат не будет точной единицей.

На множестве F, являющемся моделью системы действительных чисел, определены арифметические операции в соответствии с тем, как они выполняются цифровой вычислительной машиной. Предположим, что x и y — числа с плавающей точкой. Тогда обычная сумма $x+y$ зачастую уже не принадлежит F. К примеру, возьмем 33-точечную систему, изображенную на рис. 2.1, и положим $x=\frac{5}{4}$ и $y=\frac{3}{8}$. Таким образом, операция сложения, например, сама должна моделироваться в машине посредством приближения, называемого *плавающим сложением*. Если x, $y \in F$ и число $x+y$ не выходит за границы множества F, то идеалом было бы выполнение равенства

$$x \oplus y = fl(x+y),$$

где $\oplus$ обозначает плавающее сложение. В большинстве вычислительных машин этот идеал достигается или почти достигается для всех таких x и y. Поэтому в нашем игрушечном 33-точечном множестве F следует ожидать, что $\frac{5}{4} \oplus \frac{3}{8}$ равно $\frac{3}{2}$ либо $\frac{7}{4}$. Разность между $x \oplus y$ и $x+y$ (для x, y из F) есть ошибка округления, совершенная при плавающем сложении. Аналогичные свойства верны для плавающих операций вычитания, умножения и деления.

Причина, по которой число $\frac{5}{4}+\frac{3}{8}$ не принадлежит 33-точечному множеству F, связана с расположением элементов F. С другой стороны, сумма типа $\frac{7}{2}+\frac{7}{2}$ не принадлежит F потому, что 7 больше максимального элемента из F. Попытка образовать такую сумму вызовет на большинстве машин *сигнал переполнения*, после чего вычисления, как правило, прерываются, поскольку невозможно дать разумное приближение для чисел, выходящих за границы множества F.

В то время как многие суммы $x+y$ (для x, y из F) сами находятся в F, очень редко обычное произведение $x \cdot y$ принадлежит F, поскольку, как правило, оно записывается посредством $2t$ либо $2t-1$ значащих цифр. Кроме того, переполнение гораздо более вероятно при умножении. Наконец, при плавающем умножении возможно *возникновение машинного нуля*, когда два ненулевых числа x и y имеют ненулевое произведение, меньшее по абсолютной величине наименьшего ненулевого элемента из F. (Возникновение машинного нуля возможно и при сложении, хотя это бывает очень редко). Таким образом,

смоделированная операция умножения предполагает округление еще более частое, чем плавающее сложение.

Операции плавающего сложения и умножения коммутативны, но не ассоциативны, и дистрибутивный закон для них также не выполняется. Поскольку эти алгебраические законы имеют фундаментальное значение для математического анализа, анализ плавающих вычислений сложен. Существуют методы анализа ошибок округления, которые обходят некоторые из этих основных трудностей, и посредством таких методов был проанализирован ряд численных алгоритмов, однако этот вопрос выходит за рамки данной книги.

2.2. Вычисление машинного эпсилон

Точность плавающей арифметики можно характеризовать посредством *машинного эпсилон*, т. е. наименьшего числа с плавающей точкой ε, такого, что

$$1 \oplus \varepsilon > 1.$$

Хотя это определение выделяет в качестве машинного эпсилон единственное число плавающей системы, не так уж важно использовать его точное значение. Оно используется обычно в программах таким образом, что может отличаться от своего точного определения несколькими степенями 2, и это не вызывает серьезных затруднений.

Существует несколько методов для вычисления значения (или приближенного значения) ε. Таким образом, программа может определить точность машины, на которой она выполняется, *во время своего исполнения*. Метод, которым мы вычисляем приближение, отличающееся от ε самое большое множителем 2, иллюстрируется следующим сегментом фортранной программы:

```
      EPS = 1.
10    EPS = 0.5*EPS
      EPSP1 = EPS + 1.
      IF (EPSP1 .GT. 1.) GO TO 10
```

2.3. Пример ошибок округления

Важной функцией анализа является показательная функция e^x. Поскольку она используется очень часто, важно иметь возможность вычислять e^x машинной программой для любого числа с плавающей точкой x. Существует множество методов, на которых она может быть основана, и большинство вычисли-

тельных систем научного предназначения имеет подобную программу. Но предположим, что на нашей машине такой программы нет, и спросим себя, как бы мы ее составили. Эта ситуация может возникнуть для более сложной трансцендентной функции либо на новой вычислительной машине.

Вспомним, что для любого действительного (или даже комплексного) значения x число e^x может быть представлено как сумма сходящегося бесконечного ряда

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots .$$

Попробуем воспользоваться этим рядом для вычисления e^x. Предположим, что наша плавающая система характеризуется параметрами $\beta=10$ и $t=5$. Применим ряд для $x=-5.5$. Вот числа, которые мы получим:

$$\begin{aligned} e^{-5.5} = \quad & 1.0000 \\ & -5.5000 \\ & +15.125 \\ & -27.730 \\ & +38.129 \\ & -41.942 \\ & +38.446 \\ & -30.208 \\ & +20.768 \\ & -12.692 \\ & +6.9803 \\ & -3.4902 \\ & +1.5997 \\ & \vdots \\ & +0.0026363 \end{aligned}$$

Мы ограничиваемся при суммировании 25 членами, поскольку последующие слагаемые уже не меняют сумму. Удовлетворительный ли ответ получен? Истинный результат $e^{-5.5}=0.00408677$, так что вычисление посредством ряда привело к ответу, не имеющему верных цифр.

Что же было неправильно? Заметим, что некоторые члены, и следовательно промежуточные суммы, на несколько порядков больше конечного ответа. При этом такие слагаемые, как 38.129, уже содержат ошибку округления, почти равную по величине окончательному результату. Кроме того, четыре старших (т. е. наиболее значимых) разряда каждого из восьми членов, превышающих 10 по модулю, были потеряны. Эти восемь членов следовало бы брать с десятью значащими цифрами, чтобы

получить шесть значащих цифр в ответе. Более того, потребовалась бы еще одиннадцатая цифра, чтобы можно было надеяться получить верную шестую цифру вычисляемой суммы. Это явление иногда называется *катастрофической потерей верных знаков*; оно часто встречается в плохо продуманных вычислениях. Важно, однако, понимать, что такая потеря знаков не является причиной ошибки в ответе; она попросту увеличивает ошибки, уже присутствующие в слагаемых [1]).

Хотя иной раз возможно удерживать в вычислениях большее число значащих цифр с целью избежать катастрофической потери знаков, это всегда обходится дорого как по времени исполнения, так и по памяти, и зачастую требует специальных программных средств. Для рассматриваемой задачи есть гораздо лучшее лекарство: именно, вычислить сумму для $x=5.5$ и затем взять обратное число:

$$\begin{aligned} e^{-5\cdot5} &= \frac{1}{e^{5.5}} \\ &= \frac{1}{1+5.5+15.125+\ldots} \\ &= 0.0040865 \end{aligned}$$

(в нашей пятизначной десятичной арифметике). При таком способе вычисления ошибка снижается до 0.007%.

Один из важных выводов, следующих из этого примера,— это то, что даже короткая последовательность вычислений может быть связана с серьезными ошибками округления.

Заметьте, насколько хуже была бы задача, если бы мы хотели вычислить e^x для $x=-100$.

Реальные машинные алгоритмы для вычисления e^x начинаются с разложения x на действительную и дробную части

$$x=m+f,$$

m — целое, $0\leqslant f\leqslant 1$. Далее, используя свойства показательной функции, получаем

$$e^x = e^{m+f} = e^m \cdot [e^f],$$

или [2])

$$e^x = e^{(1+f/m)\,m} = [e^{1+f/m}]^m.$$

Каждая из величин, взятых в квадратные скобки, предполагает вычисление e^y для y из интервала $0\leqslant y<1$. Ограничение y на такой малый интервал облегчает вычисление e^y. Затем результат может быть умножен на e^m (которое вычисляется просто) или возведен в степень m.

---

[1]) Авторы имеют в виду относительное увеличение ошибки.— *Прим. перев.*

[2]) Это представление применяется при отрицательном m.— *Прим. перев.*

2.4. Неустойчивость некоторых алгоритмов

Задача вычисления $e^{-5.5}$, обсуждавшаяся в предыдущем параграфе, показывает, как плохо продуманный алгоритм может привести к неудовлетворительному результату даже для вполне хорошо поставленной задачи. Трудность была преодолена посредством изменения алгоритма. Для некоторых задач «хорошие» ответы нельзя получить *никаким* алгоритмом, потому что *задача* чувствительна к малым ошибкам, допущенным при представлении данных и в арифметике. Два примера таких задач приведены в § 2.5. Важно различать эти два класса ловушек, поскольку неустойчивые алгоритмы и чувствительные задачи встречаются почти во всех областях вычислительной математики. Однако если вы знаете их симптомы, то диагностировать эти задачи уже довольно просто. В этом параграфе мы обсудим более наглядный пример неустойчивого алгоритма.

Предположим, что мы хотим вычислить интегралы

$$E_n = \int_0^1 x^n e^{x-1}\,dx, \quad n = 1, 2, \ldots .$$

Интегрируя по частям, получим

$$\int_0^1 x^n e^{x-1}\,dx = x^n e^{x-1}\Big|_0^1 - \int_0^1 n x^{n-1} e^{x-1}\,dx,$$

или

$$E_n = 1 - nE_{n-1}, \quad n = 2, \ldots,$$

где $E_1 = 1/e$. Пусть $\beta = 10$ и $t = 6$; мы можем использовать это рекуррентное соотношение для вычисления приближений к девяти первым значениям E_n:

$$\begin{aligned} E_1 &\approx 0.367879, & E_6 &\approx 0.127120, \\ E_2 &\approx 0.264242, & E_7 &\approx 0.110160, \\ E_3 &\approx 0.207274, & E_8 &\approx 0.118720, \\ E_4 &\approx 0.170904, & E_9 &\approx -0.0684800. \\ E_5 &\approx 0.145480, \end{aligned}$$

Хотя подынтегральное выражение $x^9 e^{x-1}$ положительно на всем интервале (0,1), вычисленное нами значение для E_9 отрицательно.

Что вызвало столь большую ошибку? Заметим, что *единственная* ошибка округления, сделанная в приведенных вычисле-

ниях,— это ошибка в E_1, когда $1/e$ округляется до шести значащих цифр [1]). Поскольку рекуррентная формула, полученная интегрированием по частям, точна для действительной арифметики, ошибка в E_9 всецело обязана ошибке округления, сделанной в E_1. Чтобы понять, как ошибка в E_1, примерно равная 4.412×10^{-7}, становится столь большой, заметим, что она умножается на -2 при вычислении E_2, затем ошибка в E_2 умножается на -3 при вычислении E_3 и т. д. Таким образом, ошибка в E_9 есть в точности ошибка в E_1, помноженная на $(-2)(-3)\ldots(-9)=9!$ или примерно $9!\times4.412\times10^{-7}\approx0.1601$. Это огромное увеличение ошибки, содержащейся в исходных данных задачи, есть результат выбранного алгоритма. Подлинное значение E_9 (с тремя значащими цифрами) есть $-0.06848+0.1601=0.0916$.

Как можно было бы избрать другой алгоритм, который не имеет подобной неустойчивости? Если мы перепишем рекуррентное соотношение в виде

$$E_{n-1}=\frac{1-E_n}{n},\quad n=\ldots,3,2,$$

то теперь на каждом шаге вычисления ошибка в E_n умножается на множитель $1/n$. Таким образом, если мы начнем со значения для некоторого E_n при $n\gg1$ и будем вычислять в обратном направлении, то любая начальная ошибка или промежуточные ошибки округлений будут уменьшаться на каждом шаге. Это и называется *устойчивым алгоритмом*. Чтобы найти начальное значение, заметим, что

$$E_n=\int_0^1 x^n e^{x-1}\,dx\leqslant\int_0^1 x^n\,dx=\frac{x^{n+1}}{n+1}\Bigg|_0^1=\frac{1}{n+1}.$$

Итак, E_n стремится к нулю, когда n стремится к ∞. Например, если мы аппроксимируем нулем число E_{20} и возьмем этот нуль стартовым значением, то сделаем начальную ошибку, не превосходящую $1/21$. Эта ошибка умножается на $1/20$ при вычислении E_{19}, так что ошибка в E_{19} не превосходит $(1/20)\times(1/21)\approx0.0024$. Ко времени вычисления E_{15} начальная ошибка уменьшится до величины, меньшей 4×10^{-8}, что в свою очередь меньше единственной ошибки округления. Проводя эти вычисления, получаем

---

[1]) Имеется в виду, что последующие значения для $E_2,\ldots,E_9$ получены округлением результатов, вычисленных точно по содержащему ошибку округления значению для E_1.— *Прим. перев.*

$$E_{20} \approx 0.0, \qquad E_{14} \approx 0.0627322,$$
$$E_{19} \approx 0.0500000, \qquad E_{13} \approx 0.0669477,$$
$$E_{18} \approx 0.0500000, \qquad E_{12} \approx 0.0717733,$$
$$E_{17} \approx 0.0527778, \qquad E_{11} \approx 0.0773523,$$
$$E_{16} \approx 0.0557190, \qquad E_{10} \approx 0.0838771,$$
$$E_{15} \approx 0.0590176, \qquad E_{9} \approx 0.0916123.$$

Когда мы дойдем до E_{15}, начальная ошибка в E_{20} уже совершенно подавлена устойчивостью алгоритма и вычисленные значения от E_{15} до E_9 точны во всех шести значащих цифрах, с точностью до возможной ошибки округления в последней цифре.

2.5. Чувствительность некоторых задач

Мы покажем теперь, что некоторые вычислительные задачи поразительно чувствительны к изменению данных. Этот аспект численного анализа не зависит от плавающей системы или выбранного алгоритма.

Задача нахождения корней полинома является стандартной вычислительной задачей; зачастую она крайне чувствительна к изменению данных. Например, рассмотрим квадратичный полином с почти кратными корнями вроде

$$(x-2)^2 = 10^{-6}.$$

Корни этого уравнения суть 2 ± 10^{-3}. Однако изменение свободного члена на 10^{-6} может вызвать изменение в корнях, равное 10^{-3}. Этот тип неустойчивости еще более выражен у полиномов более высокой степени.

Однако неустойчивость можно наблюдать не только у полиномов с почти кратными корнями. Следующий пример принадлежит Уилкинсону (1963). Пусть

$$p(x) = (x-1)(x-2)\ldots(x-19)(x-20)$$
$$= x^{20} - 210x^{19} + \ldots .$$

Корни $p(x)$ суть 1, 2, ..., 19, 20 и хорошо разделены. Этот пример возник на фоне плавающей системы с параметрами $\beta = 2$, $t = 30$. Чтобы ввести тот или иной коэффициент в вычислительную машину, его необходимо округлить до 30-значного двоичного числа. Предположим, что *лишь для одного* из двадцати коэффициентов делается ошибка в младшем двоичном разряде. Именно, предположим, что коэффициент при x^{19} изменяется

с -210 на $-210+2^{-23}$. Какое воздействие это малое изменение произведет на корни полинома?

Чтобы ответить на этот вопрос, Уилкинсон тщательно вычислил (при $\beta=2$, $t=90$) корни уравнения $p(x)+2^{-23}x^{19}=0$. Приводим их значения, правильно округленные до указанного числа цифр:

1.00000 0000	10.09526 6145 ± 0.64350 0904i
2.00000 0000	11.79363 3881 ± 1.65232 9728i
3.00000 0000	13.99235 8137 ± 2.51883 0070i
4.00000 0000	16.73073 7466 ± 2.81262 4894i
4.99999 9928	19.50243 9400 ± 1.94033 0347i
6.00000 6944	
6.99969 7234	
8.00726 7603	
8.91725 0249	
20.84690 8101	

Заметим, что малое изменение в коэффициенте -210 имело следствием то, что десять корней стали комплексными и что два из них отодвинулись от действительной оси более чем на 2.81. Конечно, полная запись $p(x)$ в машине потребовала бы значительно больше округлений, а вычисление корней сопряжено с еще бóльшими ошибками.

Приведенная выше таблица корней была результатом очень точных вычислений и не пострадала сколько-нибудь заметно от ошибок округления. Причина, по которой эти корни так сильно изменились, лежит не в ошибках округления и не связана с выбором алгоритма вычисления; суть в чувствительности самой задачи. Легко провести анализ того, что произошло. Можно записать полином в виде

$$p(x, \alpha)=x^{20}-\alpha x^{19}+\ldots$$

и затем найти частную производную по α для каждого корня полинома $p(x)$. Это делается дифференцированием уравнения $p(x, \alpha)=0$ по α:

$$\frac{\partial p(x, \alpha)}{\partial x}\frac{\partial x}{\partial \alpha}+\frac{\partial p(x, \alpha)}{\partial \alpha}=0,$$

$$\frac{\partial x}{\partial \alpha}=-\frac{\partial p/\partial \alpha}{\partial p/\partial x}=\frac{x^{19}}{\sum\limits_{i=1}^{20}\prod\limits_{\substack{j=1\\ j\neq i}}^{20}(x-j)}.$$

Вычисляя это выражение для каждого корня, получаем

$$\left.\frac{\partial x}{\partial \alpha}\right|_{x=i}=\frac{i^{19}}{\prod\limits_{\substack{j=1\\ j\neq i}}^{20}(i-j)}, \quad i=1, 2, \ldots, 20.$$

Эти числа дают прямую меру чувствительности каждого из корней к изменению коэффициента α. Приводим их значения:

Корень	$\partial x/\partial\alpha\|_{x=t}$
1	-8.2×10^{-18}
2	8.2×10^{-11}
3	-1.6×10^{-6}
4	2.2×10^{-3}
5	-6.1×10^{-1}
6	5.8×10^{1}
7	-2.5×10^{3}
8	6.0×10^{4}
9	-8.3×10^{5}
10	7.6×10^{6}
11	-4.6×10^{7}
12	2.0×10^{8}
13	-6.1×10^{8}
14	1.3×10^{9}
15	-2.1×10^{9}
16	2.4×10^{9}
17	-1.9×10^{9}
18	1.0×10^{9}
19	-3.1×10^{8}
20	4.3×10^{7}

2.6. Решение квадратных уравнений

Имеется знаменитый алгоритм для решения квадратных уравнений, заключенный в следующей математической теореме:

Теорема. *Если* a, b, c — *действительные числа и* $a \neq 0$, *то уравнение* $ax^2+bx+c=0$ *имеет ровно два решения, именно*

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

и

$$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}.$$

Посмотрим, как работают эти формулы, будучи применены «в лоб» для вычисления x_1 и x_2. На этот раз мы возьмем плавающую систему с $\beta=10$, $t=8$, $L=-50$, $U=50$; она точнее многих широко используемых машинных систем.

Случай 1. $a=1$, $b=-10^5$, $c=1$.

Точные корни соответствующего квадратного уравнения, правильно округленные до 11 значащих десятичных цифр, суть

$$x_1 = 99999.999990,$$
$$x_2 = 0.000010000000001.$$

Если мы воспользуемся формулами теоремы, то вычислим

$x_1 = 100000.00$ (очень хорошо),
$x_2 = 0$ (совершенно неверно).

(Советуем читателю проверить, понимает ли он, как x_2 становится нулем в этом плавающем вычислении.)

Снова, на этот раз при вычислении x_2, мы пострадали от катастрофической потери верных знаков. Имеются различные альтернативные способы вычисления корней квадратичного полинома, которые избегают подобной потери. Одним из таких способов является использование знака b для определения, какая формула влечет меньшую потерю знаков, и затем вычисление одного из корней по этой формуле. Таким образом,

$$x_1 = -\frac{b+\operatorname{sign}(b)\sqrt{b^2-4ac}}{2a}.$$

Поскольку из равенства $ax^2+bx+c=a(x-x_1)(x-x_2)$ следует $ax_1x_2=c$, то второй корень можно вычислить по формуле

$$x_2 = \frac{c}{ax_1}.$$

Для данных случая 1 такой способ дает $x_1=100000.00$, $x_2=1.0000000/100000.00=0.000010000000$; оба результата вполне приемлемы.

Сейчас мы хотели бы предложить следующий критерий для реализации машинного алгоритма решения квадратных уравнений. Он формулируется здесь довольно вольно, однако в статье Форсайта (1969) можно найти строгую формулировку.

Мы скажем, что комплексное число z находится *строго в пределах множества* F, если либо $z=0$, либо

$$\beta^{L+2} \leqslant \operatorname{Re} z \leqslant \beta^{U-2} \text{ и } \beta^{L+2} \leqslant \operatorname{Im} z \leqslant \beta^{U-2}.$$

Это означает, что действительная и мнимая части z находятся строго внутри области чисел, которые могут быть хорошо приближены элементами из F. Довольно произвольный множитель β^2 привлечен для того, чтобы обеспечить некоторый запас надежности.

Предположим, что все три числа a, b, c находятся строго в пределах F. Тогда они должны быть допустимы в качестве

входных данных для алгоритма решения квадратных уравнений. Если $a=b=c=0$, то алгоритм должен заканчивать свою работу сообщением, что все комплексные числа удовлетворяют уравнению $ax^2+bx+c=0$. Если $a=b=0$ и $c\neq 0$, то на выходе алгоритма должна быть информация о том, что ни одно комплексное число не удовлетворяет этому уравнению.

В противном случае пусть z_1 и z_2 — точные корни уравнения, занумерованные так, что $|z_1|\leqslant|z_2|$. (Если $a=0$, положим $z_2=\infty$.) Во всех случаях, когда z_1 находится строго в пределах множества F, алгоритм должен находить хорошее приближение к z_1; под этим понимается требование, чтобы приближение отличалось от z_1 не более чем, скажем, на одну единицу предпоследнего знака корня.

То же самое должно выполняться для z_2.

Если один или два корня не находятся строго в пределах множества F, то должно выдаваться соответствующее сообщение и тот корень, который лежит строго в пределах F (если таковой имеется), должен быть определен с указанной точностью.

Этим завершается вольное описание желаемого исполнения алгоритма для решения квадратных уравнений. Вернемся теперь к рассмотрению некоторых типичных уравнений, чтобы посмотреть, как работают для них квадратичные формулы.

Случай 2. $a=6$, $b=5$, $c=-4$.

Здесь нет никаких затруднений при вычислении $x_1=0.50000000$ и $x_2=-1.3333333$ или приближений к этим значениям, какой бы формулой ни пользоваться.

Случай 3. $a=6\times 10^{30}$, $b=5\times 10^{30}$, $c=-4\times 10^{30}$.

Поскольку, с точностью до множителя 10^{30}, это те же коэффициенты, что и в случае 2, то корни не изменились. Однако, применение формул для x_1 и x_2 вызовет переполнение, так как b^2, будучи больше 10^{50}, находится за пределами F. Скорей всего, эти равномерно большие значения для $|a|$, $|b|$, $|c|$ смогут быть обнаружены до входа в алгоритм, и все три числа будут поделены на какой-нибудь масштабирующий множитель типа 10^{30}, после чего задача сведется к случаю 2.

Случай 4. $a=10^{-30}$, $b=-10^{30}$, $c=10^{30}$.

Здесь x_1 примерно равен 1, в то время как x_2 приблизительно равен 10^{60}. Следовательно, наш алгоритм должен определить x_1 очень точно, несмотря на то что x_2 находится за пределами F. Очевидно, что всякая попытка достигнуть примерного равенства модулей коэффициентов простым делением их на одно и то же число обречена на неудачу и сама по себе может повлечь переполнение или появление машинного нуля. Это уравнение в самом

деле является суровым испытанием для любой программы, решающей квадратные уравнения.

Читатель может подумать, что квадратное уравнение, у которого один корень находится внутри пределов F, а другой — вне их, есть искусственный пример, не имеющий практического значения. В таком случае он ошибается. Во многих итерационных алгоритмах, содержащих подпрограмму решения квадратных уравнений, квадратичные многочлены имеют вырожденное поведение, характеризуемое тем, что $a \to 0$ в случае сходимости алгоритма. Пример такой ситуации — метод, предложенный в статье Мюллера (1956), для нахождения нулей гладкой функции общего вида.

Случай 5. $a=1.0000000$, $b=-4.0000000$, $c=3.9999999$.

Здесь корнями являются $x_1=1.999683772$, $x_2=2.000316228$. Однако применение квадратичных формул дает

$$x_1=x_2=2.0000000,$$

и только первые четыре знака верны. Эти приближения далеко не удовлетворяют выдвинутому выше критерию, однако по сравнению с другими примерами трудность здесь совсем иная.

Данный квадратичный полином соответствует квадратному уравнению $(x-2)^2=e$, где $e=0.0000001$. Как уже говорилось в предыдущем параграфе, это случай чувствительности самого уравнения, а не метода его решения. Однако чувствительные квадратные уравнения все же могут решаться так точно, чтобы удовлетворить нашему критерию, если часть вычислений проводить с повышенной точностью (в данном случае должно быть $t=16$ или еще больше).

В определенном смысле корни, вычисленные в случае 5 ($x_1=x_2=2$), являются «хорошими» решениями этого уравнения. Заметим, что они будут точными корнями уравнения

$$x^2-4x+4=0.$$

Это то же уравнение, что и в случае 5, за исключением свободного члена, отличающегося от c единицей последнего разряда. Таким образом, мы вычислили точные корни уравнения, очень «близкого» к тому, которое пытались решить.

Этот последний способ учета ошибок округления называется *обратным анализом*; он был в большой степени разработан Дж. Уилкинсоном. Говоря общо, такой подход характеризуется следующим вопросом: насколько малые изменения в исходных данных задачи необходимы для того, чтобы представить вычисленные результаты как точное решение возмущенной задачи. В более классическом подходе к ошибкам округления в *прямом анализе* выясняется просто, насколько ошибочны полученные

ответы в качестве решения задачи с ее собственными исходными данными. Хотя оба метода полезны, важной особенностью точки зрения обратного анализа является то, что он позволяет анализировать ошибки округления больших матричных или полиномиальных задач, поскольку допускает использование ассоциативных операций, что часто очень затруднительно в прямом анализе ошибок.

Несмотря на элементарный характер квадратного уравнения, можно с большой долей уверенности сказать, что (по крайней мере на момент написания этой книги) во всем мире имелось не более пяти машинных программ, удовлетворяющих намеченным выше критериям. Создание такого алгоритма не является очень глубокой задачей, но все же оно требует ясного осознания цели и внимания к деталям при достижении этой цели. Вот сорт задач, где младшекурсник математического или вычислительного факультета может внести существенный вклад в машинные библиотеки программ.

УПРАЖНЕНИЯ

2.1. Ряд Тейлора для функции ошибок имеет вид

$$\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{n!\,(2n+1)}.$$

Этот ряд сходится для всех x. Напишите программу для вычисления $\operatorname{erf}(x)$ посредством этого ряда. Возьмите членов ряда ровно столько, чтобы первый отброшенный член не менял накопленной суммы, будучи добавлен к ней в плавающей арифметике. Так как ряд знакопеременный, то ошибка от его обрывания будет в этом случае меньше единственной ошибки округления. Исследуйте влияние ошибок округления, сравнивая вычисленную сумму со значением, взятым из таблицы $\operatorname{erf}(x)$, или значением, полученным по надежной программе. Попробуйте x=0.5, 1.0, 5.0 и 10.0. Объясните ваши результаты.

Указание: внутренний цикл вашей программы может быть таким:

```
10  OLDS = S
    EN = EN + 1.
    T = -XSQ*T*(2.*EN-1.)/(EN*(2.*EN+1.))
    S = S + T
    IF (OLDS .NE. S) GO TO 10
```

Какие значения должны быть присвоены переменным T, S, EN, XSQ перед входом в этот цикл? Не забудьте множитель $2/\sqrt{\pi}$.

2.2. Какой результат будет получен, если пропустить на вашей машине следующую программу? Объясните, почему.

```
      X = 0.0
      H = 0.1
      DO 10 I = 1, 10
         X = X + H
10    CONTINUE
      Y = 1.0 - X
      WRITE(6,20) X, Y
20    FORMAT(2E20.10)
      STOP
      END
```

2.3. Будет ли десятичное число 0.1 точно представимо в вашей машине? Если нет, то каково ближайшее к нему число с плавающей точкой? Каково ближайшее число с плавающей точкой среди чисел, меньших 0.1? Какие значения присваиваются каждому из следующих операторов?

```
      REAL X
      DOUBLE PRECISION Y
      X = 0.1
      Y = 0.1
      Y = 0.1D0
      X = 1.0/10.0
      Y = 1.0D0/10.0D0
      X = 1.0E-1
      Y = 1.0D-1
      READ(5,10) X, Y
10    FORMAT(E5.1, D5.1)
Данные: 0.1  0.1
```

2.4. а) Будет ли число z=314 159 265 358 точно представимо в вашей машине 1) как целое число, 2) как число с плавающей точкой обычной точности 3) как число с плавающей точкой двойной точности?

б) Каково ближайшее к z число с плавающей точкой обыкновенной точности? Какое значение хранится на самом деле в различных системах программирования, когда действительной переменной присваивается значение z?

в) Каково ближайшее к z число с плавающей точкой двойной точности? Какое значение хранится на самом деле в различных системах программирования, когда переменной с двойной точностью присваивается значение z?

г) Найдите косинус угла в 314 159 265 358 радиан с ошибкой, не превосходящей 10^{-9}.

д) Что произойдет, если вы станете вычислять cos $(3.14159265358*10^{11})$ в различных системах программирования?

2.5. (Кахан) а) Как представляются во внутреннем коде вашей машины числа $^1/_2$, $^2/_3$ и $^3/_5$? Используйте соответствующую запись, т. е. двоичную, восьмеричную, шестнадцатиричную, и т. д. Как представляются эти числа в плавающих системах других машин, таких, как IBM 360, CDC 6600, Univac 1108, Honeywell 6000, PDP-11, Burroughs 6500 и т. д.?

б) Прочтите следующую фортран-программу:

```
      H = 1./2.
      X = 2./3. - H
      Y = 3./5. - H
      E = (X+X+X) - H
      F = (Y+Y+Y+Y+Y) - H
      Q = F/E
      WRITE (6,10) Q
      STOP
10    FORMAT(1H, G20.10)
      END
```

Переменная Q может получить несколько различных значений в зависимости от аппаратной реализации плавающей арифметики на той или иной машине. Попробуйте предугадать значение Q для тех машин, с которыми вам приходилось иметь дело. Пропустите программу на стольких машинах, на скольких сможете, чтобы проверить свои предсказания. Объясните ваши результаты.

2.6. Прочтите две следующие фортран-программы:

```
      EPS = 1.
10    EPS = EPS/2.
      WRITE (6,20) EPS
20    FORMAT (1H, G20.10)
      EPSP1 = EPS + 1
      IF (EPSP1 .GT. 1.) GO TO 10
      STOP
      END
```

```
      EPS = 1.
10    EPS = EPS/2.
      WRITE (6,20) EPS
20    FORMAT (1H, G20.10)
      IF (EPS .GT. 0.) GO TO 10
      STOP
      END
```

Пропустите эти программы в вашей системе и объясните результаты.

2.7. Каков результат будет получен, если следующую фортран-программу пропустить на различных машинах, с которыми вы знакомы? Попробуйте предсказать ответ до выполнения программы; после выполнения сверьте с вашим предсказанием.

Каков будет эффект от включения описания

```
DOUBLE PRECISION EPS, EPSP1
```

в заголовок программы?

```
      EPS = 1.
      KBIT = 0
   10 EPS = .5*EPS
      KBIT = KBIT+1
      EPSP1 = EPS+1.
      IF ((EPSP1.GT.1.).AND.(EPSP1-EPS.EQ.1.)) GO TO 10
      IF (EPSP1-EPS.EQ.1.) EPS = 2.*EPS
      WRITE (6,20) EPS, KBIT
      STOP
   20 FORMAT(1H ,G20.10,I10)
      END
```

2.8. Рассмотрим трехзначную десятичную плавающую систему F с усечением [1]), числа которой имеют вид

$$x = \pm .\ d_1 d_2 d_3 \times 10^e,$$

где $-100 \leqslant e \leqslant 100$ и $0 \leqslant d_i \leqslant 9$. Пусть F нормализована (т. е. $d_1 \neq 0$ для $x \neq 0$). Число нуль содержится в F и имеет единственное представление $+.000 \times \times 10^{-100}$.

Посредством знака $\oplus$ будем обозначать операцию плавающего сложения. Для всех x и y из F значение $x \oplus y$ определяется как число с плавающей точкой, ближайшее к $x+y$ среди чисел с плавающей точкой, не превосходящих по модулю $x+y$. Знак $\otimes$ будет обозначать операцию плавающего умножения. Если x и y принадлежат F, то значение $x \otimes y$ определяется как число с плавающей точкой, ближайшее к $x \times y$ среди чисел с плавающей точкой, не превосходящих по модулю $x \times y$.

а) Сколько различных действительных чисел могут быть точно представлены в F?

б) Найдите примеры чисел x, y, z из F, показывающие, что нижеследующие равенства, вообще говоря, неверны, даже если результат находится в границах множества F:

$$(1)\quad (x \otimes y) \otimes z = x \otimes (y \otimes z),$$
$$(2)\quad (x \oplus y) \oplus z = x \oplus (y \oplus z).$$

в) Найдите пример, в котором сумма $(x \oplus y) \oplus z$ имеет относительную ошибку по меньшей мере 50%.

2.9. Напишите подпрограмму DIVIDE (E, F, A, B, C, D) в арифметике с двойной точностью; входом подпрограммы являются 4 числа с двойной точностью a, b, c, d, а выходом — два числа с двойной точностью e и f, такие, что

$$e + if = \frac{a+ib}{c+id},$$

где $i^2 = -1$. Возможно, вы захотите включить в подпрограмму что-нибудь вроде параметра ошибки на случай, если знаменатель окажется равен нулю. Постарайтесь предусмотреть вероятные переполнения и машинные нули и, насколько сумеете, избежать их. (Например, не начинайте с вычисления c^2+d^2 и удостоверьтесь, что ваша подпрограмма выдержит предложенный ниже тестовый пример). Обойти *все* возможные опасности — задача слишком трудная для не очень искушенного читателя, но вы должны уметь объяснить остающиеся слабые места в вашей программе.

---

[1]) В оригинале — chopped.— *Прим. перев.*

а) Проверьте вашу подпрограмму вызовом

CALL DIVIDE (E, F, 1.234D38, 2.345D37, 9.876D37, —5.565D36).

б) Для сравнения вычислите то же частное посредством встроенной подпрограммы ФОРТРАНа для деления комплексных чисел с двойной точностью, если таковая имеется на вашей машине.

2.10. Напишите подпрограмму SQROOT (E, F, A, B), которая использует действительную арифметику с двойной точностью и вычисляет один комплексный квадратный корень

$$e+if=\sqrt{a+ib},$$

скажем тот, для которого $e\geqslant 0$, а если $e=0$, то $f\geqslant 0$. Постарайтесь избежать имеющихся здесь ловушек и сравните вашу подпрограмму со встроенной подпрограммой CDSQRT на следующем примере:

$$\sqrt{1.234\times 10^{38}+i2.345\times 10^{37}}.$$

2.11. Вычислите сумму ряда

$$\varphi(x)=\sum_{k=1}^{\infty}\frac{1}{k(k+x)}$$

для $x=0$ *шаг* 0.1 *до* 1.0 с ошибкой, не превышающей $0.5*10^{-8}$.

Замечание: Эта задача требует как человеческого анализа, так и машинной вычислительной мощи, и одно без другого вряд ли приведет к успеху. Прежде всего не тратьте годовой бюджет машинного времени, пытаясь просуммировать ряд «грубой силой». (Сколько времени вам это стоило бы?)

Указание: Используйте соотношение

$$\frac{1}{k(k+1)}=\frac{1}{k}-\frac{1}{k+1}$$

для доказательства того, что $\varphi(1)=1$. Затем представьте разность $\varphi(x)-\varphi(1)$ рядом, который сходится быстрее исходного ряда, определяющего $\varphi(x)$. Вам придется повторить этот трюк, прежде чем вы получите достаточно быстро сходящийся ряд для вычисления $\varphi(x)$. Ссылка: Хемминг (1962), стр. 48—50.

2.12. Эта задача вычисления рядов иллюстрирует невозможность использования современной вычислительной техники без предварительного анализа даже в простых случаях.

Для $|x|<1$ хотим вычислить выражение

$$S(x)=\sum_{k=1}^{\infty}\frac{1}{\sqrt{k^3+x}}-\sum_{k=1}^{\infty}\frac{1}{\sqrt{k^3-x}}$$

с ошибкой, по модулю меньшей, чем $e=3\times 10^{-8}$.

а) Покажите, что каждый ряд сходится.

б) Сколько примерно членов первого ряда потребуется, чтобы просуммировать его с ошибкой, по модулю меньшей e?

в) Предполагая, что каждое слагаемое может быть вычислено за 500 микросекунд, оцените, сколько времени понадобится, чтобы в обоих рядах просуммировать то число членов, которое определено в пункте б).

г) С помощью алгебраических манипуляций перепишите выражение для $S(x)$ так, чтобы его можно было вычислить быстрее.

е) Запрограммируйте ваш метод и вычислите $S(x)$ для двух случаев: $x=0.5$ и $x=0.999999999$. Если удастся, заметьте время счета и сравните его со своей оценкой.

2.13. Во многих случаях аналитическое решение задачи приводит к необходимости вычисления ряда.

Вычислите ряд

$$\sum_{n=1}^{\infty} \frac{1}{n^2+1}$$

с ошибкой, меньшей единицы десятого десятичного значащего разряда.

Указание:

$$\sum_{n=1}^{\infty} \frac{1}{n^2}=\frac{\pi^2}{6},$$

$$\sum_{n=1}^{\infty} \frac{1}{n^4}=\frac{\pi^4}{90}.$$

Предупреждение: не тратьте понапрасну машинное время, суммируя ряд, в скорости сходимости которого вы не уверены.

2.14. *Задача для самостоятельного исследования*

Напишите фортран-подпрограмму для решения квадратного уравнения

$$ax^2+bx+c=0,$$

где a, b, c — числа с плавающей точкой и с двойной точностью. Алгоритм должен как можно ближе соответствовать идеалам, обсуждавшимся в § 2.6.

Замечание: Существуют две основные трудности:

1. Нужно считаться с возможными переполнениями и машинными нулями. Это очень серьезно главным образом потому, что ФОРТРАН не дает программе пользователя никаких средств для того, чтобы обнаружить машинный нуль или переполнение и предпринять спасательные меры. Подпрограмма должна сама проверять, возможны ли они, и предусматривать предупредительные действия.
2. Даже если переполнения и машинные нули невозможны, есть случаи, когда удовлетворительную точность нельзя получить без вычисления дискриминанта b^2-4ac с точностью, превосходящей рабочую точность арифметики; действительно, если корень должен иметь N верных знаков, то все соответствующие знаки в $\sqrt{b^2-4ac}$ также должны быть верны. Это может привести вас к желанию вычислять b^2-4ac с учетверенной точностью. Для этого требуется несколько особое программирование. В статье Деккера (1971) имеются на этот счет полезные программистские предложения.

3. ЛИНЕЙНЫЕ СИСТЕМЫ УРАВНЕНИЙ

Одна из задач, наиболее часто встречающихся в научных вычислениях,— решение системы линейных уравнений; при этом обычно число уравнений равно числу неизвестных. Такую систему можно записать в виде

$$Ax=b,$$

где A — заданная квадратная матрица порядка n, b — заданный вектор-столбец с n компонентами и x — неизвестный вектор-столбец с n компонентами.

Источники линейных уравнений включают аппроксимацию непрерывных дифференциальных или интегральных уравнений конечными, дискретными алгебраическими системами, локальная линеаризация систем нелинейных уравнений, построение полиномов или кривых какого-либо иного специального вида по заданной информации. Некоторые из этих приложений будут обсуждены в последующих главах.

Лица, изучающие линейную алгебру, знакомятся с методами решения невырожденных систем линейных уравнений. Метод, который часто приводится в таких курсах,— это правило Крамера, согласно которому все компоненты решения представляются отношениями определителей с различными числителями и общим знаменателем. Если бы вы попробовали решить систему из 20 уравнений с помощью правила Крамера, то вам потребовалось бы вычислить 21 определитель порядка 20. Определитель матрицы $A=(a_{ij})$ обычно вводится как сумма членов вида $\pm a_{1-}a_{2-}\ \ldots\ a_{n-}$, где пропущенные индексы нужно заполнить произвольной перестановкой целых чисел $1, \ldots, n$. В данном случае в сумме имеется 20! членов и каждый требует 19 умножений. Следовательно, решение линейной системы предполагает $21\times 20!\times 19$ умножений плюс примерно такое же число сложений. На современной быстродействующей вычислительной машине можно выполнить около 100 000 умножений в секунду. Таким образом, только одни умножения потребуют приблизительно

3×10^8 лет при условии, что машина не сломается в процессе вычислений. Имеются гораздо лучшие способы вычисления определителей; однако при хорошем выборе метода линейную систему возможно решить примерно за то же время, которого требует вычисление одного определителя. Кроме того, правило Крамера зачастую ведет к чрезмерным ошибкам округлений.

Лица, изучающие линейную алгебру, узнают также, что решение системы $Ax=b$ можно представить в виде $x=A^{-1}b$, где A^{-1} — матрица, обратная для A. Однако в огромном большинстве практических вычислительных задач вовсе не обязательно и даже нежелательно в действительности находить A^{-1}. В качестве крайнего, но поучительного примера рассмотрим систему, состоящую ровно из одного уравнения; например,

$$7x=21.$$

Наилучший способ решения этой задачи — деление:

$$x=\frac{21}{7}=3.$$

Использование обратной матрицы привело бы к

$$\begin{aligned} x &= 7^{-1}\cdot 21 \\ &= (.142857)\,(21) = 2.99997. \end{aligned}$$

Второй способ требует больше арифметики — деление и умножение вместо одного деления — и дает менее точный результат. Однако лишние действия — это главная причина, по которой мы рекомендуем избегать обращения. Все сказанное справедливо и для систем с многими уравнениями. Это же остается верным и в распространенной ситуации, когда имеется несколько систем уравнений с одной и той же матрицей A, но различными правыми частями b. Вследствие этого обратим главное внимание на прямое решение систем уравнений, а не на вычисление обратной матрицы.

Важно различать два типа матриц:

1. *Хранимая матрица*, т. е. матрица, все n^2 элементов которой хранятся в оперативной памяти машины. Это ограничивает порядок n примерно значением 100 для машин средней мощности и несколькими сотнями на больших машинах.

2. *Разреженная матрица*, т. е. матрица, большинство элементов которой — нули, а ненулевые элементы могут или храниться посредством какой-либо специальной структуры данных или регенерироваться по мере необходимости. (Матрицы этого типа часто происходят из конечно-разностных и конечно-элементных методов для дифференциальных уравнений с частными производными.) Порядок n зачастую достигает нескольких десятков тысяч, а иногда и того больше.

Приведем пример разреженной матрицы, элементы которой легко могут быть восстановлены:

$$\begin{pmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix}.$$

Эти два типа матриц в известной степени пересекаются. Хранимая матрица может иметь много нулевых элементов и, следовательно, быть в то же время разреженной; однако, если для нулевых элементов отводится место в оперативной памяти, то разреженность матрицы не важна. Очень большая неразреженная матрица может храниться во внешней памяти, на диске или ленте, и вследствие этого требовать более изощренных приемов обработки. *Ленточная матрица* — это матрица, все ненулевые элементы которой находятся вблизи главной диагонали; точнее, $a_{ij}=0$ для всех i, j, таких, что $|i-j|>m$, где $m \ll n$. *Шириной ленты* называется число $2m+1$, и ненулевые элементы размещаются на $2m+1$ диагоналях. Матрица, показанная выше, имеет ширину ленты 3 и называется *трехдиагональной матрицей*.

Некоторые численные методы, применяемые для хранимых матриц, весьма отличаются от методов, приспособленных для разреженных матриц. Методы для хранимых матриц более универсальны, и им будет уделено основное внимание. Эти методы можно модифицировать таким образом, чтобы обрабатывать матрицы, находящиеся во внешней памяти, ленточные матрицы и другие типы больших или умеренно разреженных матриц.

3.1. Линейные системы с хранимыми матрицами

В этом параграфе мы обсудим решение линейной системы алгебраических уравнений

$$Ax=b$$

с хранимой $n \times n$-матрицей A и векторами b и x порядка n. Будем предполагать, что A — невырожденная матрица. Если A вырождена, то с точки зрения теории это обнаружится в процессе вычислений, хотя на практике установить вырожденность матрицы может быть нелегко.

Почти всегда применяемый алгоритм, являющийся одним из старейших численных методов,— это метод последовательного исключения неизвестных, называемый обычно именем Гаусса. Исследования, проведенные в период с 1955 по 1965 год, выявили важность двух аспектов гауссова исключения, не оцененных в более ранних работах: выбора ведущих элементов и надлежащей интерпретации влияния ошибок округления.

Гауссово исключение и другие аспекты матричных вычислений подробно разбираются в книгах Форсайта, Молера (1967) и Стьюарта (1973). Читатель, желающий иметь больше информации, чем мы сообщаем в этой главе, должен воспользоваться этими ссылками.

Чтобы проиллюстрировать алгоритм, рассмотрим следующий пример порядка 3:

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}.$$

Это, в сущности запись системы из трех уравнений

$$\begin{aligned} 10x_1-7x_2 &= 7,\\ -3x_1+2x_2+6x_3 &= 4,\\ 5x_1-x_2+5x_3 &= 6. \end{aligned}$$

На первом шаге с помощью первого уравнения x_1 исключается из других уравнений. Это достигается прибавлением первого уравнения, умноженного на 0.3, ко второму уравнению и прибавлением первого уравнения, умноженного на —0.5, к третьему уравнению. Величины 0.3 и —0.5 называются *множителями*.

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}.$$

На втором шаге можно было бы использовать второе уравнение, чтобы исключить x_2 из третьего уравнения. Однако коэффициент при x_2 во втором уравнении — малое число —0.1. Вследствие этого последние два уравнения переставляются. В данном примере это делать не нужно, поскольку здесь нет ошибок округления, но в общем случае такие перестановки очень важны.

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix}.$$

Теперь с помощью второго уравнения можно исключить x_2 из третьего уравнения. Это достигается прибавлением второго

уравнения, умноженного на 0.04, к третьему уравнению. (Каков был бы множитель, если бы уравнения не переставлялись?)

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix}.$$

Последнее уравнение имеет теперь вид

$$6.2x_3=6.2.$$

Решая его, получим $x_3=1$. Это значение подставляется во второе уравнение:

$$2.5x_2+(5)(1)=2.5.$$

Следовательно, $x_2=-1$. Наконец, значения x_3 и x_2 подставляются в первое уравнение:

$$10x_1+(-7)(-1)=7.$$

Следовательно, $x_1=0$. Это решение легко проверить, используя исходные уравнения:

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix}\begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}=\begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}.$$

В общем случае гауссово исключение состоит из двух этапов: *прямого хода* и *обратной подстановки*. Прямой ход состоит из $n-1$ шагов. На k-м шаге кратные k-го уравнения вычитаются из оставшихся уравнений с целью исключить k-е неизвестное. Если коэффициент при x_k «мал», то рекомендуется переставить уравнения перед исключением. Обратная подстановка заключается в решении последнего уравнения относительно x_n, затем предпоследнего уравнения относительно x_{n-1}, и т. д., пока из первого уравнения не будет вычислено x_1.

Алгоритм в целом может быть компактно записан в матричных обозначениях. Для рассмотренного выше примера положим

$$M_1=\begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix}.$$

Тогда

$$M_1A=\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix},\quad M_1b=\begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}.$$

Пусть

$$P_2=\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix},\quad M_2=\begin{pmatrix}1&0&0\\0&1&0\\0&0.04&1\end{pmatrix}.$$

Тогда

$$M_2P_2M_1A=\begin{pmatrix}10&-7&0\\0&2.5&5\\0&0&6.2\end{pmatrix},\quad M_2P_2M_1b=\begin{pmatrix}7\\2.5\\6.2\end{pmatrix}.$$

Главное теперь состоит в том, что вследствие способа выбора матриц M_1, P_2 и M_2 произведение $U=M_2P_2M_1A$ является *верхней треугольной матрицей*, т. е. все ненулевые элементы находятся в верхней правой половине этой матрицы. Для такой матрицы система уравнений

$$Ux=c$$

легко решается обратной подстановкой. Если положить $c=M_2P_2M_1b$, то система $Ux=c$ имеет то же решение, что и исходная система $Ax=b$.

Аналогичные соотношения справедливы и в общем случае. Пусть P_k, $k=1, \ldots, n-1$, обозначает матрицу, полученную из единичной матрицы I той же перестановкой строк, какая применялась к строкам A на k-м шаге исключения. Пусть M_k обозначает матрицу, полученную из единичной матрицы записью в поддиагональные позиции k-го столбца множителей, использованных на k-м шаге. Каждая из матриц P_k является *матрицей перестановки*, каждая из матриц M_k является простой нижней треугольной матрицей. Обозначим через M и U произведения

$$M=M_{n-1}P_{n-1}\ldots M_2P_2M_1P_1,$$
$$U=MA.$$

Тогда U — верхняя треугольная матрица, система $Ux=Mb$ легко решается относительно x и x является также решением исходной системы $Ax=b$.

Матрица M — не обязательно нижняя треугольная, но она является произведением перестановок простых нижних треугольных матриц. Вследствие этого равенство $U=MA$ называется иногда *треугольным разложением матрицы* A. Следует подчеркнуть, что ничего нового здесь не было введено. Треугольная факторизация есть просто гауссово исключение, выраженное в матричных обозначениях. Треугольные множители могут быть также вычислены другими алгоритмами; см. Форсайт, Молер (1967).

Диагональные элементы матрицы U называются *ведущими элементами*; k-й ведущий элемент есть коэффициент при k-м

неизвестном в k-м уравнении на k-м шаге исключения. В нашем примере ведущими элементами являются 10, 2.5 и 6.2.

И вычисление множителей, и обратная подстановка требуют деления на ведущие элементы. Поэтому алгоритм не может быть проведен, если какой-либо из ведущих элементов равен нулю. Интуиция должна подсказать нам, что продолжать вычисления, в ходе которых некоторые из ведущих элементов оказались близки к нулю,— не слишком удачная идея. Чтобы убедиться в этом, изменим слегка наш пример так, чтобы получилась система

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 3.901 \\ 6 \end{pmatrix}.$$

Значение 2.000 элемента (2,2) матрицы изменено на 2.099, и правая часть также была изменена, чтобы вектор (0, —1, 1) по-прежнему был точным ответом. Предположим, что решение вычисляется на гипотетической машине, которая имеет десятичную плавающую арифметику с пятью значащими цифрами.

Первый шаг исключения дает

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -1.0\times 10^{-3} & 6 \\ 0 & 2.5 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 7 \\ 6.001 \\ 2.5 \end{pmatrix}.$$

Элемент (2,2) теперь очень мал в сравнении с прочими элементами матрицы. Тем не менее закончим исключение, не прибегая к перестановкам. Следующий шаг требует прибавления второго уравнения, умноженного на 2.5×10^3, к третьему. Для правых частей это означает, что нужно умножить 6.001 на 2.5×10^3. Результат равен 1.50025×10^4, и он не может быть представлен точно в нашей гипотетической плавающей системе. Он должен быть или усечен до 1.5002×10^4 или округлен до 1.5003×10^4. Полученное число прибавляется затем к 2.5. Предположим, что используется арифметика с усечением. Тогда последнее уравнение принимает вид

$$1.5005\times 10^4 x_3=1.5004\times 10^4,$$

откуда

$$x_3=\frac{1.5004\times 10^4}{1.5005\times 10^4}=0.99993.$$

Поскольку точный ответ — это $x_3=1$, кажется, что мы не сделали серьезных ошибок. К сожалению, однако, x_2 приходится определять из уравнения

$$-1.0\times 10^{-3}x_2+(6)(0.99993)=6.001,$$

что дает

$$x_2 = \frac{1.5 \times 10^{-3}}{-1.0 \times 10^{-3}} = -1.5.$$

Наконец, x_1 определяется из первого уравнения

$$10x_1 + (-7)(-1.5) = 7,$$

и это приводит к

$$x_1 = -0.35.$$

Вместо (0, —1,1) мы получили (—0.35, —1.50, 0.99993).

Где же была допущена ошибка? Здесь не было «накопления ошибок округления», вызываемого выполнением тысяч арифметических операций. Матрица не близка к вырожденной. Трудности возникли в связи с выбором малого ведущего элемента на втором шаге исключения. Как результат этого множитель равен 2.5×10^3, и последнее уравнение имеет коэффициенты, превосходящие в 10^3 раз по величине коэффициенты исходной задачи. Ошибки округления, малые по отношению к этим большим коэффициентам, неприемлемы с точки зрения уровня элементов исходной матрицы и действительного решения.

Предоставляем читателю проверить, что если второе и третье уравнения переставляются, то больших множителей не возникает и конечный результат будет удовлетворительным. Это оказывается верным и в общем случае: если множители не превосходят 1 по абсолютной величине, то можно показать, что вычисленное решение достаточно точно.

Обеспечить, чтобы множители по модулю были не больше 1, можно посредством процесса, известного как *частичный выбор ведущего элемента*: на k-м шаге прямого хода в качестве ведущего берется наибольший (по абсолютной величине) элемент в неприведенной части k-го столбца. Строка, содержащая этот элемент, переставляется с k-й строкой с тем, чтобы перевести ведущий элемент в позицию (k, k). Такие же перестановки должны производиться с элементами правой части b. Неизвестные в векторе x не переупорядочиваются, поскольку столбцы в A не переставлялись.

Если еще до начала исключения строка или столбец в A умножаются на некоторый масштабирующий множитель, то легко проделать компенсирующее изменение вычисленного решения. Подобные масштабирующие множители можно выбрать так, чтобы в процессе масштабирования не возникало ошибок округления, а тогда точное решение линейной системы не изменится. Однако масштабирование матрицы может полностью изменить выбор ведущих элементов и соответствующие перестановки и, следовательно, изменить точность вычисленного решения. Обычная практика заключается в том, чтобы *уравновесить* матрицу

таким образом, что максимальные элементы каждой строки и каждого столбца будут примерно одной величины. Однако в этом процессе есть определенные трудности. Рассмотрим матрицу

$$A=\begin{pmatrix} 1 & 1 & 1 \\ 10^9 & -1 & 1 \\ 10^9 & 1 & 0 \end{pmatrix}.$$

Деля первый столбец A на 10^9, получаем

$$B=\begin{pmatrix} 10^{-9} & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

С другой стороны, деля последние две строки A на 10^9, приходим к матрице

$$C=\begin{pmatrix} 1 & 1 & 1 \\ 1 & -10^{-9} & 10^{-9} \\ 1 & 10^{-9} & 0 \end{pmatrix}.$$

В то время как обе матрицы уравновешены, B будет хорошо вести себя в алгоритме с выбором ведущего элемента, а C даст вырожденную матрицу на любой машине, работающей менее чем с девятью значащими десятичными цифрами. (Проверьте это.)

Задачи этого типа обычно конструируются с целью иллюстрации и редко возникают на практике. Когда же такая ситуация действительно встречается, то задача или алгоритм, производящий матрицу, должны быть проанализированы более тщательно, прежде чем решать линейную систему. Проблема построения гарантированного масштабирующего алгоритма пока еще не решена. Поэтому мы решили принимать исходную матрицу такой, какая она есть, и не производить никакого масштабирования.

Ошибки округлений, совершенных в процессе вычислений, почти всегда приводят к тому, что вычисленное решение (которое мы будем теперь обозначать через x_*) в определенной степени отличается от теоретического решения $x=A^{-1}b$. В действительности оно и *должно* отличаться, поскольку компоненты вектора x обычно не являются числами с плавающей точкой.

Имеются две общепринятые меры отклонения для x_*: ошибка

$$e=x-x_*$$

и *невязка* [1])

$$r=b-Ax_*.$$

[1]) В дальнейшем авторы употребляют термин «невязка» как для определенного здесь вектора, так и для его компонент.— *Прим. перев.*

Теория матриц говорит, что если одна из этих величин равна нулю, то и другая также должна быть равна нулю. Но они не обязаны одновременно быть «малыми».

Рассмотрим такой пример:

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}.$$

Что случится, если провести гауссово исключение с частичным выбором на гипотетической трехзначной десятичной машине с усечением? Прежде всего строки (уравнения) будут переставлены так, чтобы 0.913 стало ведущим элементом. Затем вычисляется множитель

$$\frac{0.780}{0.913}=0.854 \text{ (с тремя знаками).}$$

Далее новая первая строка, умноженная на 0.854, вычитается из новой второй строки, что приводит к системе

$$\begin{pmatrix} 0.913 & 0.659 \\ 0 & 0.001 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} 0.254 \\ 0.001 \end{pmatrix}.$$

Наконец, выполняется обратная подстановка

$$x_2=\frac{0.001}{0.001}=1.00 \text{ (точный результат),}$$

$$x_1=\frac{[0.254-0.659x_2]}{0.913}$$

$$=-0.443 \text{ (с тремя знаками).}$$

Итак, вычисленное решение есть

$$x_*=\begin{pmatrix} -0.443 \\ 1.000 \end{pmatrix}.$$

Если бы мы не знали правильного ответа, то с целью оценить полученную точность могли бы (точно) вычислить невязку:

$$r=b-Ax_*=\begin{cases} 0.217-[(0.780)(-0.443)+(0.563)(1.00)] \\ 0.254-[(0.913)(-0.443)+(0.659)(1.00)] \end{cases}$$

$$=\begin{pmatrix} -0.000460 \\ -0.000541 \end{pmatrix}.$$

Все *невязки* меньше, чем 10^{-3}. Едва ли можно ожидать лучшего на трехзначной машине. Однако легко видеть, что точное решение этой системы есть

$$x=\begin{pmatrix} 1.000 \\ -1.000 \end{pmatrix}.$$

Таким образом, *ошибка* больше, чем решение.

Были ли малые невязки счастливой случайностью? Прежде всего, читатель уже начал понимать к настоящему моменту, что приведенный пример в высшей степени искусствен. Матрица невероятно близка к вырожденной и *не* является типичной для большинства задач, встречаемых в практике. Тем не менее проследим причины малости невязок.

Если гауссово исключение с частичным выбором выполнить для этого примера на машине с шестью или более разрядами, то прямой ход приведет к системе такого вида:

$$\begin{pmatrix} 0.913000 & 0.659000 \\ 0 & 0.000001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.254000 \\ -0.000001 \end{pmatrix}.$$

Заметим, что *знак* b_2 отличается от полученного при трехзначном вычислении. Теперь обратная подстановка дает

$$x_2 = \frac{0.000001}{0.000001} = -1.00000,$$

$$x_1 = \frac{0.254 - 0.659 x_2}{0.913} = 1.00000,$$

что совпадает с точным ответом. На нашей трехзначной машине x_2 было получено как частное двух величин, каждая из которых имела тот же порядок, что и ошибки округления, а одна не имела даже верного знака [1]). Следовательно, x_2 могло оказаться почти произвольным числом. (И действительно, если бы мы использовали машину с девятью двоичными разрядами, то получили бы совершенно другое значение.) Затем это вполне произвольное значение x_2 подставляется в первое уравнение для нахождения x_1. У нас есть все основания ожидать, что невязка первого уравнения будет мала: x_1 вычисляется таким образом, чтобы гарантировать это. Теперь мы подходим к тонкому, но очень важному пункту. Именно, можно также ожидать, что и невязка второго уравнения будет мала *как раз потому, что матрица столь близка к вырожденной*. Каждое из двух уравнений почти кратно другому, так что любая пара (x_1, x_2), которая почти удовлетворяет первому уравнению, будет в то же время почти удовлетворять второму. Если бы матрица была в точности вырожденной, то второе уравнение нам вовсе не потребовалось бы — любое решение первого автоматически удовлетворяло бы второму.

Хотя приведенный пример сконструирован и нетипичен, вывод, который мы из него сделали, таковым не является. Вероятно, это наиболее важный факт, который был установлен за прошедшие 15 или 20 лет людьми, связанными с матричными вычислениями: *гауссово исключение с частичным выбором ведущего элемента гарантированно дает малые невязки.*

[1]) Имеется в виду знак числа, а не значащая цифра.— *Прим. перев.*

Теперь, сформулировав это утверждение в столь сильной форме, сделаем пару уточняющих замечаний. Говоря «гарантированно», мы подразумеваем, что можно доказать точную теорему, предполагающую некоторые технические детали относительно аппаратной реализации плавающей арифметики и устанавливающую определенные неравенства, которым должны удовлетворять компоненты невязки. Если арифметическое устройство работает каким-то иным образом или в данной программе имеется дефект, то, разумеется, «гарантия» отсутствует. Далее, под «малостью» мы имеем в виду порядок ошибок округления *по отношению* к величинам следующих трех видов: элементам исходной матрицы коэффициентов, элементам матриц на промежуточных шагах процесса исключения и компонентам вычисленного решения. Если какие-либо из этих величин «большие», то невязка не обязательно будет мала в абсолютном смысле. Наконец, даже если невязка мала, мы не станем делать отсюда вывода, что и ошибка мала.

Связь между величиной невязки и величиной ошибки определяется отчасти характеристикой, называемой *числом обусловленности* матрицы, которое является предметом следующего параграфа.

3.2. Обусловленность матрицы

Коэффициенты матрицы и правой части системы линейных уравнений редко бывают известны точно. Некоторые системы возникают из эксперимента, и тогда коэффициенты подвержены ошибкам наблюдения. Коэффициенты других систем записываются формулами, что влечет ошибки округлений при их вычислении. Даже если систему можно точно записать в память машины, в ходе ее решения почти неизбежно будут сделаны ошибки округлений. Можно показать, что ошибки округлений в гауссовом исключении имеют то же влияние на ответ, что и ошибки в исходных коэффициентах.

Вследствие этого мы подходим к фундаментальному вопросу. Если в коэффициентах системы линейных уравнений делаются ошибки, то как сильно при этом меняется решение? Или, другими словами, если $Ax=b$, то как можно измерить чувствительность x по отношению к изменениям в A и b?

Ответ на этот вопрос лежит в уточнении понятия «*почти вырожденная*». Если A — вырожденная матрица, то для некоторых b решение x не существует, тогда как для других b оно будет неединственным. Таким образом, если A почти вырождена, то можно ожидать, что малые изменения в A и b вызовут очень большие изменения в x. С другой стороны, если A — единичная

матрица, то b и x — один и тот же вектор. Следовательно, если A близка к единичной матрице, то малые изменения в A и b должны влечь за собой соответственно малые изменения в x.

На первый взгляд может показаться, что есть некоторая связь между величиной ведущих элементов в гауссовом исключении с частичным выбором и *близостью к вырожденности*, поскольку если бы арифметику можно было выполнять точно, то все ведущие элементы были бы отличны от нуля тогда и только тогда, когда матрица не вырождена. До некоторой степени верно также, что если ведущие элементы малы, то матрица *близка к вырожденной*. Однако при наличии ошибок округления обратное уже неверно [1]) — матрица может быть близка к вырожденной даже если ни один из ведущих элементов не мал.

Чтобы получить более точную и надежную меру близости к вырожденности, нам потребуется ввести понятие *нормы* вектора. Норма — это число, которое измеряет общий уровень элементов вектора. Наиболее употребительной векторной нормой является евклидова длина

$$\left(\sum_{i=1}^{n} |x_i|^2\right)^{1/2}.$$

Однако использование этой нормы сделало бы слишком трудоемкими некоторые из наших вычислений. Вместо нее мы определим в этой главе норму вектора из n компонент следующим образом:

$$\|x\| = \sum_{i=1}^{n} |x_i|.$$

Эта норма обладает многими из аналитических свойств евклидовой длины, именно

$$\begin{aligned}
&\|x\| > 0, && \text{если } x \neq 0,\\
&\|0\| = 0,\\
&\|cx\| = |c| \cdot \|x\| \text{ для всех скаляров } c,\\
&\|x+y\| \leqslant \|x\| + \|y\|.
\end{aligned}$$

Некоторые из геометрических свойств евклидовой длины теряются, но они не слишком важны здесь.

Умножение вектора x на матрицу A приводит к новому вектору Ax, норма которого может очень отличаться от нормы вектора x. Это изменение нормы прямо связано с той чувствительностью, которую мы хотим измерять. Область возможных из-

[1]) Это неверно и в отсутствие ошибок округления.— *Прим. перев.*

менений может быть задана двумя числами

$$M=\max_x \frac{\|Ax\|}{\|x\|},$$

$$m=\min_x \frac{\|Ax\|}{\|x\|}.$$

Максимум и минимум берутся по всем ненулевым векторам. Заметим, что если A вырождена, то $m=0$. Отношение M/m называется *числом обусловленности* матрицы A,

$$\operatorname{cond}(A)=\frac{\max_x \frac{\|Ax\|}{\|x\|}}{\min_x \frac{\|Ax\|}{x}}.$$

Рассмотрим систему уравнений

$$Ax=b$$

и другую систему, полученную изменением правой части:

$$A(x+\Delta x)=b+\Delta b.$$

Будем считать Δb ошибкой в b, а Δx — соответствующей ошибкой в x, хотя нам нет необходимости предполагать, что ошибки малы. Поскольку $A(\Delta x)=\Delta b$, то определения M и m немедленно ведут к неравенствам

$$||b||\leqslant M||x||$$

и

$$||\Delta b||\geqslant m||\Delta x||.$$

Следовательно, при $m\neq 0$

$$\frac{\|\Delta x\|}{\|x\|}\leqslant \operatorname{cond}(A)\frac{\|\Delta b\|}{\|b\|}.$$

Величина $||\Delta b||/||b||$ есть *относительное* изменение правой части, а величина $||\Delta x||/||x||$ — *относительная* ошибка, вызванная этим изменением. Использование относительных изменений имеет то преимущество, что они *безразмерны*, т. е. нечувствительны к общим масштабирующим множителям [1]).

Полученное неравенство показывает, что число обусловленности выполняет роль множителя в увеличении относительной ошибки. Изменения правой части могут повлечь за собой изменения в решении, большие в $\operatorname{cond}(A)$ раз. Оказывается, что то же самое справедливо в отношении изменений в коэффициентах матрицы.

---

[1]) То есть не меняются при умножении вектора и его возмущения на одно и то же число.— *Прим. перев.*

Число обусловленности является также мерой близости к вырожденности. Хотя мы не имеем еще математических средств, необходимых для точной формулировки этого утверждения, можно рассматривать число обусловленности как величину, обратную к относительному расстоянию от данной матрицы до множества вырожденных матриц.

Некоторые из основных свойств числа обусловленности выводятся просто. Ясно, что $M \geqslant m$ и потому

$$\operatorname{cond}(A) \geqslant 1.$$

Если P — матрица перестановок, то компоненты вектора Px лишь порядком отличаются от компонент вектора x. Отсюда следует, что $\|Px\| = \|x\|$ для всех x и, значит,

$$\operatorname{cond}(P) = 1.$$

В частности, $\operatorname{cond}(I) = 1$. Если A умножается на скаляр c, то и M и m будут умножены на модуль этого скаляра, так что

$$\operatorname{cond}(cA) = \operatorname{cond}(A).$$

Если D — диагональная матрица, то

$$\operatorname{cond}(D) = \frac{\max |d_{ii}|}{\min |d_{ii}|}.$$

Последние два свойства в известной мере объясняют, почему $\operatorname{cond}(A)$ является лучшей мерой близости к вырожденности, чем определитель матрицы A. В качестве крайнего примера рассмотрим диагональную матрицу порядка 100 с числом 0.1 на главной диагонали. Тогда $\det(A) = 10^{-100}$, что обычно считается малым числом. Но $\operatorname{cond}(A) = 1$, и компоненты вектора Ax лишь множителем 0.1 отличаются от соответствующих компонент вектора x. Для линейных систем уравнений такая матрица A ведет себя скорее как единичная, а не как вырожденная матрица.

Следующий пример иллюстрирует понятие числа обусловленности:

$$A = \begin{pmatrix} 4.1 & 2.8 \\ 9.7 & 6.6 \end{pmatrix},$$

$$b = \begin{pmatrix} 4.1 \\ 9.7 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Ясно, что $Ax = b$ и

$$\|b\| = 13.8, \quad \|x\| = 1.$$

Если заменить правую часть на

$$b' = \begin{pmatrix} 4.11 \\ 9.70 \end{pmatrix},$$

то решением будет вектор

$$x' = \begin{pmatrix} 0.34 \\ 0.97 \end{pmatrix}.$$

Пусть $\Delta b = b - b'$ и $\Delta x = x - x'$. Тогда

$$\|\Delta b\| = 0.01, \quad \|\Delta x\| = 1.63.$$

Очень малое возмущение, внесенное нами в b, совершенно изменило x. Действительно, относительные изменения равны

$$\frac{\|\Delta b\|}{\|b\|} = 0.0007246, \quad \frac{\|\Delta x\|}{\|x\|} = 1.63.$$

Так как $\operatorname{cond}(A)$ характеризует максимальное возможное увеличение, то

$$\operatorname{cond}(A) \geqslant \frac{1.63}{0.0007246} = 2249.4.$$

На самом деле выбранные b и Δb как раз и дают максимум, так что для этого примера

$$\operatorname{cond}(A) = 2249.4.$$

Важно понимать, что в данном примере речь шла о точных решениях двух слабо различающихся систем уравнений и что метод, которым были получены эти решения, здесь не важен. Пример конструировался с очень большим числом обусловленности так, чтобы эффект изменений в b был отчетливо выражен; однако, подобного же поведения можно ожидать в любой задаче с большим числом обусловленности.

Предположим, что мы хотим решить систему, в которой $a_{11} = 0.1$, а все остальные элементы в A и b — целые числа, и $\operatorname{cond}(A) = 10^5$. Предположим, далее, что у нас двоичная машина с 27 битами для дробной части и что мы каким-то образом умеем вычислять точное решение для системы, уже записанной в память машины. Тогда единственная ошибка будет связана с двоичным представлением числа 0.1, и тем не менее можно ожидать, что

$$\frac{\|\Delta x\|}{\|x\|} \approx 10^5 \times 2^{-27} \approx 10^{-3}.$$

Другими словами, простой акт записи матрицы коэффициентов в машине может вызвать изменения в третьей значащей цифре компонент правильного решения.

Число обусловленности также играет фундаментальную роль в анализе ошибок округлений, совершенных в процессе гауссова исключения. Предположим, что все элементы и A, и b точно представляются числами с плавающей точкой, и пусть x_* — вектор, составленный из чисел с плавающей точкой, который

получен на выходе подпрограммы решения линейных уравнений типа той, какую мы представим в следующем параграфе. Предположим также, что точная вырожденность матрицы, если она имеет место, не была обнаружена и что не было ни машинных нулей, ни переполнений. Тогда можно установить следующие неравенства:

$$\frac{\|b-Ax_*\|}{\|A\|\cdot\|x_*\|}\leqslant\rho\cdot\beta^{-t},$$

$$\frac{\|x-x_*\|}{\|x_*\|}\leqslant\rho\,\mathrm{cond}\,(A)\,\beta^{-t}.$$

Здесь β — основание системы для представления чисел с плавающей точкой, а t — число разрядов дробной части, так что β^{-t} имеет примерно величину машинного эпсилон, о котором было сказано в гл. 2. Величина ρ ниже определяется более точно, но обычно ее значение не превосходит β.

Первое неравенство говорит, что, как правило, можно рассчитывать на то, что *относительная невязка* будет иметь величину, сравнимую с ошибкой округления, независимо от того, насколько плохо обусловлена матрица. Это обстоятельство было проиллюстрировано примером в предыдущем параграфе. Второе неравенство требует, чтобы A была не вырождена; в него входит точное решение x. Это неравенство вытекает непосредственно из первого и определения $\mathrm{cond}\,(A)$; его смысл — *относительная ошибка* будет мала, если мало число $\mathrm{cond}\,(A)$, но она может быть очень велика, если матрица почти вырождена. В предельном случае, когда A вырождена, но это не было обнаружено в процессе вычислений, первое неравенство все же сохраняет силу, в то время как второе не имеет смысла.

Чтобы точнее определить величину ρ, необходимо ввести понятие матричной нормы и установить некоторые вспомогательные неравенства. Читатели, которые не интересуются такими деталями, могут опустить остальную часть этого параграфа.

Определенная ранее величина M называется нормой матрицы. Обозначение матричной нормы такое же, как и у векторной нормы

$$\|A\|=\max_x\frac{\|Ax\|}{\|x\|}.$$

Вследствие нашего специального определения для $\|x\|$ нетрудно показать, что

$$\|A\|=\max_j\|a_j\|,$$

где a_j — столбцы матрицы A. Если бы мы выбрали евклидову длину вектора, то $\|A\|$ вычислялась бы гораздо труднее. Это обсуждается в гл. 9.

Основной результат в исследовании ошибок округлений в гауссовом исключении принадлежит Дж. Уилкинсону. Он доказал, что вычисленное решение x_* точно удовлетворяет системе

$$(A+E)\,x_* = b,$$

где E — матрица, элементы которой имеют величину порядка ошибок округлений в элементах матрицы A. Встречаются редкие ситуации, когда промежуточные матрицы, полученные в ходе гауссова исключения, имеют много большие элементы, чем A, и накопление ошибок округлений в этих больших матрицах оказывает определенное влияние на полученный результат; можно, однако, ожидать, что если величина ρ определена равенством

$$\frac{\|E\|}{\|A\|} = \rho\beta^{-t},$$

то ρ лишь в редких случаях будет больше, чем β.

Из этого основного результата можно тотчас вывести неравенства, относящиеся к невязке и ошибке в вычисленном решении. Невязка определена как

$$b - Ax_* = Ex_*,$$

и, следовательно,

$$\|b - Ax_*\| = \|Ex_*\| \leqslant \|E\|\,\|x_*\|.$$

В определении невязки участвует произведение Ax_*, так что уместно рассматривать относительную невязку, которая сравнивает норму вектора $b—Ax_*$ с нормами A и x_*. Из приведенных выше неравенств сразу следует, что

$$\frac{\|b - Ax_*\|}{\|A\|\,\|x_*\|} \leqslant \rho\beta^{-t}.$$

Если A не вырождена, то с помощью обратной матрицы ошибку можно записать в виде

$$x - x_* = A^{-1}(b - Ax_*),$$

а тогда

$$\|x - x_*\| \leqslant \|A^{-1}\|\,\|E\|\,\|x_*\|.$$

Проще всего сравнивать норму ошибки с нормой вычисленного решения. Таким образом, относительная ошибка удовлетворяет неравенству

$$\frac{\|x - x_*\|}{\|x_*\|} \leqslant \rho\,\|A\|\,\|A^{-1}\|\,\beta^{-t}.$$

Оказывается, что $\|A^{-1}\| = 1/m$ и потому

$$\operatorname{cond}(A) = \|A\|\,\|A^{-1}\|.$$

Итак,

$$\frac{\|x-x_*\|}{\|x_*\|} \leqslant \rho \operatorname{cond}(A) \beta^{-t}.$$

Реальное вычисление числа $\operatorname{cond}(A)$ предполагает знание A^{-1}. Если a_j — столбцы A, $\tilde{a}_j$ — столбцы A^{-1}, то для векторной нормы, которую мы используем,

$$\operatorname{cond}(A) = \max_j \|a_j\| \cdot \max_j \|\tilde{a}_j\|.$$

Легко вычислить $\|A\|$, однако вычисление $\|A^{-1}\|$ примерно удваивает время, нужное для гауссова исключения. К счастью, точное значение $\operatorname{cond}(A)$ требуется редко. Обычно бывает достаточно любой разумной оценки для него.

Подпрограмма DECOMP, описываемая в следующем параграфе, оценивает число обусловленности матрицы таким образом:

$$\operatorname{cond}(A) \approx \max_j \|a_j\| \frac{\|z\|}{\|y\|},$$

где y и z — векторы, определяемые подпрограммой, такие, что $\|z\|/\|y\| \approx \|A^{-1}\|$. Для этого решаются две системы уравнений

$$A^T y = e,$$
$$Az = y,$$

где A^T — транспонированная для матрицы A, а e — вектор с компонентами ± 1, выбираемый так, чтобы максимизировать рост в процессе обратной подстановки для y.

Эта оценка является лишь нижней границей для действительного числа обусловленности, однако есть определенные теоретические основания ожидать, что это очень точная оценка.

3.3. Подпрограммы DECOMP и SOLVE

Почти в любой машинной библиотеке имеются подпрограммы, основанные на вариантах гауссова исключения с частичным выбором ведущего элемента для решения систем линейных уравнений. Детали реализации разных подпрограмм очень различны. Эти детали могут серьезно отразиться на времени выполнения данной подпрограммы, но они не должны сильно влиять на ее точность, если подпрограмма правильно составлена.

В этом параграфе мы опишем две такие подпрограммы — DECOMP и SOLVE. DECOMP выполняет ту часть гауссова исключения, которая зависит лишь от матрицы. Она сохраняет множители и информацию о ведущих элементах. SOLVE использует эти результаты, чтобы получить решение для произвольной правой части.

DECOMP вычисляет также оценку обусловленности матрицы. Такая оценка является намного более надежной и полезной мерой близости к вырожденности, чем такие величины, как определитель или наименьший ведущий элемент.

Эта оценка дает нижнюю границу для действительной обусловленности, но она вычисляется таким образом, что почти всегда отличается от действительной обусловленности не более чем в n раз, а обычно гораздо ближе к ней. Иными словами, почти для всех матриц DECOMP определяет величину COND, удовлетворяющую неравенствам

$$\frac{\mathrm{cond}\,(A)}{n} \leqslant \mathrm{COND} \leqslant \mathrm{cond}\,(A).$$

В тех случаях, когда COND$<$cond$\,(A)/n$, это число все же правильно оценивает чувствительность решений для большинства правых частей.

Ошибки округлений обычно не дают возможности подпрограмме DECOMP или любой другой подпрограмме гауссова исключения определить, вырождена или нет входная матрица. Если в процессе исключения встретится ведущий элемент, равный точному нулю, то DECOMP присваивает COND значение 10^{32}, чтобы сигнализировать, что обнаружена вырожденность. Число 10^{32} находится между β^t и β^U во всех современных плавающих системах, так что оно больше величины, обратной к машинному эпсилон, и меньше уровня переполнения.

Однако появление нулевого ведущего элемента не обязательно означает, что матрица вырождена, так же как вырожденная матрица не обязательно порождает нулевой ведущий элемент. В действительности самым обычным источником нулевых ведущих элементов являются ошибки в вызывающей программе.

Нужно осознать, что при наличии частичного выбора ведущего элемента *любая* матрица имеет треугольное разложение. На самом деле DECOMP работает даже быстрее, когда встречается нулевой ведущий элемент, поскольку это означает, что соответствующий столбец уже находится в треугольной форме. Единственная трудность с таким элементом состоит в том, что SOLVE будет делить на него в процессе обратной подстановки. Поэтому SOLVE не должна использоваться, если DECOMP присвоила COND значение, много большее, чем β^t.

Некоторые подпрограммы, имеющиеся в машинных библиотеках, включают в себя прием, известный как итерационное улучшение или итерационное уточнение. Это процесс, состоящий в вычислении невязки в арифметике повышенной точности и решении системы уравнений, правой частью которой является эта невязка, чтобы получить поправку к первоначально вычисленному решению. Уточненный результат зачастую имеет меньшую

ошибку, но не обязательно меньшую невязку. Кроме того, величина поправки дает еще одну меру чувствительности решения к ошибкам в исходных данных и при вычислениях.

Мы решили не включать в эту книгу программу итерационного улучшения по нескольким причинам. Во-первых, для большинства приложений решение, полученное без улучшения, является удовлетворительным. Во-вторых, ошибки во входной информации обычно влияют на решение сильнее, чем округления в процессе вычислений. В-третьих, наша оценка обусловленности дает ту же информацию, что и величина поправки. Наконец, что, вероятно, самое важное, наличие и доступность требуемой высокоточной арифметики неодинаковы для разных машин. Универсальная программа решения линейных уравнений, которая эффективно включает в себя итерационное улучшение, не может быть написана на стандартном ФОРТРАНе. На машинах типа IBM 360 в обычных вычислениях, как правило, пользуются удвоенной точностью, так что арифметика еще более высокой точности должна реализовываться специальными подпрограммами.

Чтобы прокомментировать некоторые детали в подпрограммах DECOMP и SOLVE, мы должны напомнить, как фортран-системы хранят матрицы. Если программа содержит оператор

```
DIMENSION A(3, 5),
```

то в памяти будет зарезервировано $3\times5=15$ мест для элементов матрицы A. Они будут храниться в следующем порядке:

$$A(1,1)\quad A(2,1)\quad A(3,1)\quad A(1,2)\quad A(2,2)\quad A(3,2)\quad A(1,3)\ldots$$

Другими словами, элементы каждого *столбца* хранятся подряд. Элементы одной *строки* отделены друг от друга числом мест, равным первому *индексу* в операторе размерности. Это соглашение включено в спецификации Американского национального института стандартов для ФОРТРАНа.

Многие употребительные матричные операции наиболее естественно описываются в терминах строк. Например, в гауссовом исключении кратное одной строки вычитается из другой строки. Реализованные на ФОРТРАНе, такие операции обычно приводят к внутренним циклам, меняющим второй индекс массивов. Вследствие этого возможны два потенциально неблагоприятных воздействия на эффективность программы. Вычисления индексов могут оказаться дорогостоящими, поскольку для них нужна информация, содержащаяся в операторе размерности. Операционным системам, управляющим обменом между *быстродействующей* и *внешней* памятью в ходе вычисления, возможно, придется проделать чрезмерную работу. По этим причинам мы реализовали гауссово исключение несколько необычным образом, так,

что все внутренние циклы меняют первый индекс. Такая реализация для некоторых типов операционных систем может оказаться значительно более эффективной. Относительно деталей см. статью Моулер (1972).

Большинство диалектов ФОРТРАНа (хотя и не все) содержат условие относительно *переменных размерностей* массивов, являющихся параметрами подпрограмм. В основную программу можно включить описание

```
DIMENSION A(50, 50),
```

но в действительности работать с $N \times N$-матрицей, где N меняется от задачи к задаче. Подпрограммы типа DECOMP и SOLVE требуют задания как действительного рабочего порядка N, так и величины 50, содержащейся в операторе размерности, поскольку эта величина дает приращение памяти между последовательными элементами строки. Такая информация о размерности в подпрограммах DECOMP и SOLVE обозначается NDIM.

Подпрограмму DECOMP можно использовать для вычисления определителей. Эта возможность вытекает из трех основных свойств определителей. Прибавление кратного одной строки к другой не меняет определителя. Перестановка двух строк изменяет знак определителя. Определитель *треугольной* матрицы равен попросту произведению ее диагональных элементов. DECOMP использует последнюю компоненту вектора ведущих элементов, чтобы поместить туда значение $+1$, если было произведено четное число перестановок, и значение -1, если нечетное. Чтобы получить определитель, это значение нужно умножить на произведение диагональных элементов выходной матрицы.

Неприятной особенностью вычисления определителей является то, что промежуточные произведения, а зачастую и сам определитель, имеют тенденцию быть очень большими или очень малыми числами и, следовательно, легко могут повести к переполнению или машинному нулю. Одна из простых предупредительных мер — вычисление логарифма абсолютной величины определителя как суммы логарифмов его сомножителей.

Приведенная ниже основная программа демонстрирует использование подпрограмм DECOMP и SOLVE. Заметьте, что заявленная размерность массива A — переменная NDIM — имеет значение 10, в то время как N — действительный порядок матрицы — равно 3. В нашей практике обнаружилось, что люди, впервые пользующиеся этими программами, очень часто ошибаются, неправильно присваивая значение переменной NDIM.

Значения для правых частей устанавливаются лишь после того, как выяснилось, что матрица достаточно хорошо обусловлена, чтобы стоило решать линейную систему. При этом мы уста-

навливаем значения элементов матрицы и правой части именно посредством операторов присваивания лишь для того, чтобы не беспокоиться о формате данных.

Для примера, использованного в § 3.1, на выходе будет получена такая информация:

```
 10.     -7.     0.
 -3.      2.     6.
  5.     -1.     5.
COND=    0.12608E+02
  7.
  4.
  6.
 0.00000
-1.00000
 1.00000
```

```
C  ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ ПОДПРОГРАММ DECOMP
C  И SOLVE

      REAL A(10, 10), B(10), WORK(10), COND, CONDP1
      INTEGER IPVT(10), I, J, N, NDIM
      NDIM=10
      N=3
      A(1, 1)=10
      A(2, 1)=-3
      A(3, 1)=5
      A(1, 2)=-7
      A(2, 2)=2
      A(3, 2)=-1
      A(1, 3)=0
      A(2, 3)=6
      A(3, 3)=5
      DO 1 I=1, N
         WRITE(6, 2) (A(I, J), J=1, N)
    1 CONTINUE
    2 FORMAT(1H, 10F5.0)
      WRITE(6, 8)
      CALL DECOMP(NDIM, N, A, COND, IPVT, WORK)
      WRITE(6, 3) COND
    3 FORMAT(6H COND=, E15.5)
      WRITE(6, 8)
      CONDP1=COND+1
      IF (CONDP1 .EQ. COND) WRITE(6, 4)
    4 FORMAT(40H MATRIX IS SINGULAR TO WORKING
     PRECISION)
      IF (CONDP1 .EQ. COND) STOP
      B(1)=7
      B(2)=4
      B(3)=6
      DO 5 I=1, N
         WRITE(6, 2) B(I)
    5 CONTINUE
```

```
      WRITE (6, 8)
      CALL SOLVE (NDIM, N, A, B, IPVT)
      DO 6 I=1, N
         WRITE (6, 7) B (I)
    6 CONTINUE
    7 FORMAT (1H, F10.5)
      STOP
    8 FORMAT (1H)
      END

      SUBROUTINE DECOMP (NDIM, N, A, COND, IPVT, WORK)
C
      INTEGER NDIM, N
      REAL A (NDIM, N), COND, WORK (N)
      INTEGER IPVT (N)
C
C     ПРОГРАММА ВЫЧИСЛЯЕТ РАЗЛОЖЕНИЕ ВЕЩЕСТВЕННОЙ
C     МАТРИЦЫ ПОСРЕДСТВОМ ГАУССОВА ИСКЛЮЧЕНИЯ И
C     ОЦЕНИВАЕТ ОБУСЛОВЛЕННОСТЬ МАТРИЦЫ.
C
C     ОНА ИСПОЛЬЗУЕТСЯ ДЛЯ ВЫЧИСЛЕНИЯ РЕШЕНИЙ ЛИ-
C     НЕЙНЫХ СИСТЕМ.
C
C     ВХОДНАЯ ИНФОРМАЦИЯ..
C
C        NDIM=ЗАЯВЛЕННАЯ СТРОЧНАЯ РАЗМЕРНОСТЬ МАС-
C        СИВА, СОДЕРЖАЩЕГО A.
C
C        N=ПОРЯДОК МАТРИЦЫ.
C
C        A=МАТРИЦА, КОТОРУЮ НУЖНО РАЗЛОЖИТЬ.
C
C     ВЫХОДНАЯ ИНФОРМАЦИЯ..
C
C        A СОДЕРЖИТ ВЕРХНЮЮ ТРЕУГОЛЬНУЮ МАТРИЦУ U
C          И УЧИТЫВАЮЩУЮ ПЕРЕСТАНОВКИ ВЕРСИЮ
C          НИЖНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ I-L, ТАКИЕ,
C          ЧТО (МАТРИЦА ПЕРЕСТАНОВОК) *A=L*U
C
C        COND=ОЦЕНКА ОБУСЛОВЛЕННОСТИ A.
C          ДЛЯ ЛИНЕЙНОЙ СИСТЕМЫ A*X=B ИЗМЕНЕНИЯ
C          B A И B МОГУТ ВЫЗВАТЬ ИЗМЕНЕНИЯ B X, БОЛЬ-
C          ШИЕ В COND РАЗ. ЕСЛИ COND+1.0.EQ.COND, ТО A
C          В ПРЕДЕЛАХ МАШИННОЙ ТОЧНОСТИ ЯВЛЯЕТСЯ
C          ВЫРОЖДЕННОЙ МАТРИЦЕЙ. COND ПОЛАГАЕТСЯ
C          РАВНЫМ 1.0E+32, ЕСЛИ ОБНАРУЖЕНА ТОЧНАЯ
C          ВЫРОЖДЕННОСТЬ.
C
C        IPVT=ВЕКТОР ВЕДУЩИХ ЭЛЕМЕНТОВ.
C          IPVT (K)=ИНДЕКС K-Й ВЕДУЩЕЙ СТРОКИ
C          IPVT (N)=(-1)** (ЧИСЛО ПЕРЕСТАНОВОК)
C
C     РАБОЧЕЕ ПОЛЕ.. ВЕКТОР WORK ДОЛЖЕН БЫТЬ ОПИСАН
C                    И ВКЛЮЧЕН В ВЫЗОВ. ЕГО ВХОДНОЕ СОДЕР-
C                    ЖАНИЕ ОБЫЧНО НЕ ДАЕТ ВАЖНОЙ ИНФОР-
C                    МАЦИИ.
```

```
C
C     ОПРЕДЕЛИТЕЛЬ МАТРИЦЫ A МОЖЕТ БЫТЬ ПОЛУЧЕН
C     НА ВЫХОДЕ ПО ФОРМУЛЕ
C        DET (A) = IPVT (N) *A (1, 1) *A (2, 2) *. . .*A (N, N).
C
      REAL EK, T, ANORM, YNORM ZNORM
      INTEGER NM1, I, J, K, KP1, KB, KM1, M
C
      IPVT (N) = 1
      IF (N.EQ.1) GO TO 80
      NM1 = N - 1
C
C     ВЫЧИСЛИТЬ 1-НОРМУ МАТРИЦЫ A [1])
C
      ANORM = 0.0
      DO 10 J = 1, N
         T = 0.0
         DO 5 I = 1, N
            T = T + ABS (A (I, J))
    5    CONTINUE
         IF (T .GT. ANORM) ANORM = T
   10 CONTINUE
C
C     ГАУССОВО ИСКЛЮЧЕНИЕ С ЧАСТИЧНЫМ ВЫБОРОМ ВЕ-
C     ДУЩЕГО ЭЛЕМЕНТА
C
      DO 35 K = 1, NM1
         KP1 = K + 1
C
C        НАЙТИ ВЕДУЩИЙ ЭЛЕМЕНТ
C
         M = K
         DO 15 I = KP1, N
            IF (ABS (A (I, K)) .GT. ABS (A (M, K))) M = I
   15    CONTINUE
         IPVT (K) = M
         IF (M .NE. K) IPVT (N) = - IPVT (N)
         T = A (M, K)
         A (M, K) = A (K, K)
         A (K, K) = T
C
C        ПРОПУСТИТЬ ЭТОТ ШАГ, ЕСЛИ ВЕДУЩИЙ ЭЛЕМЕНТ
C        РАВЕН НУЛЮ
C
         IF (T .EQ. 0.0) GO TO 35
C
C        ВЫЧИСЛИТЬ МНОЖИТЕЛИ
C
         DO 20 I = KP1, N
         A (I, K) = - A (I, K)/T
   20    CONTINUE
C
C        ПЕРЕСТАВЛЯТЬ И ИСКЛЮЧАТЬ ПО СТОЛБЦАМ
```

[1]) Для векторной нормы, которую используют авторы, соответствующую матричную норму, определенную в § 3.2, принято помечать индексом 1. — *Прим. перев.*

```
C
      DO 30 J=KP1, N
         T=A(M, J)
         A(M, J)=A(K, J)
         A(K, J)=T
         IF (T . EQ. 0.0) GO TO 30
         DO 25 I=KP1, N
            A(I, J)=A(I, J)+A(I, K)*T
25       CONTINUE
30    CONTINUE
35 CONTINUE
C
C     COND=(1-НОРМА МАТРИЦЫ A) * (ОЦЕНКА ДЛЯ 1-НОРМЫ
C     МАТРИЦЫ, ОБРАТНОЙ К A)
C     ОЦЕНКА ПОЛУЧАЕТСЯ ПОСРЕДСТВОМ ОДНОГО ШАГА МЕ-
C     ТОДА ОБРАТНЫХ ИТЕРАЦИЙ ДЛЯ НАИМЕНЬШЕГО СИН-
C     ГУЛЯРНОГО ВЕКТОРА. ЭТО ТРЕБУЕТ РЕШЕНИЯ ДВУХ
C     СИСТЕМ УРАВНЕНИЙ, (ТРАНСПОНИРОВАННАЯ ДЛЯ A)
C     *Y=E И A*Z=Y, ГДЕ E—ВЕКТОР ИЗ +1 И —1, ВЫБРАН-
C     НЫЙ ТАК, ЧТОБЫ МАКСИМИЗИРОВАТЬ ВЕЛИЧИНУ Y.
C     ESTIMATE=(1-НОРМА Z)/(1-НОРМА Y)
C
C     РЕШИТЬ СИСТЕМУ (ТРАНСПОНИРОВАННАЯ ДЛЯ A)*Y=E
C
      DO 50 K=1, N
         T=0.0
         IF(K . EQ. 1) GO TO 45
         KM1=K—1
         DO 40 I=1, KM1
            T=T+A(I, K)*WORK(I)
40       CONTINUE
45       EK=1.0
         IF (T . LT. 0.0) EK+—1.0
         IF (A(K, K) . EQ. 0.0) GO TO 90
         WORK(K)=—(EK+T)/A(K, K)
50    CONTINUE
      DO 60 KB=1, NM1
         K=N—KB
         T=0.0
         KP1=K+1
         DO 55 I=KP1, N
            T=T+A(1, K)*WORK(K)
55       CONTINUE
         WORK(K)=T
         M=IPVT(K)
         IF (M . EQ. K) GO TO 60
         T=WORK(M)
         WORK(M)=WORK(K)
         WORK(K)=T
60    CONTINUE
C
      YNORM=0.0
      DO 65 I=1, N
         YNORM=YNORM+ABS(WORK(I))
65    CONTINUE
C
C     РЕШИТЬ СИСТЕМУ A*Z=Y
```

```
C
      CALL SOLVE (NDIM, N, A, WORK, IPVT)
C
      ZNORM = 0.0
      DO 70 I = 1, N
         ZNORM = ZNORM + ABS (WORK (I))
   70 CONTINUE
C
CC    ОЦЕНИТЬ ОБУСЛОВЛЕННОСТЬ
C
      COND = ANORM * ZNORM/YNORM
      IF (COND . LT. 1.0) COND = 1.0
      RETURN
C
CC    СЛУЧАЙ МАТРИЦЫ 1×1
C
   80 COND = 1.0
      IF (A (1, 1). NE. 0.0) RETURN
C
CC    ТОЧНАЯ ВЫРОЖДЕННОСТЬ
C
   90 COND = 1.0E + 32
      RETURN
      END
         SUBROUTINE SOLVE (NDIM, N, A, B, IPVT)
C
         INTEGER NDIM, N, IPVT (N)
         REAL A (NDIM, N), B (N)
C
CC    РЕШЕНИЕ ЛИНЕЙНОЙ СИСТЕМЫ A*X = B
CC    ПОДПРОГРАММУ НЕ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ, ЕСЛИ
CC    DECOMP ОБНАРУЖИЛА ВЫРОЖДЕННОСТЬ
CC
CC    ВХОДНАЯ ИНФОРМАЦИЯ..
CC
CC       NDIM = ЗАЯВЛЕННАЯ СТРОЧНАЯ РАЗМЕРНОСТЬ МАС-
CC       СИВА, СОДЕРЖАЩЕГО A.
CC
CC       N = ПОРЯДОК МАТРИЦЫ.
CC
CC       A = ФАКТОРИЗОВАННАЯ МАТРИЦА, ПОЛУЧЕННАЯ ИЗ
CC       DECOMP
CC
CC       B = ВЕКТОР ПРАВЫХ ЧАСТЕЙ.
CC
CC       IPVT = ВЕКТОР ВЕДУЩИХ ЭЛЕМЕНТОВ, ПОЛУЧЕННЫЙ
CC       ИЗ DECOMP
CC
CC    ВЫХОДНАЯ ИНФОРМАЦИЯ..
CC
CC       B = ВЕКТОР РЕШЕНИЯ X.
CC
      INTEGER KB, KM1, NM1, KP1, I, K, M
      REAL T
C
CC    ПРЯМОЙ ХОД
```

```
C
      IF(N .EQ. 1) GO TO 50
      NM1 = N - 1
      DO 20 K = 1, NM1
         KP1 = K + 1
         M = IPVT(K)
         T = B(M)
         B(M) = B(K)
         B(K) = T
         DO 10 I = KP1, N
            B(I) = B(I) + A(I, K) *T
   10    CONTINUE
   20 CONTINUE
C
CC    ОБРАТНАЯ ПОДСТАНОВКА
C
      DO 40 KB = 1, NM1
         KM1 = N - KB
         K = KM1 + 1
         B(K) = B(K)/A(K, K)
         T = - B(K)
         DO 30 I = 1, KM1
            B (I) = B(I) + A(I, K)*T
   30    CONTINUE
   40 CONTINUE
   50 B(1) = B(1)/A(1, 1)
      RETURN
      END
```

3.4. Большие разреженные системы

Методы исключения

Многие линейные алгебраические системы имеют столь большое число n уравнений и неизвестных, что хранить в оперативной памяти полную квадратную матрицу из n^2 элементов невозможно. Обычно такие системы получаются при дискретизации дифференциальных уравнений, обыкновенных или с частными производными, а также при расчете различных конструкций или цепей. Зачастую матрицы таких задач настолько разрежены, что быстродействующей памяти хватает для хранения всех ненулевых элементов вместе с информацией об их расположении. Как же нужно решать соответствующую систему линейных уравнений?

Если применение гауссова исключения возможно, то оно остается очень экономичным, точным и полезным алгоритмом. Исключение возможно, если удается разместить в памяти ненулевые элементы треугольных матриц, строящихся в ходе исключения, вместе с информацией о том, где они помещены. Пусть LU — массив, нижний треугольник которого есть матрица множителей, а верхний треугольник — триангулированная матри-

ца [1]. Тогда LU обычно значительно более заполнен, чем A, хотя и остается все еще разреженным. Говорят, что позиции, соответствующие нулевым элементам A и ненулевым элементам LU, *заполнились* в гауссовом исключении. В принципе возможно предсказать точную степень заполнения, вызываемого исключением, однако детали могут быть столь сложны, что программист отступится от такой возможности.

Для некоторых матриц размер заполнения оценить легко. Примером является ленточная матрица. Договоримся пока рассматривать ленточную матрицу как заполненную, т. е. игнорировать все нули внутри ленты. Если гауссово исключение можно проводить без выбора ведущих элементов, что действительно возможно и безопасно для одного класса матриц, называемых *положительно определенными* (Форсайт, Моулер, (1967)), то заполнения нет вообще: LU имеет ту же ширину ленты, что и A. Если же выбор ведущих элементов необходим (для невырожденной матрицы общего вида), то заполнение ограничено лентой полуторной ширины по сравнению с лентой матрицы A.

Ленточную матрицу A удобно хранить прямоугольным массивом длины n и ширины $2m+1$; более широкий ленточный массив LU также удобно хранить и обрабатывать. Отсюда заключаем: линейные системы с ленточными матрицами легко решаются посредством исключения при условии, что ленточный массив LU можно разместить в быстродействующей памяти. Существует несколько программ для обработки ленточных матриц; они являются сравнительно простыми модификациями таких программ исключения, как DECOMP и SOLVE.

Если необходимо принимать в расчет нули внутри ленточной структуры матрицы для экономии памяти либо сокращения числа арифметических операций, то структура данных и ее обработка могут значительно усложниться. Такая детальная структура может возникать при применении конечно-разностных методов к краевым задачам для эллиптических уравнений в частных производных с переменными коэффициентами. Решение подобных систем посредством гауссова исключения все еще является исследовательской задачей, и универсальные программы этого сорта только начинают создаваться.

Иногда полная или даже ленточная матрица слишком велика, чтобы разместить ее в быстродействующей памяти. В этом случае необходимо часть матрицы хранить во внешней памяти — на дисках или магнитных лентах. Если задача столь велика, то хотя гауссово исключение и возможно, один только объем вычислений делает его очень дорогостоящим. Время выполнения

[1] То есть верхняя треугольная матрица, получаемая прямым ходом гауссова исключения.— *Прим. перев.*

арифметических операций обычно значительно больше, чем время, нужное для операций обмена частями матрицы с внешней памятью. Поэтому в целях экономии важно так организовать ход вычислений либо работу операционной системы, чтобы процессор не простаивал, ожидая завершения обмена. Это можно сделать различными способами. Однако готовых универсальных программ пока не существует, хотя большинство мощных вычислительных установок имели опыт решения таких больших систем.

В последние годы огромные усилия системных программистов были сосредоточены на том, чтобы создать у пользователя иллюзию, будто он располагает очень большой (так называемой *виртуальной*) памятью для своих данных, хотя в действительности эти данные разбиты на *страницы* или *сегменты*, которыми постоянно идет обмен с внешней памятью. Таковы, например, системы машин Burroughs B5500 или некоторых машин фирмы IBM серий 360 и 370. Наличие виртуальной памяти освобождает программиста от забот, связанных с обменом данными. Однако, цена этой свободы может для некоторых стратегий разбиения на страницы оказаться очень высокой: если программа вынуждена ждать, пока механизм замещения отыскивает каждую очередную строку матрицы, как это происходит на машине Burroughs B5500 в пакетном режиме, то время выполнения для большой матрицы может вырасти в недопустимой степени.

Однако, виртуальная память обычно объединена с режимом мультипрограммирования, и во время поиска очередной страницы процессор обычно принимает к исполнению другую программу. Во многих операционных системах время прерывания не записывают на счет пользователя даже в том случае, когда другой программы нет. Следовательно, «стоимость» выполнения матричной программы остается приблизительно той же, был или нет обмен страницами. Однако в любом случае обмен удлиняет время *ожидания* завершения работы программы.

Итерационные методы

Имеется важный класс систем линейных уравнений, для которых элементы матрицы A задаются какой-нибудь простой формулой и, следовательно, могут вычисляться по мере необходимости. Например, при моделировании уравнения Лапласа конечно-разностными методами элементы A обычно равны одному из чисел 0, 1 или —4; какое именно значение принимается, легко определить, исходя из геометрии сетки. В таком случае элементы можно вовсе не хранить, а генерировать их, когда они понадобятся. Кроме того, часто порядки n столь велики, что было бы невозможно хранить заполненный массив LU.

Желательно решать такие линейные системы $Ax=b$ методами, которые вообще не меняют матрицу A и требуют хранения лишь нескольких векторов длины n. (Заметим, что вектор b обычно должен храниться, так же, как и вектор x.)

Методы, отвечающие этим требованиям, существуют и называются *итерационными*. В них начинают с какого-нибудь пробного приближения $x^{(0)}$ и выполняют некоторый процесс, использующий A, b и $x^{(0)}$ и приводящий к новому вектору $x^{(1)}$. Затем процесс повторяют. На k-м шаге итерационного процесса по A, b и $x^{(k-1)}$ получают $x^{(k)}$. При соответствующих предположениях векторы $x^{(k)}$ сходятся к пределу при $k\to\infty$. Имеется множество подобных итерационных процессов. Наиболее успешные из них обязаны своим успехом тесной связи с решаемой задачей. Поэтому редко можно встретить библиотечные подпрограммы итерационных методов. Хотя итерационный процесс математически может быть несложен, структура матрицы A, скорей всего, сложна и специальным образом связана с данной конкретной задачей.

Один простой итерационный метод обсуждается в § 24 книги Форсайт, Моулер (1967); этот метод известен под различными названиями: *процесс Либмана*, метод *Гаусса — Зейделя* или *метод последовательных замещений*. Здесь основной итерационный шаг состоит в том, что i-е уравнение решается относительно i-й компоненты x_i нового вектора x, причем для всех прочих компонент x берутся прежде вычисленные значения. Можно доказать, что метод сходится для некоторых типов матриц, например, для любой симметричной положительно определенной матрицы A. Однако сходимость обычно медленная.

У многих итерационных процессов численного анализа сходимость столь медленная, что очень важная задача — найти способ ускорения сходимости, например, векторов $x^{(k)}$ к решению. И действительно, алгоритмы, ускоряющие сходимость последовательностей, составляют важную часть вычислительной математики. Метод *последовательной сверхрелаксации* (SOR) является подобным алгоритмом для процесса Гаусса — Зейделя. Он может ускорять сходимость в такой степени, что весьма широко используется при решении конечно-разностных уравнений, моделирующих двумерные эллиптические краевые задачи. Опять-таки этого метода нет в библиотеках программ.

Имеется еще семейство итерационных методов, называемых методами *сопряженных градиентов* или *сопряженных направлений*. Хорошее изложение этих алгоритмов можно найти в книге Фаддеев, Фаддеева (1963). Методы этой группы по-видимому довольно успешны для симметричных положительно определенных матриц и не содержат предположений относительно структуры матрицы A. Есть ряд опубликованных алгоритмов метода сопряженных градиентов.

УПРАЖНЕНИЯ

3.1. Найдите решение следующей системы порядка 3:

$$\begin{pmatrix} 1.00 & 0.80 & 0.64 \\ 1.00 & 0.90 & 0.81 \\ 1.00 & 1.10 & 1.21 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \mathrm{erf}\,(0.80) \\ \mathrm{erf}\,(0.90) \\ \mathrm{erf}\,(1.10) \end{pmatrix}.$$

Определение функции erf см. в упр. 1.1. Воспользуйтесь подпрограммами DECOMP и SOLVE. Выведите оценку обусловленности матрицы и решение x_1, x_2, x_3. Напечатайте также сумму $x_1+x_2+x_3$ и сравните ее со значением erf (1.00). Почему эти два числа так близки? Если вы не сумеете ответить на этот вопрос, загляните в § 4.1.

3.2. Что будет получено на выходе следующей программы:

```
      DIMENSION A(10,10),WORK(10),IPVT(10)
      NDIM = 10
      N = 2
      A(1,1) = 3.
      A(1,2) = 6.
      A(2,1) = 1.
      A(2,2) = 2.
      CALL DECOMP(NDIM, N, A, COND, IPVT, WORK)
      WRITE(6,1) A(2,2)
      WRITE(6,1) COND
    1 FORMAT(E16.6)
      STOP
      END
```

а) Объясните подробно, как связано значение элемента A (2,2) после обращения к DECOMP с машинным эпсилон вашей плавающей системы? Объясните в общих чертах, как связано с машинным эпсилон значение COND (Не нужно вспоминать все детали вычисления COND.) Если у вас есть доступ к арифметике другой разрядности, измените программу, включая и DECOMP, и снова ответьте на те же вопросы.

б) Какой результат был бы получен, если бы эту программу пропустили на советской вычислительной машине Сетунь? (См. таблицу в § 2.1).

3.3. Эта задача заключается в проверке двух неравенств

$$\frac{\| b - Ax_* \|}{\| A \| \| x_* \|} \leqslant \rho \beta^{-t},$$

$$\frac{\| x - x_* \|}{\| x_* \|} \leqslant \rho\, \mathrm{cond}\,(A)\, \beta^{-t},$$

где норма вектора x есть число

$$\| x \| = \sum_{i=1}^{n} | x_i |,$$

а норма матрицы со столбцами a_j равна

$$\| A \| = \max_j \| a_j \|.$$

Вы должны экспериментально проверить наши утверждения о том, что в первом неравенстве ρ почти всегда меньше, чем β, и что величина COND, вычисляемая подпрограммой DECOMP, является удовлетворительной оценкой для $\mathrm{cond}(A)$ во втором неравенстве.

Возьмите любую матрицу A, элементы которой точно представляются числами с плавающей точкой. Так как второе неравенство предполагает знание точного решения, то либо возьмите b, для которого вы знаете точный вектор x, либо возьмите x и вычислите $b=Ax$. Удостоверьтесь, что в A, b и x нет ошибок округления, так что равенство $Ax=b$ выполняется точно.

С помощью DECOMP факторизуйте матрицу и вычислите COND. Для вычисления x_* используйте SOLVE. Вычислите $\|A\|$, $\|x_*\|$, $\|b-Ax_*\|$ и $\|x-x_*\|$. Предусмотрите хранение копий A и b, поскольку они изменяются подпрограммами.

Вычислите ρ так, чтобы первое неравенство было в действительности равенством. Если вы обнаружите, что ρ много больше, чем β, внимательно проверьте свою программу. Большие значения ρ теоретически возможны, однако они случаются очень редко. Они связаны с ростом величины элементов матрицы в процессе исключения. Относительно дальнейшей информации см. книгу Уилкинсон (1963).

Используя ваше значение ρ, проверьте, будет ли удовлетворяться второе неравенство, в котором $\mathrm{cond}(A)$ заменено на COND. Если это не так, то причина в том, что COND значительно меньше подлинного числа обусловленности. Опять-таки подобные примеры очень редки.

Выполните это задание для нескольких различных матриц как с числами обусловленности, близкими к 1, так и с очень большими числами обусловленности.

3.4. Обратная к матрице A может быть определена как матрица X, столбцы которой удовлетворяют условиям

$$Ax_j = e_j,$$

где e_j есть j-й столбец единичной матрицы. Напишите подпрограмму с заголовком

SUBROUTINE INVERT (NDIM, N, A, X, COND, IPVT, WORK),

входом которой является матрица порядка N, а выходом — матрица X, приближающая обратную к A, а также оценка обусловленности и информация о ведущих элементах. Ваша подпрограмма должна обращаться к DECOMP только один раз, а к SOLVE всего n раз, по одному вызову для каждого столбца матрицы X. Содержимое X не определено, если DECOMP обнаруживает вырожденность.

Проверьте вашу подпрограмму на некоторых матрицах, элементы которых могут быть точно представлены числами с плавающей точкой и для которых вы знаете A^{-1}. Имеется несколько способов оценки точности результатов:

$$\|AX-I\|,$$
$$\|XA-I\|,$$
$$\|X-A^{-1}\|.$$

Можно было бы также дважды использовать подпрограмму INVERT; первый раз для обращения A и второй раз для обращения X. Результатом будет матрица Z, которая совпала бы с A, не будь округлений. Таким образом, еще одной мерой точности является величина

$$\|Z-A\|.$$

Можете ли вы вывести неравенство, включающее ρ, $\mathrm{cond}(A)$ и β^{-t}, которое оценивает, насколько велико может быть $\|Z-A\|$?

3.5. Напишите фортран-подпрограмму, которая вычисляет определитель квадратной матрицы A порядка n из разложения, вычисленного подпрограммой DECOMP. Используйте логарифмы, как предложено в § 3.3, или какой-нибудь другой прием, чтобы избежать переполнений и машинных нулей.

3.6. Пусть

$$A=\begin{pmatrix}0.1 & 0.2 & 0.3\\ 0.4 & 0.5 & 0.6\\ 0.7 & 0.8 & 0.9\end{pmatrix},\quad b=\begin{pmatrix}0.1\\ 0.3\\ 0.5\end{pmatrix}.$$

а) Покажите, что система линейных уравнений $Ax=b$ имеет много решений. Опишите множество возможных решений.

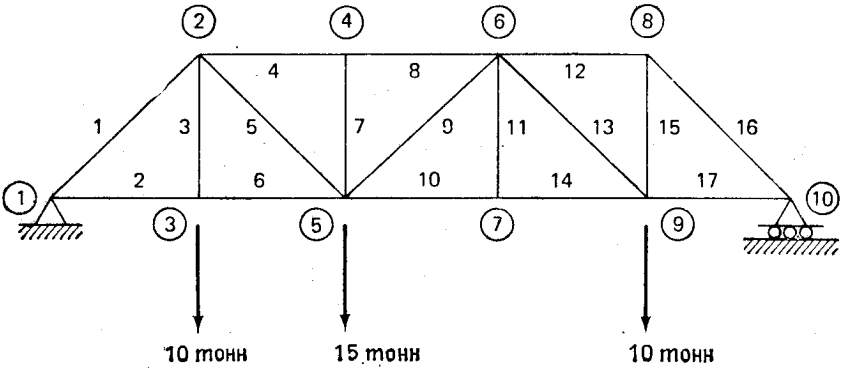
б) Предположим, что система $Ax=b$ решалась посредством DECOMP и SOLVE на гипотетической машине, которая точно выполняет арифметику. Поскольку решений много, нет оснований ожидать, что будет вычислено какое-то одно, частное решение. Что произойдет?

в) С помощью DECOMP и SOLVE вычислите решение на машине, для которой β есть степень двойки. Так как на подобной машине некоторые элементы A не являются точными числами с плавающей точкой, то матрица, задаваемая подпрограмме DECOMP, не будет точно вырожденной. Какое решение получено? Почему? В каком смысле оно будет «хорошим» решением? В каком смысле оно будет «плохим» решением?

3.7. При интерполировании данных кубическими сплайнами возникает, как мы увидим в следующей главе, такая трехдиагональная матрица:

$$A=\begin{pmatrix}-1 & 1 & & & & & & \\ 1 & 4 & 1 & & & & 0 & \\ & 1 & 4 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & & 1 & 4 & 1 & \\ & 0 & & & & 1 & 4 & 1\\ & & & & & & 1 & -1\end{pmatrix}$$

Как меняется оценка обусловленности этой матрицы по мере роста ее порядка? Какие специальные свойства имеет факторизованный массив, получаемый на выходе DECOMP? Как упростить алгоритм гауссова исключения для этого специального случая? Как бы вы стали решать линейную систему очень высокого порядка с такой матрицей?



3.8. Требуется рассчитать усилия в плоской ферме, состоящей из 17 стержней, которая изображена на диаграмме. Предполагается, что стержни фермы соединены в узлах шарнирами без трения. Теорема из элементарной механики утверждает, что поскольку число узлов узлов j и число стержней m связаны соотношением $2j-3=m$, то ферма статически определима. Это значит, что усилия полностью определяются условиями статического равновесия в узлах. Пусть F_x обозначает горизонтальные, а F_y — вертикальные компоненты сил. Если положить $\alpha=\sin 45°=\cos 45°$ и допустить малые перемещения, то условиями равновесия будут:

$$\text{узел } 2 \begin{cases} \sum F_x = -\alpha f_1 + f_4 + \alpha f_5 = 0, \\ \sum F_y = -\alpha f_1 - \alpha f_3 - \alpha f_5 = 0; \end{cases}$$

$$\text{узел } 3 \begin{cases} \sum F_x = -f_2 + f_6 = 0, \\ \sum F_y = f_3 - 10 = 0; \end{cases}$$

$$\text{узел } 4 \begin{cases} \sum F_x = -f_4 + f_8 = 0, \\ \sum F_y = -f_7 = 0; \end{cases}$$

$$\text{узел } 5 \begin{cases} \sum F_x = -\alpha f_5 - f_6 + \alpha f_9 + f_{10} = 0, \\ \sum F_y = \alpha f_5 + f_7 + \alpha f_9 - 15 = 0; \end{cases}$$

$$\text{узел } 6 \begin{cases} \sum F_x = -f_8 - \alpha f_9 + f_{12} + \alpha f_{13} = 0, \\ \sum F_y = -\alpha f_9 - f_{11} - \alpha f_{13} = 0; \end{cases}$$

$$\text{узел } 7 \begin{cases} \sum F_x = -f_{10} + f_{14} = 0, \\ \sum F_y = f_{11} = 0; \end{cases}$$

$$\text{узел } 8 \begin{cases} \sum F_x = -f_{12} + \alpha f_{16} = 0, \\ \sum F_y = -f_{15} - \alpha f_{16} = 0; \end{cases}$$

$$\text{узел } 9 \begin{cases} \sum F_x = -\alpha f_{13} - f_{14} + f_{17} = 0, \\ \sum F_y = \alpha f_{13} + f_{15} - 10 = 0; \end{cases}$$

$$\text{узел } 10 \begin{cases} \sum F_x = -\alpha f_{16} - f_{17} = 0. \end{cases}$$

Напишите фортран-программу, которая использует подпрограммы DECOMP и SOLVE для нахождения усилий из этой линейной системы уравнений. Хорошо ли обусловлена матрица линейной системы?

4. ИНТЕРПОЛЯЦИЯ

Предположим, что задано множество вещественных абсцисс $x_1, \ldots, x_n$ и соответствующие ординаты $y_1, \ldots, y_n$. Здесь $x_1 < x_2 < \ldots < x_n$, а каждое y_i — некоторое вещественное число, отвечающее x_i, определенное математически или являющееся результатом какого-либо наблюдения. Задача одномерной интерполяции состоит в построении функции f, такой, что $f(x_i) = y_i$ для всех i, и при этом $f(x)$ должна принимать «разумные» значения для x, лежащих между заданными точками. Критерий разумности меняется от задачи к задаче и ему, возможно, никогда не будет дано точного определения.

Если ординаты $\{y_i\}$ происходят от гладкой математической функции и ошибки в них не превосходят уровня округлений, то можно рассчитывать, что задача имеет удовлетворительное решение. Читатель, быть может, знаком, например, с линейной интерполяцией в таблице логарифмов.

Если точки (x_i, y_i) получены из очень точных экспериментальных наблюдений, то зачастую их можно считать лишенными ошибок, и тогда вполне разумно интерполировать их гладкой функцией. Если, с другой стороны, эти точки проистекают из сравнительно грубых экспериментов, то неправомерно требовать от интерполирующей функции точно удовлетворять таким данным. Позволяя значениям $f(x_i)$ отличаться от y_i, можно очень хорошо отразить характер изменения данных и даже поправить некоторые из содержащихся в них ошибок. Приближение данных, не являющееся точной интерполяцией, обсуждается в гл. 9. Многие методы там совершенно иные.

Цели интерполяции разнообразны, но почти всегда в ее основе — желание иметь быстрый алгоритм вычисления значений $f(x)$ для x, не содержащихся в таблице данных (x_i, y_i). Компактная таблица данных и небольшая подпрограмма интерполирования могут заменить очень длинную таблицу значений функции. Подчас нужно находить значения $f'(x)$ или $f''(x)$ в промежуточных точках или вычислять интеграл от функции f по произвольному подинтервалу (a, b) интервала (x_1, x_n).

Очень важно для задачи интерполирования определение того, как должна вести себя приемлемая функция между заданными точками. В конце концов эти точки могут быть интерполированы бесконечным множеством различных функций, и нужно иметь некоторый критерий выбора. Обычно критерии формулируются в терминах гладкости и простоты; например, функция f должна быть аналитична и максимальное значение $|f''(x)|$ по всему интервалу должно быть насколько возможно мало, или f должна быть полиномом наименьшей степени, и т. п.

Многие интерполирующие функции генерируются линейными комбинациями элементарных функций. Линейные комбинации одночленов $\{x^k\}$ приводят к полиномам. Линейные комбинации тригонометрических функций $\{\cos kx, \sin kx\}$ ведут к тригонометрическим полиномам. Используются также, хотя и реже, линейные комбинации экспонент $\{\exp(\beta_k x)\}$ или рациональные функции вида

$$\frac{\alpha_0+\alpha_1 x+\ldots+\alpha_m x^m}{\beta_0+\beta_1 x+\ldots+\beta_n x^n}.$$

В этой главе мы рассмотрим сначала полиномиальную интерполяцию, а затем один вид кусочно-полиномиальной интерполяции, так называемую *сплайн-интерполяцию.*

4.1. Полиномиальная интерполяция

Исторически и прагматически наиболее важным классом интерполирующих функций является множество алгебраических полиномов. Полиномы имеют очевидное достоинство — их значения легко вычислять (§ 4.2). Их также легко складывать, умножать, интегрировать или дифференцировать. Еще одно свойство полиномов: если c — константа, а $p(x)$ — полином, то полиномами будут и $p(cx)$, и $p(x+c)$.

Разумеется, класс функций может иметь все вышеуказанные свойства и тем не менее не иметь удовлетворительных аппроксимационных качеств. К счастью, у нас есть веские основания полагать, что *любая* непрерывная функция $f(x)$ на замкнутом интервале может быть хорошо приближена некоторым полиномом $p_n(x)$. Это следует из раннего результата теории приближений, так называемой *аппроксимационной теоремы Вейерштрасса: Если f — непрерывная на конечном замкнутом интервале $[a, b]$ функция, то для любого $\varepsilon>0$ существует полином $p_n(x)$ степени $n=n(\varepsilon)$, такой, что*

$$\max_{x\in[a,\,b]} |f(x)-p_n(x)| < \varepsilon.$$

Подробное доказательство можно найти в книгах Рэлстон (1965) или Вендрофф (1966).

Хотя некоторые доказательства теоремы Вейерштрасса имеют конструктивный характер, получаемый полином p_n обычно столь высокой степени, что пользоваться им непрактично. Далее, теорема Вейерштрасса ничего не говорит о существовании удовлетворительного *интерполяционного* полинома для заданного множества точек $\{(x_i, y_i)\}$. И хотя утешительно знать, что некоторый полином аппроксимирует $f(x)$ с заданной точностью на всем интервале $[a, b]$, нет гарантии, что такой полином можно найти посредством практичного алгоритма.

Полином степени n можно записать по степеням x:

$$y(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

(здесь $n+1$ коэффициентов). Чтобы однозначно определить коэффициенты, нужно задать $n+1$ корректно поставленных условий. При интерполяции обычно требуют, чтобы полином проходил через $n+1$ точек (x_i, y_i), $i=0, 1, \ldots, n$, где все x_i различны. Это дает $n+1$ линейных уравнений для неизвестных коэффициентов a_j:

$$y_i = \sum_{j=0}^{n} a_j x_i^j \quad (i = 0, 1, \ldots, n).$$

Решение, если оно существует, определяется единственным образом, как показывает следующее рассуждение. Зафиксируем множество $\{x_i\}$, состоящее из различных точек. Если бы имелось два полинома y и z, принимающих одни и те же значения y_i в точках x_i ($i=0, 1, \ldots, n$), то разность $y-z$ обращалась в нуль для $n+1$ различных значений переменной x. Поскольку степень $y-z$ не превышает n, то $y-z$ обязан быть нулевым полиномом, следовательно, y и z должны совпадать. Это доказывает *единственность* полинома степени $\leqslant n$, интерполирующего произвольное множество из $n+1$ точек, но не его существование.

Вспомним теперь, что $n+1$ уравнений, определяющих коэффициенты полинома y через заданные $\{y_i\}$, являются линейными. Один из главных результатов линейной алгебры состоит в том, что линейная система либо имеет единственное решение для любого набора $\{y_i\}$, либо существуют наборы $\{y_i\}$, для которых решений много. Поскольку мы только что доказали, что более одного решения не может быть, значит, должен существовать единственный полином для любой интерполяционной задачи с $n+1$ различными абсциссами.

Итак, мы можем провести единственный полином степени $\leqslant n$ через любые $n+1$ точек с различными абсциссами, и это один из возможных подходов к поставленной в начале этой главы

задаче интерполирования. Поскольку все множество данных включается в одну полиномиальную функцию, то она может быть названа *глобальным* интерполянтом [1]) этих данных.

После того как мы решили для интерполирования воспользоваться многочленом степени $\leqslant n$, в наших руках еще остается возможность выбора базиса в пространстве таких многочленов. В приведенном выше доказательстве был взят базис из одночленов $1, x, x^2, \ldots, x^n$. Это приводит, как мы видели, к системе линейных уравнений, которая в принципе может быть решена методами гл. 3. Однако во многих случаях уравнения чрезвычайно плохо обусловлены. Предположим, например, что абсциссы $\{x_i\}$ распределены приблизительно равномерно на интервале $[0, 1]$. Оказывается, что последовательные степени $1, x, x^2, \ldots$ почти линейно зависимы на интервале $[0, 1]$, отчасти потому, что все они положительны и графики всех идут от $(0, 0)$ к $(1, 1)$. Именно эта близость к линейной зависимости делает решение линейной системы при нормальной рабочей точности крайне трудным делом для порядка n, превышающего 10.

Гораздо более удовлетворительный способ вычисления полинома, который интерполирует точки (x_i, y_i), состоит в использовании базиса так называемых *лагранжевых полиномов*, ассоциированных с множеством $\{x_i\}$. Это полиномы $\{l_j(x)\}$ $(j=0, 1, \ldots, n)$ степени n, такие, что

$$l_j(x_i) = \begin{cases} 1, & \text{если } i = j, \\ 0 & \text{в противном случае.} \end{cases}$$

Легко видеть, что полином степени n

$$l_j(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}$$

удовлетворяет этим условиям. По предыдущему l_j определяется единственным образом. Каждый множитель числителя обращает $l_j(x_i)$ в нуль при некотором $i \neq j$. Соответствующие множители знаменателя нормируют полином так, что $l_j(x_j)=1$. Полином $l_j(x)y_j$ принимает значение y_j в точке x_j и равен нулю во всех точках x_i $(i \neq j)$. Таким образом, интерполяционный полином степени n, который проходит через $n+1$ точек (x_i, y_i), выражается формулой

$$y(x) = \sum_{j=0}^{n} l_j(x)\, y_j.$$

Ясно, что число арифметических операций (и, следовательно, время выполнения) для этого метода пропорционально n^2.

---

[1]) В оригинале — global fit.— *Прим. перев.*

Важно подчеркнуть еще раз, что существует *один и только один* полином степени $\leqslant n$, который интерполирует заданные $n+1$ точек. В литературе имеется множество различных формул для интерполяционных полиномов, основанных на различном выборе базисов; однако при данном наборе точек все они порождают один и тот же полином. Таким образом, полином, получаемый посредством этого подхода, совпадает с полиномом, найденным путем решения линейных уравнений, при условии, что вычисления проводятся в точной арифметике.

Ошибки округлений, соображения, связанные с памятью и временем, могут повлиять на выбор метода. Главное соображение при выборе — это конкретное применение интерполяционного полинома. Коэффициенты такого полинома нужны редко. Обычно лагранжева интерполяция или какой-либо аналогичный метод является лучшим выбором.

Часто глобальную полиномиальную интерполяцию лучше не применять вообще. Мы приведем сейчас один довод в пользу этой точки зрения. *Интерполяционным массивом* на отрезке $[a, b]$ назовем любой треугольный массив

$$
\begin{array}{ccccc}
x_0^1 & & & & \\
x_0^2 & x_1^2 & & & \\
x_0^3 & x_1^3 & x_2^3 & & \\
\cdot & \cdot & \cdot & & \\
\cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot
\end{array}
$$

с тем свойством, что все $x_j^i \in [a, b]$ и элементы каждой отдельной строки различны. Пусть $P_n(f)(x)$ — интерполяционный полином, совпадающий с функцией $f(x)$ в точках $n+1$-й строки.

Теорема Фабера

Для любого интерполяционного массива найдутся непрерывная функция g и точка x из $[a, b]$, такие, что $P_n(g)(x)$ не сходятся к $g(x)$ при $n \to \infty$.

В § 4.3 мы приведем пример, иллюстрирующий теорему Фабера. Упражнения 4.5 и 4.7 дают другие убедительные образцы особенностей в поведении полиномиальной интерполяции.

Один из возможных способов оценки ошибки при интерполяции заключается в следующем. Предполагают, что заданные числа y_i являются в действительности значениями некоторой функции f для данных абсцисс x_i. Положим, что f имеет для всех x $n+1$ непрерывных производных. Пусть p_n — единственный полином степени $\leqslant n$, интерполирующий заданные точки $\{(x_i,$

$y_i)\}$. Тогда можно показать, что для любого действительного x

$$f(x)-p_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x-x_i),$$

где ξ — некоторая неизвестная точка на интервале, определяемом точками $x_0, \ldots, x_n$ и x. Когда известны границы для $f^{(n+1)}(x)$, этот результат дает оценку ошибки.

Существует много обобщений лагранжевой интерполяции; наиболее употребительна среди них *эрмитова интерполяция*. Здесь фиксируются n абсцисс $\{x_i\}$, n заданных значений $\{y_i\}$ и n заданных угловых коэффициентов $\{y_i'\}$. Задача состоит в том, чтобы найти полином $P(x)$ максимальной степени $2n-1$, такой, что для $i=1, 2, \ldots, n$

$$P(x_i)=y_i$$

и

$$P'(x_i)=y_i'.$$

Опять-таки, если все x_i различны, то существует единственное решение, и оно может быть построено способом, вполне аналогичным методу Лагранжа. Мы отсылаем читателя к книгам Вендрофф (1966) или Рэлстон (1965).

Помимо вопросов глобальной сходимости, полиномиальная интерполяция имеет и другие недостатки. Время построения и вычисления интерполяционных полиномов высокой степени может для некоторых приложений оказаться чрезмерным. Полиномы высокой степени могут приводить также к трудным проблемам, связанным с ошибками округлений.

4.2. Вычисление полиномов

Некоторые программы многократно вычисляют определенные полиномы для различных значений их аргументов. Поэтому важно уметь быстро вычислять полиномы.

Вычисление полинома

$$P(x)=a_1x^n+a_2x^{n-1}+\ldots+a_{n+1}$$

посредством фортран-текста

```
      P = A(N+1)
      DO 10 I = 1, N
        P = P + A(I)*X**(N-I+1)
   10 CONTINUE
```

требует $n(n+1)/2$ умножений и n сложений. Простой метод, называемый *схемой Горнера*, состоит в перезаписи $P(x)$ в виде

$$P(x)=a_{n+1}+x(a_n+x(a_{n-1}+x(\ldots(a_2+a_1x)\ldots))).$$

Это легко программируется. Например,

```
      P = A(1)
      DO 10 I=1, N
        P = P*X + A(I+1)
10    CONTINUE
```

Схема Горнера требует только n умножений и n плавающих сложений. Тот же алгоритм в книгах по алгебре называется также *синтетическим делением* или *синтетической подстановкой*.

Имя Горнера присвоено методу потому, что он изложил это правило в знаменитой статье (на другую тему), опубликованной в 1819 г. В действительности более чем за сто лет до этого идея переупорядочения была опубликована Исааком Ньютоном.

Известно, что схема Горнера является оптимальной среди методов, переупорядочивающих полином для быстрого вычисления и при этом не делающих значительных вычислений в процессе переупорядочения. Таким образом, если заданы коэффициенты полинома и аргумент x, то, вообще говоря, нельзя вычислить полином за меньшее число сложений и умножений, чем для схемы Горнера.

Интересное обсуждение вопроса о вычислении полиномов и некоторые обобщения схемы Горнера можно найти в книге Кнут (1969).

4.3. Пример: функция Рунге

Опасности, связанные с полиномиальной интерполяцией, впервые обнаружил Рунге в 1901 г. Он пытался интерполировать полиномами на интервале [—1, 1] простую функцию

$$y(x) = \frac{1}{1+25x^2}$$

при равномерном распределении абсцисс. Выяснилось, что при бесконечном увеличении порядка n интерполяционного полинома p_n последовательность $p_n(x)$ расходится в интервалах $0.726\ldots \leqslant |x| < 1$. Это явление графически показано на рис. 4.1. Отметим, что глобальная полиномиальная интерполяция дает в примере Рунге очень хорошее согласие в центральной части интервала. Это привело к идее об интерполировании посредством *движущегося полинома*. Например, можно провести полином 10-й степени через 11 последовательных точек, но использовать его значения только в центральной части этого интервала. Поскольку сплайн-интерполяция дает много лучшие результаты, мы не станем обсуждать подробней идею движущихся полиномов.

Если заданные абсциссы распределены не равномерно, а так, что вблизи концов интервала они помещены в корни чебышевского полинома степени $n+1$, то проблема расходимости для функции Рунге исчезает. Получающиеся интерполяционные многочлены $p_n(x)$ сходятся при $n\to\infty$ к $y(x)$ для всех x из $[-1, 1]$.

Разумеется, этот трюк с помещением задаваемых точек в корни чебышевского полинома работает не для всех непрерывных

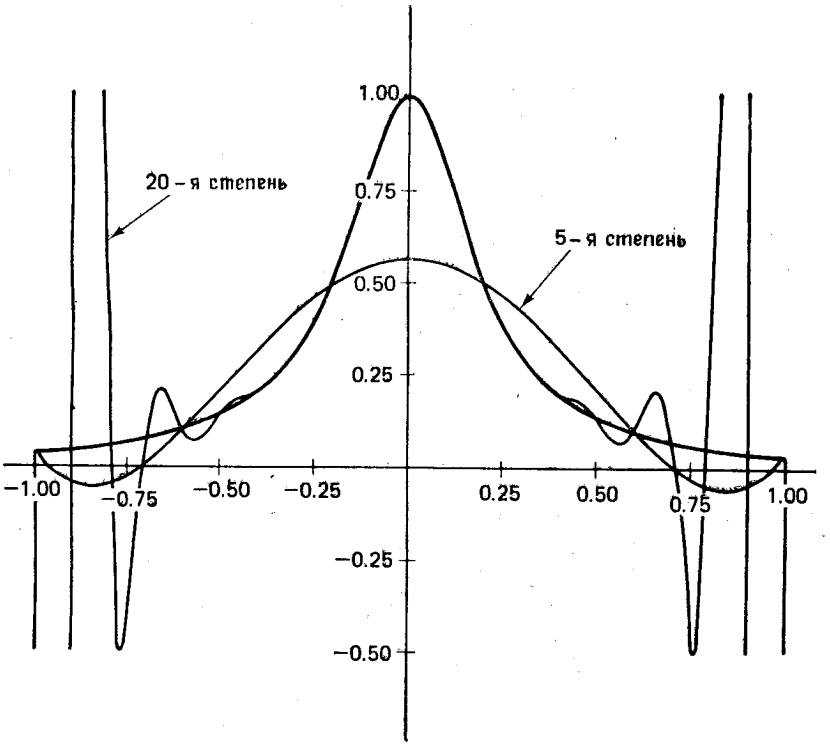


Рис. 4.1. Функция Рунге, интерполированная многочленами 5-й и 20-й степени при равноудаленных абсциссах.

функций. Теорема Фабера отрицает существование какой-либо схемы, подходящей для всех случаев.

Однако можно показать, что если функция f имеет непрерывную производную в $[-1, 1]$, то интерполяционные многочлены p_n, совпадающие с f в корнях полиномов Чебышева степени $n+1$, сходятся при $n\to\infty$ к f для любой точки x из $[-1, 1]$. Вопрос о чебышевских полиномах хорошо изложен в книгах Ланцош (1956), стр. 178, и Вендрофф (1966), стр. 53.

4.4. Сплайн-интерполяция

Кубические сплайн-функции — это недавнее математическое изобретение, но они моделируют очень старое механическое устройство. Чертежники издавна пользовались механическими сплайнами, представляющими собой гибкие рейки из какого-нибудь упругого материала, обычно дерева или (в последнее время) пластика. Механический сплайн закрепляют, подвешивая грузила в точках интерполяции, называемых исторически *узлами*. Сплайн принимает форму, минимизирующую его потенциальную энергию, и в теории балок устанавливается, что эта энергия пропорциональна интегралу по длине дуги от квадрата кривизны сплайна.

Если сплайн представить функцией $s(x)$, то при малых наклонах вторая производная $s''(x)$ приблизительно равна кривизне, а дифференциал длины дуги можно приближенно заменить на dx. Таким образом, энергия подобного *линеаризованного сплайна* пропорциональна интегралу $\int (s''(x))^2 dx$. Если заданы узлы (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), то линеаризованный сплайн $s(x)$ есть функция, для которой $s(x_i)=y_i$ $(i=1, 2, \ldots, n)$, и при этом интеграл $\int_{x_1}^{x_n} (s''(x))^2 dx$ имеет минимальное значение.

Поскольку механический сплайн не разрушается, то следует считать, что s и s' непрерывны на $[x_1, x_n]$. Далее, элементарная теория балок показывает, что $s(x)$ является кубическим полиномом между каждой соседней парой узлов и что соседние полиномы соединяются непрерывно, так же как и их первые и вторые производные.

Кубическая сплайн-функция, удовлетворяющая условиям $s''(x_1)=s''(x_n)=0$, называется *естественным кубическим сплайном*. С математической точки зрения (Алберг и др. (1967)) доказано, что она является единственной функцией, обладающей свойством минимальной кривизны, среди всех функций, интерполирующих данные точки и имеющих квадратично интегрируемую вторую производную. В этом смысле естественный кубический сплайн есть самая гладкая из функций, интерполирующих заданные точки.

Стоит подсчитать число параметров в кубической сплайн-функции. Соответственно n—1 интервалам между узлами имеется n—1 фрагментов кубических кривых, у каждой из них четыре параметра; всего, таким образом, нужно определить $4n$—4 параметров. Тот факт, что функция s и ее первая и вторая производные непрерывны во всех n—2 внутренних узлах x_i, равносилен $3(n$—2) условиям на s. Далее, требование, чтобы $s(x_i)=y_i$ для

каждого из n узлов, накладывает еще n условий на s; в общей сложности пока получается $4n$—6 условий. Следовательно, нужно еще два условия, чтобы однозначно определить сплайн. Полагая $s''(x_1)=s''(x_n)=0$, мы и придем к упомянутому выше *естественному сплайну*.

Иногда при интерполировании посредством кубических сплайнов вместо естественных граничных условий накладывают какие-либо другие два требования на концах или вблизи концов сплайна: например, предписывают значения наклонов $s'(x_1)$ и $s'(x_n)$. Такие кубические сплайны минимизируют линеаризованный интеграл энергии при соблюдении наложенных двух условий.

Построение кубического сплайна — простой и численно устойчивый процесс. Рассмотрим подынтервал $[x_i, x_{i+1}]$, и пусть

$$h_i = x_{i+1} - x_i,$$
$$w = \frac{x - x_i}{h_i},$$
$$\bar{w} = 1 - w.$$

Когда x пробегает этот подынтервал, w изменяется от 0 до 1, а $\bar{w}$ — от 1 до 0. Немножко интуиции, и мы придем к представлению сплайна на этом подынтервале формулой

$$s(x) = wy_{i+1} + \bar{w}y_i + h_i^2[(w^3 - w)\sigma_{i+1} + (\bar{w}^3 - \bar{w})\sigma_i],$$

где σ_i и σ_{i+1} — некоторые константы, которые еще предстоит определить. Первые два члена этого выражения соответствуют стандартной линейной интерполяции, а взятый в скобки член есть кубическая поправка, которая обеспечит нам дополнительную гладкость. Заметим, что поправочный член обращается в нуль на концах подынтервала, так что

$$s(x_i) = y_i$$

и

$$s(x_{i+1}) = y_{i+1}.$$

Таким образом, $s(x)$ интерполирует заданные значения независимо от выбора чисел σ_i.

Продифференцируем теперь $s(x)$ трижды как сложную функцию от x, учитывая, что $w' = 1/h_i$ и $\bar{w}' = -1/h_i$:

$$s'(x) = \frac{y_{i+1} - y_i}{h_i} + h_i[(3w^2 - 1)\sigma_{i+1} - (3\bar{w}^2 - 1)\sigma_i],$$
$$s''(x) = 6w\sigma_{i+1} + 6\bar{w}\sigma_i,$$
$$s'''(x) = \frac{6(\sigma_{i+1} - \sigma_i)}{h_i}.$$

Заметим, что $s''(x)$ — линейная функция, интерполирующая значения $6\sigma_i$ и $6\sigma_{i+1}$. Следовательно,

$$\sigma_i = \frac{s''(x_i)}{6}.$$

Это проясняет смысл коэффициента σ_i, но не позволяет определить его величину. Отметим еще, что $s'''(x)$ является константой на каждом подынтервале и что четвертая производная функции $s(x)$ тождественно равна нулю. Так, конечно, и должно быть, поскольку локально $s(x)$ совпадает с кубической кривой.

Вычисление $s'(x)$ в концевых точках подынтервала дает

$$s'_+(x_i) = \Delta_i - h_i(\sigma_{i+1} + 2\sigma_i),$$
$$s'_-(x_{i+1}) = \Delta_i + h_i(2\sigma_{i+1} + \sigma_i),$$

где $\Delta_i = (y_{i+1} - y_i)/h_i$. Приходится временно пользоваться обозначениями s'_+ и s'_-, так как наша формула для $s(x)$ имеет силу только на $[x_i, x_{i+1}]$, и потому производные в концевых точках являются односторонними. Чтобы получить желаемую непрерывность $s'(x)$, наложим во внутренних узлах условия

$$s'_-(x_i) = s'_+(x_i), \quad i = 2, \ldots, n-1.$$

[Непрерывность $s''(x)$ следует непосредственно из принятого представления для $s(x)$.] Хотя значение $s'_-(x_i)$ выводится из рассмотрения подынтервала $[x_{i-1}, x_i]$, формула для него получается простой заменой i на $i-1$ в выражении для $s'_-(x_{i+1})$. Это приводит к равенству

$$\Delta_{i-1} + h_{i-1}(2\sigma_i + \sigma_{i-1}) = \Delta_i - h_i(\sigma_{i+1} + 2\sigma_i)$$

и, следовательно,

$$h_{i-1}\sigma_{i-1} + 2(h_{i-1} + h_i)\sigma_i + h_i\sigma_{i+1} = \Delta_i - \Delta_{i-1}, \quad i = 2, \ldots, n-1.$$

Это система из $n-2$ линейных уравнений относительно неизвестных коэффициентов σ_i, $i = 1, \ldots, n$. Нужно указать еще два условия, чтобы однозначно определить интерполяционный сплайн. Среди нескольких различных способов выбора этих двух условий мы предпочли следующий.

Пусть $c_1(x)$ и $c_n(x)$ — единственные кубические кривые, которые проходят соответственно через четыре первые и четыре последние из заданных точек. Два граничных условия связывают третьи производные функции $s(x)$ с третьими производными этих кубических кривых, именно

$$s'''(x_1) = c_1'''$$

и

$$s'''(x_n) = c_n'''.$$

Константы c_1''' и c_n''' можно определить прямо из данных задачи, минуя действительное нахождение $c_1(x)$ и $c_n(x)$. Мы уже ввели величины

$$\Delta_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i},$$

являющиеся приближениями к значениям первой производной. Пусть

$$\Delta_i^{(2)} = \frac{\Delta_{i+1} - \Delta_i}{x_{i+2} - x_i}$$

и

$$\Delta_i^{(3)} = \frac{\Delta_{i+1}^{(2)} - \Delta_i^{(2)}}{x_{i+3} - x_i}.$$

Эти величины называются разделенными разностями; при этом $2\Delta_i^{(2)}$ и $6\Delta_i^{(3)}$ аппроксимируют соответственно вторую и третью производные. В частности,

$$c_1''' = 6\Delta_1^{(3)}$$

и

$$c_n''' = 6\Delta_{n-3}^{(3)}.$$

Итак, мы требуем, чтобы

$$\frac{\sigma_2 - \sigma_1}{h_1} = \Delta_1^{(3)}$$

и

$$\frac{\sigma_n - \sigma_{n-1}}{h_{n-1}} = \Delta_{n-3}^{(3)}.$$

Чтобы симметризовать полученную систему уравнений, эти последние два уравнения нужно умножить соответственно на h_1^2 и $-h_{n-1}^2$, что дает

$$-h_1\sigma_1 + h_1\sigma_2 = h_1^2\Delta_1^{(3)}$$

и

$$h_{n-1}\sigma_{n-1} - h_{n-1}\sigma_n = -h_{n-1}^2\Delta_{n-3}^{(3)}.$$

Для сплайна с этими граничными условиями коэффициенты σ_i, таким образом, удовлетворяют следующей системе из n линейных уравнений с n неизвестными:

$$\begin{pmatrix} -h_1 & h_1 & 0 & 0 & & \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \\ 0 & h_2 & 2(h_2+h_3) & h_3 & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ 0 & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & 0 & h_{n-1} & -h_{n-1} & \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \cdot \\ \cdot \\ \cdot \\ \sigma_{n-1} \\ \sigma_n \end{pmatrix} = \begin{pmatrix} h_1^2\Delta_1^{(3)} \\ \Delta_2 - \Delta_1 \\ \Delta_3 - \Delta_2 \\ \cdot \\ \cdot \\ \cdot \\ \Delta_{n-1} - \Delta_{n-2} \\ -h_{n-1}^2\Delta_{n-3}^{(3)} \end{pmatrix}$$

Эту систему из n уравнений можно решить методом исключения. Для вычисления неизвестных σ_i можно было бы использовать подпрограммы DECOMP и SOLVE. Однако матрица коэффициентов имеет несколько специальных свойств:

1. Матрица трехдиагональная.
2. Матрица симметричная.
3. При любом выборе $x_1<x_2<\ldots<x_n$ матрица невырожденная и диагонально доминирующая.

Следовательно, всегда существует единственное решение $\sigma_1, \ldots, \sigma_n$. Можно показать также, что для любого разумного выбора точек $x_1, x_2, \ldots, x_n$ матрица коэффициентов хорошо обусловлена. Основываясь на этом и на свойстве диагонального доминирования, можно ожидать, что применение гауссова исключения без масштабирования и без выбора ведущих элементов позволит тем не менее получить решение с хорошей точностью.

Гауссово исключение приводит исходную систему к верхней треугольной форме:

$$\begin{pmatrix} \alpha_1 & h_1 & & & & 0 \\ & \alpha_2 & h_2 & & & \\ & & \alpha_3 & h_3 & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & \cdot \\ 0 & & & & \cdot & \cdot \\ & & & & & \alpha_n \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \cdot \\ \cdot \\ \cdot \\ \sigma_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \cdot \\ \cdot \\ \cdot \\ \beta_n \end{pmatrix},$$

здесь диагональные элементы α_i вычисляются по формулам

$$\alpha_1 = -h_1,$$

$$\alpha_i = 2(h_{i-1}+h_i) - \frac{h_{i-1}^2}{\alpha_{i-1}}, \quad i = 2, 3, \ldots, n-1,$$

$$\alpha_n = -h_{n-1} - \frac{h_{n-1}^2}{\alpha_{n-1}},$$

а правые части β_i — по формулам

$$\beta_1 = h_1^2 \Delta_1^{(3)},$$

$$\beta_i = (\Delta_i - \Delta_{i-1}) - \frac{h_{i-1}\beta_{i-1}}{\alpha_{i-1}}, \quad i = 2, 3, \ldots, n-1,$$

$$\beta_n = -h_{n-1}^2 \Delta_{n-3}^{(3)} - h_{n-1}\frac{\beta_{n-1}}{\alpha_{n-1}}.$$

Наконец, коэффициенты σ_i определяются посредством обратной подстановки:

$$\sigma_n = \frac{\beta_n}{\alpha_n},$$

$$\sigma_i = \frac{\beta_i - h_i\sigma_{i+1}}{\alpha_i}, \quad i = n-1, n-2, \ldots, 1.$$

В некоторых приложениях желательно хранить σ_i и использовать их непосредственно для вычисления сплайна. Однако если сплайн нужно вычислять многократно, то стоит переупорядочить члены. По разным причинам предпочтительней вычислить и сохранять коэффициенты b_i, c_i и d_i в кубическом представлении

$$s(x)=y_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3, \qquad x_i\leqslant x\leqslant x_{i+1},$$

на каждом подынтервале $[x_i, x_{i+1}]$, $i=1, 2, \ldots, n-1$. Эти коэффициенты выражаются формулами

$$\sigma_i=\frac{y_{i+1}-y_i}{h_i}-h_i(\sigma_{i+1}+2\sigma_i),$$
$$c_i=3\sigma_i,$$
$$d_i=\frac{\sigma_{i+1}-\sigma_i}{h_i},$$

$i=1, 2, \ldots, n-1$. Использование такой формы хранения сплайна упрощает операции с ним, например вычисление производных и интегралов.

4.5. Подпрограммы SPLINE и SEVAL

Метод вычисления параметров кубической сплайн-функции, обсуждавшийся сейчас, реализован в подпрограмме SPLINE. Комментарии в программе описывают ее использование. В процессе исключения в массивах B, C и D хранятся величины α_i, β_i и h_i (см. предыдущий параграф).

С помощью подпрограммы-функции SEVAL можно вычислять значения сплайна после того, как его коэффициенты были определены подпрограммой SPLINE. Если по сравнению с предыдущим вызовом независимая переменная U уже не находится в том же интервале, то для разыскания нужного интервала применяется двоичный поиск. Читателю, не слишком хорошо знакомому с двоичным поиском, мы настоятельно советуем просчитать несколько примеров на руках. Разумеется, двоичный поиск не является наилучшим для всех случаев способом (почему?).

Приводимый ниже простой пример демонстрирует обращение к подпрограммам SPLINE и SEVAL. На выходе будет получено: 2.50000 15.62500. (Почему?)

```
C     ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ ПРОГРАММ SPLINE
C     И SEVAL
C
      REAL X(10), Y(10), B(10), C(10), D(10)
      REAL S, U, SEVAL
      INTEGER I, N
C
      N=10
      DO 1 I=1, N
```

```
      X(I)=I
      Y(I)=X(I) **3
    1 CONTINUE
C
      CALL SPLINE (N, X, Y, B, C, D)
C
      U=2.5
      S=SEVAL (N, U, X, Y, B, C, D)
      WRITE (6, 2) U, S
    2 FORMAT (2F 10.5)
      STOP
      END

        SUBROUTINE SPLINE (N, X, Y, B, C, D)
        INTEGER N
        REAL X(N), Y(N), B(N), C(N), D(N)
C
C     ВЫЧИСЛЯЮТСЯ КОЭФФИЦИЕНТЫ B(I), C(I) И D(I), I=1,
C     2, ..., N, ДЛЯ КУБИЧЕСКОГО ИНТЕРПОЛЯЦИОННОГО
C     СПЛАЙНА
C
C       S(X)=Y(I)+B(I)*(X-X(I))+C(I)*(X-X(I))**2+
C       +D(I)*(X-X(I))**3
C
C       ДЛЯ X(I) .LE. X .LE. X(I+1)
C
C     ВХОДНАЯ ИНФОРМАЦИЯ..
C
C       N=ЧИСЛО ЗАДАННЫХ ТОЧЕК ИЛИ УЗЛОВ (N .GE. 2)
C       X=АБСЦИССЫ УЗЛОВ В СТРОГО ВОЗРАСТАЮЩЕМ
C       ПОРЯДКЕ
C       Y=ОРДИНАТЫ УЗЛОВ
C
C     ВЫХОДНАЯ ИНФОРМАЦИЯ...
C
C       B, C, D=МАССИВЫ ОПРЕДЕЛЕННЫХ ВЫШЕ КОЭФФИ-
C       ЦИЕНТОВ СПЛАЙНА.
C
C     ЕСЛИ ОБОЗНАЧИТЬ ЧЕРЕЗ P СИМВОЛ ДИФФЕРЕНЦИРО-
C     ВАНИЯ, ТО
C
C       Y(I)=S(X(I))
C       B(I)=SP(X(I))
C       C(I)=SPP(X(I))/2
C       D(I)=SPPP(X(I))/6 (ПРАВОСТОРОННЯЯ ПРОИЗВОДНАЯ)
C
C     С ПОМОЩЬЮ СОПРОВОЖДАЮЩЕЙ ПОДПРОГРАММЫ-ФУНК-
C     ЦИИ SEVAL МОЖНО ВЫЧИСЛЯТЬ ЗНАЧЕНИЯ СПЛАЙНА.
C
        INTEGER NM1, IB, I
        REAL T
C
        NM1=N-1
        IF (N .LT. 2) RETURN
        IF (N .LT. 3) GO TO 50
C
C   ПОСТРОИТЬ ТРЕХДИАГОНАЛЬНУЮ СИСТЕМУ
```

```
C
C   B=ДИАГОНАЛЬ, D=НАДДИАГОНАЛЬ, C=ПРАВЫЕ ЧАСТИ.
C
      D(1)=X(2)-X(1)
      C(2)=(Y(2)-Y(1))/D(1)
      DO 10 I=2, NM1
        D(I)=X(I+1)-X(I)
        B(I)=2. *(D(I-1)+D(I))
        C(I+1)=(Y(I+1)-Y(I))/D(I)
        C(I)=C(I+1)-C(I)
10    CONTINUE
C
C   ГРАНИЧНЫЕ УСЛОВИЯ. ТРЕТЬИ ПРОИЗВОДНЫЕ В ТОЧКАХ
C   X(1) И X(N) ВЫЧИСЛЯЮТСЯ С ПОМОЩЬЮ РАЗДЕЛЕННЫХ
C   РАЗНОСТЕЙ
C
      B(1)=-D(1)
      B(N)=-D(N-1)
      C(1)=0.
      C(N)=0.
      IF (N .EQ. 3) GO TO 15
      C(1)=C(3)/(X(4)-X(2))-C(2)/(X(3)-X(1))
      C(N)=C(N-1)/(X(N)-X(N-2))-C(N-2)/(X(N-1)-X(N-3))
      C(1)=C(1)*D(1)**2/(X(4)-X(1))
      C(N)=-C(N)*D(N-1)**2/(X(N)-X(N-3))
C
C   ПРЯМОЙ ХОД
C
15    DO 20 I=2, N
        T=D(I-1)/B(I-1)
        B(I)=B (I)-T*D(I-1)
        C(I)=C (I)-T*C(I-1)
20    CONTINUE
C
C   ОБРАТНАЯ ПОДСТАНОВКА
C
      C(N)=C(N)/B(N)
      DO 30 IB=1, NM1
        I=N-IB
        C(I)=(C(I)-D(I)*C(I+1))/B(I)
30    CONTINUE
C
C   В C(I) ТЕПЕРЬ ХРАНИТСЯ ВЕЛИЧИНА SIGMA(I), ОПРЕДЕЛЕН-
C   НАЯ В § 4.4.
C
C   ВЫЧИСЛИТЬ КОЭФФИЦИЕНТЫ ПОЛИНОМОВ
C
      B(N)=(Y(N)-Y(NM1))/D(NM1)+D(NM1)*(C(NM1)+2.*C(N))
      DO 40 I=1, NM1
        B(I)=(Y(I+1)-Y(I))/D(I)-D(I)*(C(I+1)+2.*C(I))
        D(I)=(C(I+1)-C(I))/D(I)
        C(I)=3.*C(I)
40    CONTINUE
      C(N)=3.*C(N)
      D(N)=D(N-1)
      RETURN
C
50    B(1)=(Y(2)-Y(1))/(X(2)-X(1))
```

```
      C(1)=0.
      D(1)=0.
      B(2)=B(1)
      C(2)=0.
      D(2)=0.
      RETURN
      END
      DOUBLE PRECISION FUNCTION SEVAL (N, U, X, Y, B, C, D)
      INTEGER N
      DOUBLE PRECISION U, X(N), Y(N), B(N), C(N), D(N)
C
C  ЭТА ПОДПРОГРАММА ВЫЧИСЛЯЕТ ЗНАЧЕНИЕ КУБИЧЕСКОГО
C  СПЛАЙНА
C
C     SEVAL=Y(I)+B(I)*(U-X(I))+C(I)*(U-X(I))**2+
C     D(I)*(U-X(I))**3
C
C     ГДЕ X(I) .LT. U .LT. X(I+1). ИСПОЛЬЗУЕТСЯ СХЕМА
C     ГОРНЕРА
C
C  ЕСЛИ U .LT. X(1), ТО БЕРЕТСЯ ЗНАЧЕНИЕ I=1.
C  ЕСЛИ U .GE. X(N), ТО БЕРЕТСЯ ЗНАЧЕНИЕ I=N.
C
C  ВХОДНАЯ ИНФОРМАЦИЯ..
C
C     N=ЧИСЛО ЗАДАННЫХ ТОЧЕК
C     U=АБСЦИССА, ДЛЯ КОТОРОЙ ВЫЧИСЛЯЕТСЯ ЗНАЧЕНИЕ
C     СПЛАЙНА
C     X, Y=МАССИВЫ ЗАДАННЫХ АБСЦИСС И ОРДИНАТ
C     B, C, D=МАССИВЫ КОЭФФИЦИЕНТОВ СПЛАЙНА, ВЫЧИС-
C     ЛЕННЫЕ ПОДПРОГРАММОЙ SPLINE
C
C  ЕСЛИ ПО СРАВНЕНИЮ С ПРЕДЫДУЩИМ ВЫЗОВОМ U НЕ
C  НАХОДИТСЯ В ТОМ ЖЕ ИНТЕРВАЛЕ, ТО ДЛЯ РАЗЫСКАНИЯ
C  НУЖНОГО ИНТЕРВАЛА ПРИМЕНЯЕТСЯ ДВОИЧНЫЙ ПОИСК.
C
      INTEGER I, J, K
      DOUBLE PRECISION DX
      DATA I/1/
      IF (I .GE. N) I=1
      IF(U .LT. X(I)) GO TO 10
      IF (U .LE. X(I+1)) GO TO 30
C
C  ДВОИЧНЫЙ ПОИСК
C
   10 I=1
      J=N+1
   20 K=(I+J)/2
      IF (U .LT. X(K)) J=K
      IF (U .GE. X(K)) I=K
      IF (J .GT. I+1) GO TO 20
C
C  ВЫЧИСЛИТЬ СПЛАЙН
C
   30 DX=U-X(I)
      SEVAL=Y(I)+DX*(B(I)+DX*(C(I)+DX*D(I)))
      RETURN
      END
```

УПРАЖНЕНИЯ

4.1. Постройте $n=11$ точек, полагая

$$\left.\begin{aligned} x_i &= \frac{i-1}{10}, \\ y_i &= \operatorname{erf}(x_i), \end{aligned}\right\} \quad i=1, \ldots, n.$$

Значения функции erf можно взять из имеющихся таблиц либо вычислить на своей машине по соответствующей подпрограмме. С помощью подпрограммы SPLINE найдите коэффициенты кубического сплайна, интерполирующего эти точки. С помощью подпрограммы SEVAL вычислите сплайн в промежуточных точках и сравните значения сплайна со значениями функции erf. Какова максимальная ошибка для этих промежуточных точек?

4.2. Используя подпрограммы SPLINE и SEVAL, проинтерполируйте функцию Рунге

$$f(x)=\frac{1}{1+25x^2}$$

по точкам $x_i=-1.0, -0.9, \ldots, 0.9, 1.0$; $n=21$. Сравните результаты с полиномом 20-й степени, интерполирующим те же точки.

4.3. Переделайте подпрограмму SPLINE так, чтобы она находила коэффициенты *естественного* кубического сплайна, т. е. сплайна с граничными условиями $s''(x_1)=s''(x_n)=0$. Переделка понадобится весьма существенная, поскольку, помимо прочего, теперь будет только $n-2$ неизвестных коэффициента δ_i. Нужно ли переделывать и подпрограмму SEVAL?

4.4. Сравните естественный сплайн со сплайном, построение которого описано в § 4.4. Именно, постройте тестовый пример, беря в качестве x_i, $i=1, \ldots, n$, n равноудаленных точек на интервале $[0, \pi]$ (в это число входят и концевые точки), а в качестве y_i числа $\cos x_i$. Вычислите коэффициенты сплайна по подпрограмме SPLINE в ее исходном варианте. Вычислите также коэффициенты по подпрограмме SPLINE, переделанной в соответствии с упр. 4.3. Сравните две различные функции $s(x)$ с $\cos x$ для значений x, не совпадающих с заданными точками. Особое внимание обратите на значения вблизи концов интервала. Проделайте это для нескольких значений n, например 10, 20 и 30. Почему в этом упражнении нужно брать именно $\cos x$, а не $\sin x$?

4.5. Следующие данные Бюро переписей показывают население Соединенных Штатов:

Год	Население
1900	75 994 575
1910	91 972 266
1920	105 710 620
1930	123 203 000
1940	131 669 275
1950	150 697 361
1960	179 323 175
1970	203 211 926

а) Поскольку здесь восемь точек, имеется единственный полином степени 7, интерполирующий эти точки. Однако некоторые способы представления этого полинома с вычислительной точки зрения предпочтительней других. Вот четыре

возможности; в каждом случае t изменяется на интервале $1900 \leqslant t \leqslant 1970$:

$$\sum_{j=0}^{7} a_j t^j,$$

$$\sum_{j=0}^{7} b_j (t-1900)^j,$$

$$\sum_{j=0}^{7} c_j (t-1935)^j,$$

$$\sum_{j=0}^{7} d_j \left(\frac{t-1935}{35}\right)^j.$$

Во всех случаях коэффициенты находятся из системы уравнений 8-го порядка, но матрицы этих систем весьма различны. Постройте каждую из четырех матриц и найдите оценки их обусловленности, пользуясь подпрограммой DECOMP. Затем по подпрограмме SOLVE определите коэффициенты. Проверьте, хорошо ли каждое представление воспроизводит исходные данные.

б) Проинтерполируйте данные точки полиномом 7-й степени, пользуясь представлением с наилучшей обусловленностью, найденным в задании а), а также кубическим сплайном, вычисляя его по подпрограмме SPLINE. Протабулируйте полученные функции за период с 1900 по 1980 г. с интервалом в 1 г. и нарисуйте их графики. Вы обнаружите, что обе функции очень хорошо согласуются до 1970 г., но их поведение между 1970 и 1980 г. совершенно различно. Какой из двух подходов предсказывает, что до 1980 г. наступит момент с *нулевым ростом населения*? Если вы выполняете это задание после того, как опубликованы данные переписи 1980 г., проверьте, какой способ дает более точный прогноз.

в) Переделав подпрограмму SPLINE так, чтобы она использовала естественные граничные условия, выясните, как это изменение влияет на прогноз с 1970 по 1980 г. Поскольку при естественных граничных условиях сплайн вне заданного интервала считается линейной функцией, то нулевой рост населения не может быть предсказан.

4.6. Обозначим через $P(x)$ полином степени n, такой, что $P(k)=k!k+1$ для $k=0, 1, \ldots, n$. Найдите $P(n+1)$.

Указания: рассмотрите отдельно случаи четного и нечетного n. Если вы не сможете найти аналитическое решение, используйте машину. (Эта задача, разумеется, без указаний, входила в число задач Четвертой математической олимпиады для учеников средних школ США.)

4.7. В ходе воображаемого химического эксперимента получены следующие семь пар данных (их ошибками можно пренебречь):

t	−1.000	−0.960	−0.860	−0.790	0.220	0.500	0.930
y	−1.000	−0.151	0.894	0.986	0.895	0.500	−0.306

Нужно оценить значения $y(t)$ для t, находящихся между −1.000 и 1.000, посредством интерполирования по данным точкам. Предполагается, что $y(t)$ — очень гладкая кривая.

а) Нанесите точки на график и проведите гладкую интерполирующую кривую по интуиции.

б) Найдите (единственный) полином 6-й степени, интерполирующий данные точки, и начертите его график. Хорошо ли отражает полином характер изменения данных?

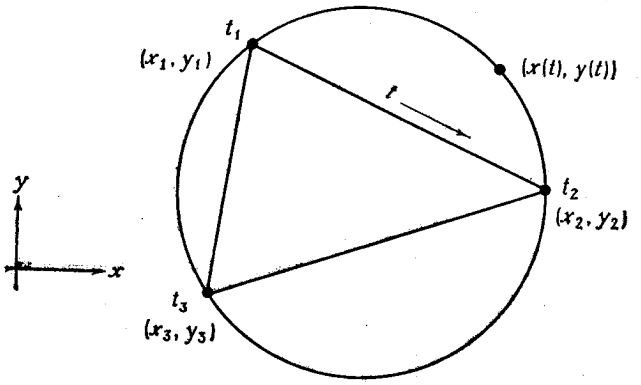
в) По подпрограмме SPLINE найдите коэффициенты кубического сплайна, интерполирующего данные точки. По подпрограмме SEVAL найдите значения сплайна в достаточном числе промежуточных точек, чтобы начертить его график. Хорошо ли отражает сплайн характер изменения данных? (Мы признательны Ричарду Ф. Томпсону, предложившему эту задачу).

4.8. Сколько плавающих умножений, делений и сложений или вычитаний выполняется подпрограммой SPLINE? Каково общее число плавающих операций? Выразите ваши результаты функциями от N.

4.9. Для случая равноудаленных узлов (т. е. $h_i=h$, $i=1, 2, \ldots, n-1$) алгоритм, реализованный в SPLINE, можно упростить. Каков алгоритм для равноудаленных узлов? Сколько плавающих операций в нем требуется? Загляните в упр. 3.7. гл. 3.

4.10. Покажите, что коэффициенты b_i, c_i и d_i кубического сплайна действительно выражаются последними тремя формулами § 4.4.

4.11. Предположим, что на евклидовой плоскости даны n точек (x_i, y_i) $(i=1, 2, \ldots, n)$ и нужно провести гладкую замкнутую кривую через эти точки в заданном порядке. Один из методов состоит в следующем: строится замкнутый



многоугольник, соединяющий точки в указанном порядке; пусть t обозначает длину дуги вдоль многоугольника $(0 \leqslant t \leqslant T)$, так что вершинам многоугольника соответствуют значения

$$0 = t_1 < t_2 < \ldots < t_n = T.$$

Далее, точки (t_i, x_i) $(i=1, \ldots, n)$ интерполируют периодическим кубическим сплайном $x(t)$ периода T. Это значит, что в каждом интервале $t_i \leqslant t \leqslant t_{i+1}$ функция $x(t)$ является кубическим полиномом и что

$$x(0) = x(T), \quad x'(0) = x'(T) \text{ и } x''(0) = x''(T).$$

Аналогичным образом интерполируют точки (t_i, y_i) периодическим кубическим сплайном $y(t)$.

Нужная кривая задается теперь параметрически функциями $x(t)$ и $y(t)$.

Ваша задача — так переделать подпрограмму SPLINE, чтобы можно было вычислять периодические кубические сплайны $x(t)$ и $y(t)$. Проверьте программу, беря в качестве исходных данных вершины правильных n-угольников, $n=3,4$. Напоминают ли ваши решения формой окружности?

4.12. Пусть $x_i = i - 11$, $i=1, 2, \ldots, 21$. Положим

$$y_i = \begin{cases} 1, & i = 11, \\ 0, & i \neq 11. \end{cases}$$

Вычислите и распечатайте кубический сплайн через 21 точку (x_i, y_i) и начертите его график.

То же сделайте для лагранжева полинома $l_{11}(x)$, интерполирующего эти же точки.

Сравните эти важные функции.

При вычерчивании графиков можно ограничиться областью $x \geqslant 0$. Почему?

4.13. Рассмотрим функцию s, определенную формулами

$$s(x) = \begin{cases} 1-2x, & x < -3, \\ 28+25x+9x^2+x^3, & -3 \leqslant x < -1, \\ 26+19x+3x^2-x^3, & -1 \leqslant x < 0, \\ 26+19x+3x^2-2x^3, & 0 \leqslant x < 3, \\ -163+208x-60x^2+5x^3, & 3 \leqslant x < 4, \\ 157-32x, & 4 \leqslant x. \end{cases}$$

Покажите, что s есть естественный кубический сплайн с узлами $\{-3, -1, 0, 3, 4\}$. Сформулируйте каждое из свойств s, которое необходимо для справедливости этого утверждения.

4.14. Переделайте подпрограмму SEVAL в подпрограмму с заголовком

SUBROUTINE SEVAL (N, U, X, Y, B, C, D, S, SP).

Она должна вычислять сплайн и его производную в точке U и помещать вычисленные значения в S и SP. Проверьте вашу программу для каких-нибудь тестовых данных, где значение производной известно. Посмотрите также часть а) в упр. 7.7.

4.15. Предположим, вам заданы n точек $(x_i, f(x_i))$, $i=1, \ldots, n$, и нужно найти точки, в которых $f'(x)=0$.

а) Объясните, почему не слишком хороша идея проинтерполировать посредством полинома $p(x)$ степени $n-1$, а затем решить уравнение $p'(x)=0$?

б) Опишите эффективный алгоритм для решения этой задачи, основанный на использовании кубических интерполяционных сплайнов.

в) Напишите программу, использующую ваш метод и подпрограмму SPLINE. Проверьте вашу программу на данных упр. 7.7.

5. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

Методы приближения функций типа описанных в предыдущей главе составляют основу многих численных алгоритмов. Здесь мы будем заниматься главным образом использованием таких приближений для вычисления интегралов. Будет также сделано несколько замечаний относительно вычисления производных.

Существует много различных машинных методов интегрирования и дифференцирования. Метод, более всего подходящий для данной конкретной задачи, в значительной степени зависит от информации о соответствующей функции. Существенный интерес представляют задачи для одной функции $f(x)$ одного действительного переменного x, заданной на интервале $[a, b]$; их можно разделить на четыре широкие категории:

1. Значения функции $f(x)$ заданы только на *фиксированном* конечном множестве точек x_i интервала $[a, b]$.
2. Функция $f(x)$ определена и может быть вычислена для любого *действительного* x из интервала $[a, b]$.
3. Определение функции может быть аналитически продолжено на *комплексные* значения x.
4. Имеется явная формула для $f(x)$, пригодная для символического манипулирования.

Функции, входящие в первую категорию, могут происходить из экспериментальных измерений при значениях x_i, зачастую неравномерно распределенных, или получены из таблиц, где x_i расположены равномерно.

Для функций из первых двух категорий численное дифференцирование по-существу является более трудной задачей, чем численное интегрирование, потому что численному дифференцированию свойственна тенденция увеличивать любую ошибку, присутствующую в данных, в то время как численное интегрирование обычно сглаживает и уменьшает такие ошибки. Если значения функции известны или могут быть вычислены с большой точностью и если требуются производные относительно невысо-

кого порядка, то часто вполне удовлетворительны методы, основанные на интерполяции сплайнами или полиномами. Но если нужны производные высокого порядка или если значения функции *зашумлены*, то результаты могут быть очень неточны.

Примером функции, входящей в третью категорию, могло бы быть сложное выражение, составленное из тригонометрических и других элементарных функций. Для задач этого типа следует извлекать выгоду из их комплексного расширения, если оно доступно. В подобной ситуации производные функции f можно выразить через комплексные контурные интегралы, и сглаживающий эффект численного интегрирования можно использовать для получения хороших приближений к производным высокого порядка; см. Линес, Моулер (1967) и Линес, Санде (1971).

Для задач четвертой категории символическое машинное дифференцирование проще символического интегрирования, так же как и в элементарном анализе (Мозес (1972)). Символические вычисления этого рода станут в будущем вполне обыденными; однако сейчас лишь немногие пользуются программами символического манипулирования.

Для численного приближения определенных интегралов часто используется термин *квадратура*, чтобы избежать путаницы с численным *интегрированием* обыкновенных дифференциальных уравнений. Остальная часть этой главы будет посвящена квадратурам функций первых двух категорий.

5.1. Формулы прямоугольников и трапеций

Пусть $[a, b]$ — конечный интервал оси x, разбитый на n подынтервалов, называемых *элементарными отрезками*[1]), $[x_i, x_{i+1}]$, $i=1, \ldots, n$. Предположим, что $x_1=a$, $x_{n+1}=b$ и $x_1<x_2<\ldots<x_{n+1}$. Через $h_i=x_{i+1}-x_i$ обозначим длину i-го элементарного отрезка.

Пусть $f(x)$ — функция, определенная на $[a, b]$. Предположим, что нужно найти приближение к определенному интегралу

$$I(f)=\int_a^b f(x)\,dx.$$

Ясно, что $I(f)$ можно представить суммой интегралов по элементарным отрезкам

$$I(f)=\sum_{i=1}^{n} I_i,$$

[1]) В оригинале — panels.— *Прим. перев.*

где

$$I_i = I_i(f) = \int_{x_i}^{x_{i+1}} f(x)\,dx.$$

Квадратурной формулой называется всякая простая формула, аппроксимирующая отдельный интеграл I_i. *Составная квадратурная формула* — это формула, дающая приближение к $I(f)$ в виде суммы приближений по данной квадратурной формуле к отдельным интегралам I_i. Двумя простейшими квадратурными формулами являются формула прямоугольников и формула трапеций. В некоторых случаях они входят также и в число самых эффективных.

Формула прямоугольников использует значения функции в средних точках элементарных отрезков

$$y_i = \frac{x_i + x_{i+1}}{2}, \quad i = 1, \ldots, n.$$

Она аппроксимирует каждый интеграл I_i площадью прямоугольника с основанием h_i и высотой $f(y_i)$. Это дает

$$I_i \approx h_i f(y_i)$$

и, следовательно, *составную формулу прямоугольников*

$$R(f) = \sum_{i=1}^{n} h_i f(y_i).$$

Формула трапеций использует значения функции в концевых точках элементарных отрезков. Она аппроксимирует интеграл I_i площадью трапеции с основанием h_i и высотой, изменяющейся линейно от $f(x_i)$ слева до $f(x_{i+1})$ справа. Это приводит к приближенному равенству

$$I_i \approx h_i \frac{f(x_i) + f(x_{i+1})}{2},$$

следовательно, имеем *составную формулу трапеций:*

$$T(f) = \sum_{i=1}^{n} h_i \frac{f(x_i) + f(x_{i+1})}{2}.$$

Легко показать, что если $f(x)$ непрерывна (или даже просто интегрируема по Риману) на $[a, b]$ и если $h = \max\limits_i h_i$, то

$$\lim_{h \to 0} R(f) = I(f)$$

и

$$\lim_{h \to 0} T(f) = I(f).$$

Таким образом, обе формулы сходятся, когда длины интервалов бесконечно убывают. Важнейшим практическим вопросом является: насколько *быстро* они сходятся?

Формула прямоугольников основана на кусочно-постоянной (или степени нуль) интерполяции, в то время как формула трапеций использует кусочно-линейную (или степени один) интерполяцию. Можно было бы поэтому ожидать, что формула трапеций точнее, чем формула прямоугольников. Например, пусть $[a, b]=[0,1]$, $n=1$ (так что имеется лишь один элементарный отрезок) и $f(x)=x$. Очевидно, что формула трапеций дает точный результат, поскольку линейная функция, интерполирующая $f(x)$ в любых двух точках, совпадает с $f(x)$ всюду. Однако и формула прямоугольников также дает точный результат, несмотря на то что функция-константа, интерполирующая $f(x)$ при $x=\frac{1}{2}$, не совпадает с $f(x)$ ни в какой другой точке. Средняя же ошибка интерполяции по $[0,1]$ равна нулю.

Является ли этот простой пример типичным или же формуле прямоугольников случайно повезло? Чтобы ответить на эти вопросы, мы должны предпринять тщательный анализ ошибок, основанный на некоторых предположениях относительно $f(x)$. Чтобы результатами этого анализа можно было пользоваться и в последующих параграфах, в него будут включены некоторые избыточные члены.

Предположим, что f имеет пять непрерывных производных и что значения этих производных не слишком велики. Рассмотрим элементарный отрезок $[x_i, x_{i+1}]$. Разложение по формуле Тейлора для $f(x)$ относительно центра этого элементарного отрезка y_i имеет вид

$$f(x)=f(y_i)+(x-y_i)f'(y_i)+\frac{1}{2}(x-y_i)^2 f''(y_i)$$
$$+\frac{1}{6}(x-y_i)^3 f'''(y_i)+\frac{1}{24}(x-y_i)^4 f^{IV}(y_i)+\dots .$$

Предположения относительно f означают, что остаточный член, обозначенный многоточием, по величине меньше явно выписанных членов.

Чтобы проинтегрировать этот ряд по $[x_i, x_{i+1}]$, заметим, что

$$\int_{x_i}^{x_{i+1}} (x-y_i)^p\,dx=\begin{cases} h_i, & p=0,\\ 0, & p=1,\\ \frac{h_i^3}{12}, & p=2,\\ 0, & p=3,\\ \frac{h_i^5}{80}, & p=4. \end{cases}$$

Отметим, что интегралы нечетных степеней равны нулю. Следовательно,

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = h_i f(y_i) + \frac{1}{24} h_i^3 f''(y_i) + \frac{1}{1920} h_i^5 f^{IV}(y_i) + \ldots .$$

Это показывает, что, когда h_i мало, ошибка формулы прямоугольников на элементарном отрезке есть $\frac{1}{24} h_i^3 f''(y_i)$ плюс члены более высокого порядка.

Возвращаясь к разложению Тейлора и подставляя в него $x=x_i$ и $x=x_{i+1}$, получаем

$$f(x_i) = f(y_i) - \frac{1}{2} h_i f'(y_i) + \frac{1}{8} h_i^2 f''(y_i) - \frac{1}{48} h_i^3 f'''(y_i) + \frac{1}{384} h_i^4 f^{IV}(y_i) + \ldots,$$

$$f(x_{i+1}) = f(y_i) + \frac{1}{2} h_i f'(y_i) + \frac{1}{8} h_i^2 f''(y_i) + \frac{1}{48} h_i^3 f'''(y_i) + \frac{1}{384} h_i^4 f^{IV}(y_i) + \ldots$$

таким образом,

$$\frac{f(x_i)+f(x_{i+1})}{2} = f(y_i) + \frac{1}{8} h_i^2 f''(y_i) + \frac{1}{384} h_i^4 f^{IV}(y_i) + \ldots .$$

Объединяя это с разложением интеграла, находим

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = h_i \frac{f(x_i)+f(x_{i+1})}{2} - \frac{1}{12} h_i^3 f''(y_i) - \frac{1}{480} h_i^5 f^{IV}(y_i) + \ldots .$$

Это показывает, что при малых h_i ошибка формулы трапеций на элементарном отрезке равна $-\frac{1}{12} h_i^3 f''(y_i)$ плюс члены более высокого порядка.

Общая ошибка каждой из формул есть сумма ошибок на отдельных элементарных отрезках. Пусть

$$E = \frac{1}{24} \sum_{i=1}^{n} h_i^3 f''(y_i),$$

$$F = \frac{1}{1920} \sum_{i=1}^{n} h_i^5 f^{IV}(y_i).$$

Тогда

$$I(f) = R(f) + E + F + \ldots = T(f) - 2E - 4F + \ldots .$$

Если h_i достаточно малы, то $h_i^5 \ll h_i^3$ и, если только f^{IV} не слишком плохо ведет себя, $F \ll E$.

Из этих результатов можно извлечь несколько важных следствий. Во-первых, для многих функций $f(x)$ формула прямоугольников примерно вдвое точнее формулы трапеций. Мы выведем вскоре другие формулы, для которых погрешность выражается более высокими степенями h_i, так что множитель 2 не имеет большого значения — и все же это многим кажется удивительным.

Следующий вывод заключается в том, что разность значений, полученных по формулам прямоугольников и трапеций, можно использовать для оценки погрешности каждой из них. Однако эта оценка не безукоризнена; может случиться, что обе формулы дадут один и тот же и тем не менее неверный результат.

Еще одно следствие касается эффекта от изменения числа элементарных отрезков. Проще всего поделить каждый элементарный отрезок пополам. (Мы предполагаем, что значения функции заданы или могут быть вычислены также и в новых точках). Каждое h_i уменьшится в два раза, и, следовательно, все h_i^3, входящие в главный член E погрешности, уменьшатся в 8 раз. Однако общее число элементарных отрезков удвоится, так что в целом член E уменьшится примерно в 4 раза. Коэффициент уменьшения ошибки обычно не равен в точности 4, поскольку f'', как правило, не является константой и сказывается также влияние членов более высокого порядка. Однако при реальных вычислениях с функциями, имеющими непрерывные ограниченные вторые производные, можно ожидать, что *удвоение* числа элементарных отрезков для любой формулы — прямоугольников или трапеций — приблизительно *учетверяет* точность. Разность результатов, полученных по любой формуле до и после удвоения числа элементарных отрезков, можно использовать, чтобы оценить погрешность или уточнить вычисленный результат.

Прием многократного удвоения числа элементарных отрезков и оценки погрешности можно запрограммировать и получить метод, который автоматически определяет элементарные отрезки так, чтобы приближенное значение интеграла вычислялось с предписанной точностью. Этот прием можно применять и к другим, более точным квадратурным формулам. Один такой метод будет рассмотрен в § 5.4.

5.2. Сплайн-квадратура

С помощью кубической сплайн-интерполяции можно получить интересную и полезную квадратурную формулу. Для $x_i \leqslant x \leqslant x_{i+1}$ зададим

$$s(x) = f_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

— кубический сплайн, который интерполирует $f(x)$ в узлах x_i.

Тогда при $a=x_1$ и $b=x_{n+1}$

$$\int_a^b f(x)\,dx \approx \int_a^b s(x)\,dx = \sum_{i=1}^{n} h_i f_i + \frac{1}{2} h_i^2 b_i + \frac{1}{3} h_i^3 c_i + \frac{1}{4} h_i^4 d_i.$$

Коэффициенты b_i, c_i и d_i можно найти по подпрограмме SPLINE из § 4.5. Поскольку здесь $n+1$ узлов, подпрограмму SPLINE нужно вызывать при значении $N=n+1$.

Однако полезно записать эту формулу по-иному. Напомним, что, согласно § 4.4, $s(x)$ на $[x_i, x_{i+1}]$ можно представить в виде

$$s(x) = wf_{i+1} + \bar{w}f_i + h_i^2[(w^3-w)\sigma_{i+1} + (\bar{w}^3-\bar{w})\sigma_i],$$

где $w=1-\bar{w}=(x-x_i)/h_i$ и

$$\sigma_i = \frac{s''(x_i)}{6} = \frac{c_i}{3}.$$

Коэффициенты σ_i вычисляются из симметричной трехдиагональной системы уравнений, выведенной в § 4.4. Чтобы получить квадратурную формулу, заметим, что

$$\int_{x_i}^{x_{i+1}} s(x)\,dx = h_i \int_0^1 s(w)\,dw,$$

$$\int_0^1 w\,dw = \frac{1}{2},$$

и

$$\int_0^1 (w^3-w)\,dw = -\frac{1}{4}.$$

Следовательно,

$$\int_{x_i}^{x_{i+1}} s(x)\,dx = h_i \frac{f_i+f_{i+1}}{2} - h_i^3 \frac{\sigma_i+\sigma_{i+1}}{4}.$$

Другими словами, формула сплайн-квадратуры есть формула трапеций плюс поправочный член, содержащий коэффициенты σ_i.

Для реального вычисления удобна формула

$$\int_a^b s(x)\,dx = \sum_{i=1}^{n} h_i \frac{f_i+f_{i+1}}{2} - h_i^3 \frac{c_i+c_{i+1}}{12}.$$

При счете нужны только массивы данных X и F, а также массив вторых производных C, вычисляемый подпрограммой SPLINE. Однако еще два массива B и D требуются подпрограммой SPLINE для хранения промежуточных результатов.

Чтобы оценить точность сплайн-квадратуры, заметим, что, исключая патологию в поведении $f''(x)$, справедливо соотношение

$$h_i^3 \frac{c_i + c_{i+1}}{12} = h_i^3 \frac{s''(x_i) + s''(x_{i+1})}{24} \approx \frac{h_i^3}{12} f''(y_i).$$

Таким образом, поправочный член, даваемый сплайном, аппроксимирует ошибку самой формулы трапеций.

У нас не было пока сколько-нибудь обширной практики использования этого метода, однако, основываясь на приведенном анализе, мы рассчитываем, что он будет вполне успешным. Поэтому мы рекомендуем его для тех случаев, когда узлы фиксированы, а автоматические методы, описываемые в последующих параграфах, неприменимы.

Нужно отметить, что сплайн-квадратура *не* является обычной квадратурной формулой. Если записать ее в виде

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n+1} \alpha_i f(x_i),$$

то каждый коэффициент α_i зависит весьма сложным образом от *всех* значений функции $f(x_i)$.

5.3. Формула Симпсона

В § 5.1 составные формулы прямоугольников и трапеций были определены как

$$R(f) = \sum_{i=1}^{n} h_i f\left(\frac{x_i + x_{i+1}}{2}\right)$$

и

$$T(f) = \sum_{i=1}^{n} h_i \frac{f(x_i) + f(x_{i+1})}{2}.$$

Далее, было показано, что погрешности этих формул выражаются так:

$$I(f) - R(f) = E + F + \ldots$$

и

$$I(f) - T(f) = -2E - 4F + \ldots,$$

где

$$E = \frac{1}{24} \sum_{i=1}^{n} h_i^3 f''(y_i),$$

$$F = \frac{1}{1920} \sum_{i=1}^{n} h_i^5 f^{\mathrm{IV}}(y_i).$$

Комбинируя эти две формулы надлежащим образом, мы можем получить новую формулу, для которой погрешность уже не содержит E. Поскольку $R(f)$ примерно вдвое точнее, чем $T(f)$, то подходящим будет выбор

$$S(f)=\frac{2}{3}R(f)+\frac{1}{3}T(f).$$

Расписывая это, чтобы выявить составляющие члены, получим

$$S(f)=\sum_{i=1}^{n}\frac{1}{6}h_i\left[f(x_i)+4f\left(\frac{x_i+x_{i+1}}{2}\right)+f(x_{i+1})\right].$$

Многие читатели узнают здесь *составную формулу Симпсона*. Она может быть также выведена интегрированием кусочно-параболической функции, интерполирующей заданные узлы.

Погрешность формулы Симпсона можно получить непосредственно из выражений погрешностей формул прямоугольников и трапеций:

$$\begin{aligned}I(f)-S(f)&=\frac{2}{3}[I(f)-R(f)]+\frac{1}{3}[I(f)-T(f)]\\&=\left(\frac{2}{3}-\frac{2}{3}\right)E+\left(\frac{2}{3}-\frac{4}{3}\right)F+\ldots\\&=-\frac{2}{3}F+\ldots=-\frac{1}{2880}\sum_{i=1}^{n}h_i^5f^{\mathrm{IV}}(y_i)+\ldots.\end{aligned}$$

Заметьте, что, хотя $S(f)$ основана на интерполяции степени два, в выражение погрешности входит четвертая производная и, следовательно, формула Симпсона точна для кубических функций. Другими словами, как и формула прямоугольников, формула Симпсона получает один «дополнительный» порядок точности. (Читатель может проверить, что формула Симпсона точна в случае кубических функций.)

Если длину каждого элементарного отрезка уменьшить вдвое, то каждое h_i^5 в формуле погрешности уменьшится в 32 раза. Общее число членов возрастет в два раза, так что погрешность в целом уменьшится примерно в 16 раз. Этот факт важен для программ, автоматически выбирающих число и величину элементарных отрезков.

Прием комбинирования двух приближений с погрешностями одного порядка с целью получить гораздо более точное приближение можно применять повторно. Например, значения $S(f)$ для двух различных наборов элементарных отрезков можно скомбинировать так, что будет получено новое значение, отличающееся от $I(f)$ величиной, зависящей от h_i^7 и $f^{(6)}(x)$. Проведение этой

процедуры систематическим образом приводит к популярному методу, называемому *квадратурами Ромберга.*

Ту же идею можно применить к численному дифференцированию, к численному решению дифференциальных уравнений — вообще к любому численному процессу, аппроксимирующему пределы, поведение погрешности которого известно. Этот общий прием называется *экстраполяцией Ричардсона* или *экстраполяцией к пределу* [1]). Прекрасный обзор истории и приложений этого метода дан в статье Джойс (1971). Мы отметим еще один пример экстраполяции Ричардсона в § 6.4.

Любая попытка оценить сравнительные достоинства методов, описанных до сих пор, именно формул прямоугольников, трапеций, Симпсона и сплайн-квадратуры, связана с вопросами типа «Что больше, $h^2f''(x)$ или $h^4f^{IV}(y)$?» Ответ и соответствующие выводы, разумеется, зависят от природы интегрируемой функции. К счастью, для ответа на некоторые вопросы можно привлечь машину. Получающиеся *адаптивные квадратурные подпрограммы* обсуждаются в следующих двух параграфах.

5.4. Адаптивные квадратурные программы

Адаптивная квадратурная программа — это алгоритм численных квадратур, использующий одну или две основные квадратурные формулы и автоматически определяющий величины подынтервалов так, чтобы вычисленный результат удовлетворял предписанной точности. В разных частях интервала могут использоваться сетки различных размеров, сравнительно грубые там, где интегрируемая функция гладкая и меняется медленно, и сравнительно мелкие в областях, где интегрирование становится затруднительным. Таким образом, делается попытка получить результат с предписанной точностью за (по возможности) малое машинное время.

Первая адаптивная программа, опубликованная Маккиманом (1962), была основана на формуле Симпсона. Эта и ей подобные программы оказались очень успешными на практике. Очень полный анализ метода и некоторые предложения относительно модификаций были сделаны Линесом (1969а). Райсу (1975) и Малкольму и Симпсону (1975) принадлежат замечания относительно машинной реализации. В этом параграфе будут обсуждены общие принципы адаптивных программ. В следующем параграфе описывается конкретная такая программа.

Пользователь подобной программы указывает конечный интервал $[a, b]$, заготавливает подпрограмму, вычисляющую $f(x)$ для

[1]) В оригинале — deferred approach to the limit.— *Прим. перев.*

любого x из этого интервала, и выбирает допустимую точность ε. Программа пытается вычислить величину Q, такую, что

$$\left|Q-\int_a^b f(x)\,dx\right|\leqslant\varepsilon.$$

Программа может сделать вывод, что предписанная точность недостижима, получить наилучший возможный для нее результат и выдать оценку реально достигнутой точности.

При оценке эффективности квадратурных программ обычное предположение состоит в том, что бóльшая часть стоимости счета приходится на вычисление подынтегральной функции $f(x)$. Таким образом, если для интегрирования данной функции имеются две подпрограммы и обе дают ответы примерно одинаковой точности, то подпрограмма, требующая меньшего количества вычислений функции, рассматривается как более эффективная для данной конкретной задачи.

Всегда можно сконструировать подынтегральную функцию $f(x)$, которая могла бы одурачить данную программу, приводя к совершенно неверному результату; однако для хороших адаптивных программ класс таких примеров был сужен, насколько это возможно без неоправданного усложнения логики или потери эффективности на разумных задачах.

В процессе вычислений интервал $[a, b]$ разбивается на подынтервалы $[x_i, x_{i+1}]$. В большинстве программ каждый подынтервал получается делением пополам подынтервала, полученного на более раннем этапе вычислений. Реальное число подынтервалов, так же как их расположение и длины, зависит от подынтегральной функции $f(x)$ и требуемой точности ε. При нумерации условливаются, что $x_1=a$, и программа определяет n так, что $x_{n+1}=b$. В этом параграфе длина подынтервала будет обозначаться через $h_i=x_{i+1}-x_i$.

Типичная схема применяет к каждому подынтервалу две различные квадратурные формулы. Обозначим соответствующие два результата через P_i и Q_i. Например, схемы, основанные на формуле Симпсона, используют основную формулу (два элементарных отрезка)

$$P_i=\frac{h_i}{6}\left[f(x_i)+4f\left(x_i+\frac{h_i}{2}\right)+f(x_i+h_i)\right]$$

и составную формулу (четыре элементарных отрезка)

$$Q_i=\frac{h_i}{12}\left[f(x_i)+4f\left(x_i+\frac{h_i}{4}\right)+2f\left(x_i+\frac{h_i}{2}\right)+4f\left(x_i+\frac{3h_i}{4}\right)+f(x_i+h_i)\right].$$

Выражения P_i и Q_i являются приближениями к

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx.$$

Основная идея адаптивной квадратуры состоит в сравнении двух приближений P_i и Q_i и получении при этом оценки их точности. Если точность приемлема, то одно из чисел принимается за значение интеграла по данному подынтервалу. Если же точность недостаточна, то подынтервал делится на две или более частей и процесс повторяется для меньших подынтервалов.

Сокращение общего количества вычислений функции обычно достигается тем, что две формулы, дающие P_i и Q_i, используют значения подынтегральной функции в некоторых общих точках. Например, в случае формулы Симпсона Q_i требует пяти значений функции; три из них используются и в P_i. Следовательно, обработка нового подынтервала требует лишь двух новых значений функции. В остальной части этой главы будем считать, что Q_i получается двукратным применением формулы для P_i, т. е. применением ее к каждому полуинтервалу. Это общий прием, наиболее упрощающий анализ.

Предположим еще, что формула для P_i дает точный результат, если интегрируемая функция есть полином степени $p-1$, или, другими словами, если p-я производная $f^{(p)}(x)$ тождественно равна нулю. Раскладывая подынтегральную функцию в ряд Тейлора относительно середины подынтервала, можно показать, что для некоторой константы c

$$I_i - P_i = ch^{p+1} f^{(p)}\left(x_i + \frac{h_i}{2}\right) + \dots .$$

Показатель h_i равен $p+1$, а не p, потому что длина подынтервала равна h_i. Из предположения о том, что Q_i есть сумма двух P, взятых по двум подынтервалам длины $h_i/2$, следует:

$$I_i - Q_i = c\left(\frac{h_i}{2}\right)^{p+1}\left[f^{(p)}\left(x_i + \frac{h_i}{4}\right) + f^{(p)}\left(x_i + \frac{3h_i}{4}\right)\right] + \dots .$$

Поскольку

$$f^{(p)}\left(x_i + \frac{h_i}{4}\right) + f^{(p)}\left(x_i + \frac{3h_i}{4}\right) = 2f^{(p)}\left(x_i + \frac{h_i}{2}\right) + \dots ,$$

то обе ошибки связаны соотношением

$$I_i - Q_i = \frac{2}{2^{p+1}}(I_i - P_i) + \dots = \frac{1}{2^p}(I_i - P_i) + \dots .$$

Оно показывает, что деление подынтервала пополам уменьшает ошибку примерно в 2^p раз. Разрешая относительно неизвест-

ного I_i и переупорядочивая члены, получаем

$$Q_i - I_i = \frac{1}{2^p - 1}(P_i - Q_i) + \dots .$$

Другими словами, ошибка более точного приближения Q_i приблизительно в $2^p - 1$ раз меньше разности между двумя приближениями.

Основная операция типичной программы состоит в делении каждого подынтервала пополам до тех пор, пока не будет выполнено следующее неравенство:

$$\frac{1}{2^p - 1}|P_i - Q_i| \leqslant \frac{h_i}{b-a}\varepsilon,$$

где ε — указанная пользователем граница допустимой точности. Если весь интервал $[a, b]$ можно покрыть n подынтервалами, для которых это справедливо, то программа выдаст результат

$$Q = \sum_{i=1}^{n} Q_i.$$

Комбинируя полученные соотношения и игнорируя члены более высокого порядка, находим

$$\left|Q - \int_a^b f(x)\,dx\right| = \left|\sum_{i=1}^{n}(Q_i - I_i)\right| \leqslant \sum_{i=1}^{n}|Q_i - I_i|$$

$$\leqslant \frac{1}{2^p - 1}\sum_{i=1}^{n}|P_i - Q_i|$$

$$\leqslant \frac{1}{2^p - 1}\cdot\frac{2^p - 1}{b-a}\varepsilon\sum_{i=1}^{n}h_i = \varepsilon,$$

что и требовалось.

Этот анализ предполагает непрерывность $f^{(p)}(x)$ и пропорциональность ошибки величине $h_i^{p+1}f^{(p)}(x)$. Даже если эти предположения не вполне выполняются, программа может тем не менее давать неплохой результат, находящийся в пределах желаемой точности, хотя локальное поведение процесса будет иным.

Для простоты мы предположили также, что программа использует критерий абсолютной ошибки, т. е.

$$\left|Q - \int f\right| < \varepsilon.$$

Во многих ситуациях предпочтительней тот или иной вид ограничения, наложенного на относительную ошибку, которая не зависит от масштабирующих множителей в f. Проверка прямого

относительного критерия

$$\frac{\left|Q-\int f\right|}{\left|\int f\right|}<\varepsilon$$

усложнена рядом обстоятельств. Знаменатель $\int f$ может быть нулем. Но на практике критерий, возможно, не удастся удовлетворить и в том случае, когда знаменатель просто близок к нулю за счет того, что положительные значения f на одной части интервала почти погашают отрицательные значения f на другой. Кроме того, хорошее приближение к значению знаменателя доступно лишь на заключительном этапе вычислений.

Некоторые программы используют критерий, основанный на $\int|f|$,

$$\frac{\left|Q-\int f\right|}{\int|f|}<\varepsilon.$$

Знаменатель не может быть нулем, если f отлична от тождественного нуля, и не подвержен взаимному уничтожению, связанному с осциллированием подынтегральной функции. Однако здесь все же требуется больше вычислений, хотя и то же число значений функции, а также более сложное истолкование результата.

Пользователи адаптивных квадратурных программ должны быть осведомлены о наличии различных тестов точности и проверять документацию конкретной программы, чтобы установить, какой именно тест в ней используется.

Рисунок 5.1 иллюстрирует поведение типичной адаптивной программы для одного интересного примера. Задача состоит в интегрировании функции

$$f(x)=\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6$$

на интервале [0,1]. Использовалась программа QUANC8 на машине IBM 360 в режиме удвоенной точности; при этом RELERR$=10^{-14}$ (см. следующий параграф). К программе были добавлены операторы, вызывающие подпрограмму, которая наносит на график все точки $\{x_i,\ f(x_i))$ и $(x_i,\ 0)$, $i=1,\ \ldots,\ n$. Интегрируемая функция имеет высокий пик около точки $x=0.3$ и меньший пик около точки $x=0.9$. График соответствующей производной $f^{(10)}(x)$ также имел максимумы в этих точках. На рисунке видно, что подпрограмма выбирала очень малые интервалы в окрестности пиков и большие интервалы в отдалении от них.

Такое поведение весьма типично для программы QUANC8, так же как и для других адаптивных программ. Если бы величина интервалов, необходимая для достижения заданной точности

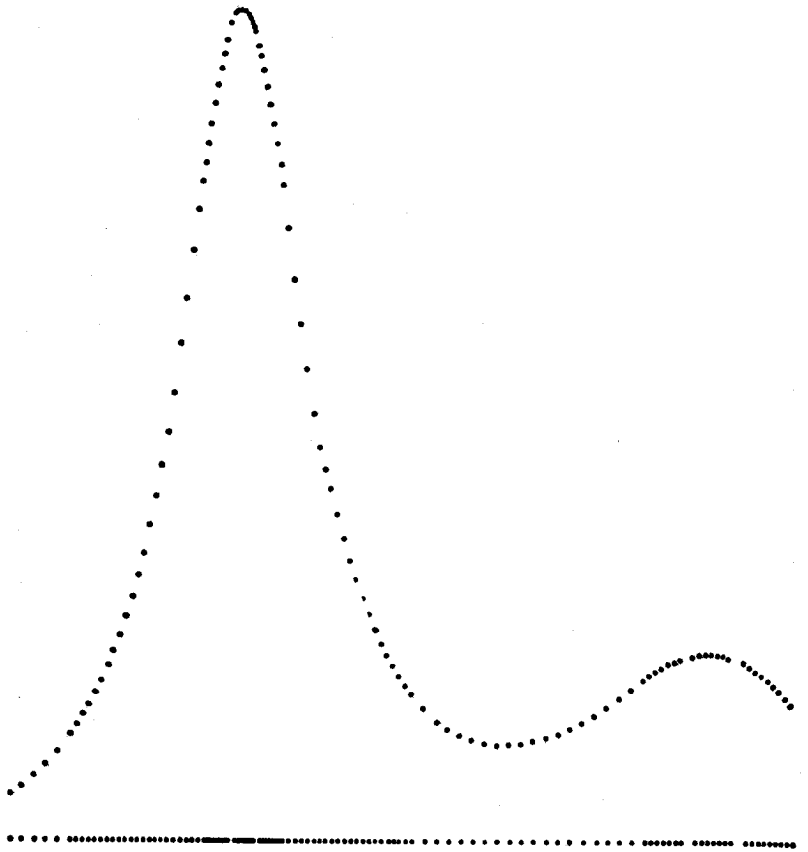


Рис. 5.1. Поведение адаптивной квадратурной программы.

вблизи пиков, использовалась по всему интервалу интегрирования, то вычисления потребовали бы значительно больше времени.

5.5. Подпрограмма QUANC8

К 1976 г. было опубликовано около дюжины достаточно эффективных адаптивных квадратурных программ. Среди наиболее широко используемых программ — SQUANK, написанная Джеймсом Линесом, и QNC7, авторы Дэвид Каханер и Рондэл Джонс.

В этом параграфе будет представлена еще одна адаптивная программа — QUANC8. Она объединяет основные идеи программ SQUANK и QNC7, но не включает некоторые из их наиболее изощренных особенностей, такие, как контроль ошибки округления и выявление сингулярностей некоторых типов. Мы полагаем, что программа QUANC8 достаточно проста для чтения и понимания и что тем не менее ее результаты для широкого круга задач вполне конкурентоспособны с результатами более сложных программ.

Пользователь программы QUANC8 должен задать подпрограмму-функцию FUN(X) для подынтегральной функции $f(x)$; нижний и верхний пределы интегрирования $A=a$ и $B=b$; и границы абсолютной и относительной ошибок ABSERR и RELERR. Выходные параметры программы: RESULT = приближение к интегралу, ERREST = оценка абсолютной ошибки в RESULT; NOFUN = число требуемых вычислений функции, и FLAG = =индикатор надежности. Если значение FLAG равно нулю, то RESULT, по всей видимости, имеет требуемую точность. Если значение FLAG не равно нулю, но мало, значение RESULT все еще может быть приемлемым. Если же значение FLAG велико, то подынтегральная функция принадлежит к одному из типов, неприемлемых для программы.

Название QUANC8 произведено из слов Quadrature, Adaptive, Newton-Cotes' 8-panel [1]. *Формулы Ньютона — Котеса* — это семейство квадратурных формул, получаемых интегрированием интерполяционных полиномов, построенных по равноудаленным узлам. Формулы прямоугольников, трапеций и Симпсона выводятся интегрированием полиномов соответственно нулевой, первой и второй степени и являются первыми тремя членами этого семейства. В данном случае формула получается интегрированием полинома восьмой степени, но так же, как и в случае формул прямоугольников и Симпсона, приобретается дополнительная точность, так что формула дает точный результат для многочленов степени 9.

В случае интегрирования по интервалу [0,1] формула дает

$$\int_0^1 f(x)\,dx \approx \sum_{k=0}^{8} w_k f\left(\frac{k}{8}\right).$$

Приближенные значения весов w_k суть

---

[1]) То есть квадратура, адаптивная, Ньютона — Котеса, 8 элементарных отрезков. Словосочетание «8-panel» мы в дальнейшем переводим как «формула 8-го порядка», имея в виду степень интерполяционного полинома, по которому она построена (точность же этой формулы, как указано в тексте, равна 9).— *Прим. перев.*

k	w_k
0	0.0349
1	0.2077
2	−0.0327
3	0.3702
4	−0.1602
5	0.3702
6	−0.0327
7	0.2077
8	0.0349

Рациональные выражения для действительных весов (умноженные на 8) включены в саму подпрограмму. Поскольку формула точна для $f(x)\equiv 1$, то веса удовлетворяют равенству

$$\sum_{k=0}^{8} w_k = 1.$$

Однако некоторые из весов отрицательны, и

$$\sum_{k=0}^{8} |w_k| = 1.4512.$$

Этот фактор важен при анализе ошибок. Например, предположим, что функция $f(x)\equiv 1$ известна лишь с некоторой ошибкой, так что $f(x)=0.99$ при $x=2/8,\ 4/8,\ 6/8$, в то время как в прочих узлах $f(x)=1.01$. Тогда формула в качестве значения интеграла даст число 1.014512. Ошибки, содержавшиеся в значениях функции, слегка увеличились в результате. Коэффициент увеличения несуществен в данной формуле, однако для формул Ньютона — Котеса, основанных на полиномах более высокой степени, он становится неприемлемым.

Мы выбрали формулу 8-го порядка с тем, чтобы длина каждого элементарного отрезка получалась из длины $b-a$ основного интервала делением на степень двойки. Поэтому на двоичной машине при вычислении узлов x_i не будет ошибок округления.

Формула 8-го порядка используется для вычисления величины P_i, определенной в предыдущем параграфе. Величина Q_i получается применением той же формулы к двум половинам интервала; тем самым используется 16 элементарных отрезков. В программе величинам P_i и Q_i соответствуют переменные QPREV и QNOW. Переменные QLEFT и QRIGHT содержат результаты, полученные по двум половинам интервала.

Поскольку основная формула точна для полиномов девятой степени, то приложим анализ предыдущего параграфа при $p=10$. Ошибка в Q_i приблизительно равна ошибке в P_i, взятой с мно-

жителем $1/(2^{10}-1)=1/1023$, при условии, что подынтегральная функция гладкая и члены более высокого порядка можно игнорировать.

Как было указано в § 5.4, результат, полученный для данного подынтервала, приемлем, если

$$\frac{1}{1023}|P_i-Q_i|\leqslant\frac{h_i}{b-a}\varepsilon.$$

В программе QUANC8 этот тест выполняется сравнением двух величин

```
ESTERR = ABS(QNOW - QPREV)/1023
TOLERR = AMAX1(ABSERR, RELERR*ABS(AREA))*
                                (STEP/STONE).
```

Отношение STEP / STONE равно $h_i/(b-a)$, а AREA есть оценка интеграла по всему интервалу, так что ε есть максимум из ABSERR и RELERR* $|\int f|$. Подынтервал приемлем, если

```
ESTERR.LE.TOLERR.
```

Если подынтервал принимается, то QNOW добавляется к выходному параметру RESULT, а ESTERR добавляется к выходному параметру ERREST. Кроме того, величина (QNOW—QPREV)/1023 прибавляется к внутренней переменной COR11, поскольку $Q_i+(Q_i-P_i)/1023$ обычно является более точной оценкой для I_i, чем само Q_i. В действительности это не что иное, как один шаг экстраполяции Ричардсона. В то время как формулы P_i и Q_i по отдельности точны для полиномов степени 9, комбинация этих двух формул дает точное значение интегралов от полиномов степени 11. Окончательное значение переменной COR11 перед самым выходом из программы прибавляется к RESULT.

При каждом делении подынтервала узлы и значения функции для правой половины запоминаются для последующего использования. Поскольку максимальный объем памяти для хранения этих величин должен быть указан заранее, то устанавливается предел LEVMAX=30 на уровень деления пополам. Когда этот максимум достигнут, то подынтервал принимается, даже если оценка ошибки слишком велика; однако ведется счет числа таких подынтервалов, и это число содержится в выходной информации в качестве целой части параметра FLAG. Длина каждого из этих подынтервалов очень мала — она равна $(b-a)/2^{30}$, так что для большинства подынтегральных функций несколько подобных подынтервалов могут быть отражены в конечном результате без серьезного ухудшения точности.

Однако если подынтегральная функция имеет разрывы или неограниченные производные или «зашумлена» ошибками округлений, то предельный уровень деления пополам может достигать-

ся очень часто. Поэтому установлен другой предел NOMAX на количество вычислений функции. Когда оказывается, что этот предел может быть превышен, параметр LEVMAX уменьшается до LEVOUT с тем, чтобы оставшаяся часть интервала могла быть обработана с меньшим, чем NOMAX, количеством вычислений функции. Кроме того, точка XO, где встретилась трудность, отмечается и величина

$$(B-XO)/(B-A)$$

содержится в выходной информации в качестве дробной части параметра FLAG. Это та часть интервала, которую предстоит обработать с меньшим пределом на уровень деления пополам. Так как

$$XO = B - (\text{дробная часть FLAG}) * (B-A),$$

то положение трудного места можно определить.

Программа QUANC8 основана на кусочно-полиномиальной аппроксимации и потому не предназначена для вычисления некоторых типов интегралов. Грубо говоря, это интегралы от функций $f(x)$, для которых некоторая производная $f^{(k)}(x)$, $k \geqslant 10$, не ограничена или не существует. Например, интегралы вида

$$\int_0^1 x^\alpha \, dx$$

при $\alpha > -1$ определены корректно, но если α не является целым числом, то QUANC8 может потребовать большого количества вычислений функции и не получить хорошую оценку ошибки.

Любой из границ погрешностей может быть задано нулевое значение; однако если обе равны нулю, то, за исключением очень простых случаев, будет достигнуто предельное количество вычислений функции.

Приводимая ниже иллюстрирующая программа вычисляет Si (2), где Si (z) есть интегральный синус:

$$\mathrm{Si}\,(2) = \int_0^2 \frac{\sin x}{x}\, dx.$$

Описание EXTERNAL в основной программе, относящееся к подпрограмме FUN, существенно. Поскольку основная программа транслируется отдельно от подпрограмм, то нет никакого другого указания на то, что FUN не является простой вещественной переменной. Данная задача очень проста, и программе QUANC8 требуется лишь 33 значения функции. Результат, полученный на IBM 360 с использованием удвоенной точности, таков:

RESULT=1.6054129768 ERREST=0.23 D-15.

Если мы изменим задачу на

$$\int_0^2 \frac{\operatorname{tg} x}{x}\,dx,$$

то поведение программы будет совершенно иным. Подынтегральная функция имеет неинтегрируемую особенность при $x=\pi/2\approx$ ≈ 1.57. QUANC8 выдает значение 91.21 для параметра FLAG. Это показывает, что для 91 интервала не было сходимости [1] и оставалось обработать 0.21 от всего интервала интегрирования, когда было обнаружено трудное место. Так как 2—0.21*2=1.58, то особенность легко находится.

```
C  ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ QUANC8
C
      REAL FUNCTION FUN(X)
      REAL X
      IF (X .EQ. 0.0) FUN=1.0
      IF (X .NE. 0.0) FUN=SIN(X)/X
      RETURN
      END
C
      EXTERNAL FUN
      REAL A, B, ABSERR, RELERR, RESULT, ERREST, FLAG
      INTEGER NOFUN
      A=0.0
      B=2.0
      RELERR=1.0E-10
      ABSERR=0.0
      CALL QUANC8(FUN, A, B, ABSERR, RELERR, RESULT,
     ERREST, NOFUN, FLAG)
      WRITE(6, 1) RESULT, ERREST
      IF (FLAG .NE. 0.0) WRITE(6,2) FLAG
    1 FORMAT(8H RESULT=, F15.10, 10H ERREST=, E10.2)
    2 FORMAT (44H WARNING.. RESULT MAY BE UNRELIABLE.
     FLAG=, F6.2)
      STOP
      END
      SUBROUTINE QUANC8(FUN, A, B, ABSERR, RELERR, RESULT,
     ERREST, NOFUN, FLAG)
C
      REAL FUN, A, B, ABSERR, RELERR, RESULT, ERREST, ELAG
      INTEGER NOFUN
C
CC    ОЦЕНИТЬ ИНТЕГРАЛ ДЛЯ FUN(X) ОТ A ДО B С ЗАДАННОЙ
CC    ПОЛЬЗОВАТЕЛЕМ ТОЧНОСТЬЮ.
CC    АВТОМАТИЧЕСКАЯ АДАПТИВНАЯ ПРОГРАММА, ОСНОВАН-
CC    НАЯ НА ФОРМУЛЕ НЬЮТОНА—КОТЕСА 8-ГО ПОРЯДКА
CC
CC    ВХОДНАЯ ИНФОРМАЦИЯ..
CC
CC    FUN       ИМЯ ПОДПРОГРАММЫ-ФУНКЦИИ FUN(X), РЕАЛИ-
C     ЗУЮЩЕЙ ПОДЫНТЕГРАЛЬНУЮ ФУНКЦИЮ
```

[1] То есть не выполнялся критерий точности.— *Прим. перев.*

```
C     A       НИЖНИЙ ПРЕДЕЛ ИНТЕГРИРОВАНИЯ
C     B       ВЕРХНИЙ ПРЕДЕЛ ИНТЕГРИРОВАНИЯ (В МОЖЕТ
C             БЫТЬ МЕНЬШЕ, ЧЕМ А)
C     RELERR  ГРАНИЦА ОТНОСИТЕЛЬНОЙ ПОГРЕШНОСТИ
C             (ДОЛЖНА БЫТЬ НЕОТРИЦАТЕЛЬНА)
C     ABSERR  ГРАНИЦА АБСОЛЮТНОЙ ПОГРЕШНОСТИ (ДОЛЖНА
C             БЫТЬ НЕОТРИЦАТЕЛЬНА)
C
C     ВЫХОДНАЯ ИНФОРМАЦИЯ..
C
C     RESULT  ПРИБЛИЖЕНИЕ К ИНТЕГРАЛУ, УДОВЛЕТВОРЯЮ-
C             ЩЕЕ, МОЖНО НАДЕЯТЬСЯ, МЕНЕЕ ЖЕСТКОЙ ИЗ
C             ДВУХ ГРАНИЦ ПОГРЕШНОСТИ.
C     ERREST  ОЦЕНКА ВЕЛИЧИНЫ ДЕЙСТВИТЕЛЬНОЙ ОШИБКИ.
C     NOFUN   ЧИСЛО ЗНАЧЕНИЙ ФУНКЦИИ, ИСПОЛЬЗОВАННЫХ
C             ПРИ ВЫЧИСЛЕНИИ RESULT.
C     FLAG    ИНДИКАТОР НАДЕЖНОСТИ. ЕСЛИ FLAG РАВЕН
C             НУЛЮ, ТО RESULT, ВЕРОЯТНО, УДОВЛЕТВОРЯЕТ
C             ЗАДАННОЙ ГРАНИЦЕ ПОГРЕШНОСТИ. ЕСЛИ
C             FLAG=XXX.YYY, ТО XXX=ЧИСЛО ИНТЕРВАЛОВ,
C             ДЛЯ КОТОРЫХ НЕ БЫЛО СХОДИМОСТИ, А
C             0.YYY=ЧАСТЬ ОСНОВНОГО ИНТЕРВАЛА, ОСТАВ-
C             ШАЯСЯ ДЛЯ ОБРАБОТКИ В ТОТ МОМЕНТ, КОГДА
C             ПРОГРАММА ПРИБЛИЗИЛАСЬ К ПРЕДЕЛЬНОМУ
C             ЗНАЧЕНИЮ ДЛЯ NOFUN.
      REAL W0, W1, W2, W3, W4, AREA, X0, F0, STONE, STEP,
     COR11, TEMP
      REAL QPREV, QNOW, QDIFF, QLEFT, ESTERR, TOLERR
      REAL QRIGHT(31), F(16), X(16), FSAVE(8, 30), XSAVE(8, 30)
      INTEGER LEVMIN, LEVMAX, LEVOUT, NOMAX, NOFIN, LEV,
     NIM, I, J
C
C     *** ЭТАП 1 *** ПРИСВОЕНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ
C     ПЕРЕМЕННЫМ, НЕ ЗАВИСЯЩИМ ОТ ИНТЕРВАЛА. ГЕНЕРИ-
C     РОВАНИЕ КОНСТАНТ.
C
      LEVMIN=1
      LEVMAX=30
      LEVOUT=6
      NOMAX=5000
      NOFIN=NOMAX-8*(LEVMAX-LEVOUT+2**(LEVOUT+1))
C
C     ЕСЛИ NOFUN ДОСТИГАЕТ ЗНАЧЕНИЯ NOFIN, ТО ТРЕВОГА
C
      W0=3956.0/14175.0
      W1=23552.0/14175.0
      W2=-3712.0/14175.0
      W3=41984.0/14175.0
      W4=-18160.0/14175.0
C
C     ПРИСВОИТЬ НУЛЕВЫЕ ЗНАЧЕНИЯ ПЕРЕМЕННЫМ СУММАМ.
C
      FLAG=0.0
      RESULT=0.0
      COR11=0.0
      ERREST=0.0
      AREA=0.0
```

```
      NOFUN=0
      IF(A .EQ. B) RETURN
C
C     ***ЭТАП 2*** ПРИСВОЕНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ
C     ПЕРЕМЕННЫМ, ЗАВИСЯЩИМ ОТ ИНТЕРВАЛА, В
C     СООТВЕТСТВИИ С ПЕРВЫМ ИНТЕРВАЛОМ
C
      LEV=0
      NIM=1
      X0=A
      X(16)=B
      QPREV=0.0
      F0=FUN(X0)
      STONE=(B-A)/16.0
      X(8)=(X0+X(16))/2.0
      X(4)=(X0+X(8))/2.0
      X(12)=(X(8)+X(16))/2.0
      X(2)=(X0+X(4))/2.0
      X(6)=(X(4)+X(8))/2.0
      X(10)=(X(8)+X(12))/2.0
      X(14)=(X(12)+X(16))/2.0
      DO 25 J=2, 16, 2
        F(J)=FUN(X(J))
   25 CONTINUE
      NOFUN=9
C
C     ***ЭТАП 3*** ОСНОВНЫЕ ВЫЧИСЛЕНИЯ
C     ТРЕБУЮТСЯ QPREV, X0, X2, X4, ..., X16, F0, F2, F4, ..., F16.
C     ВЫЧИСЛЯЮТСЯ X1, X3, ..., X15, F1, F3, ..., F15, QLEFT,
C     QRIGHT, QNOW, QDIFF, AREA.
C
   30 X(1)=(X0+X(2))/2.0
      F(1)=FUN(X(1))
      DO 35 J=3, 15, 2
        X(J)=(X(J-1)+X(J+1))/2.0
        F(J)=FUN(X(J))
   35 CONTINUE
      NOFUN=NOFUN+8
      STEP=(X(16)-X0)/16.0
      QLEFT=(W0*(F0+F(8))+W1*(F(1)+F(7))+W2*
     1      (F(2)+F(6))+W3*(F(3)+F(5))+W4*F(4))*STEP
      QRIGHT(LEV+1)=(W0*(F(8)+F(16))+W1*(F(9)+(F(15))+
     1      W2*(F(10)+F(14))+W3*(F(11)+F(13))+W4*F(12))*STEP
      QNOW=QLEFT+QRIGHT(LEV+1)
      QDIFF=QNOW-QPREV
      AREA=AREA+QDIFF
C
C     ***ЭТАП 4*** ПРОВЕРКА СХОДИМОСТИ ДЛЯ ИНТЕРВАЛА
C
      ESTERR=ABS(QDIFF)/1023.0
      TOLERR=AMAX1(ABSERR, RELERR*ABS(AREA))*(STEP/
      STONE)
      IF(LEV .LT. LEVMIN) GO TO 50
      IF(LEV .GE. LEVMAX) GO TO 62
      IF(NOFUN .GT. NOFIN) GO TO 60
      IF(ESTERR .LE. TOLERR) GO TO 70
```

```
C
C       ***ЭТАП 5*** СХОДИМОСТИ НЕТ
C       УСТАНОВИТЬ СЛЕДУЮЩИЙ ИНТЕРВАЛ.
C
   50   NIM=2*NIM
        LEV=LEV+1
C
C       ЗАПОМНИТЬ ЭЛЕМЕНТЫ, ОТНОСЯЩИЕСЯ К ПРАВОЙ
C       ПОЛОВИНЕ ИНТЕРВАЛА, ДЛЯ БУДУЩЕГО ИСПОЛЬЗОВАНИЯ.
C
        DO 52 I=1, 8
              FSAVE(I, LEV)=F(I+8)
              XSAVE(I, LEV)=X(I+8)
   52   CONTINUE
C
C       СОБРАТЬ ЭЛЕМЕНТЫ, ОТНОСЯЩИЕСЯ К ЛЕВОЙ ПОЛОВИНЕ
C       ИНТЕРВАЛА, ДЛЯ НЕМЕДЛЕННОГО ИСПОЛЬЗОВАНИЯ.
C
        QPREV=QLEFT
        DO 55 I=1, 8
           J=-I
           F(2*J+18)=F(J+9)
           X(2*J+18)=X(J+9)
   55   CONTINUE
        GO TO 30
C
C       ***ЭТАП 6*** «ПОЖАРНЫЙ» РАЗДЕЛ
C       ЧИСЛО ЗНАЧЕНИЙ ФУНКЦИИ БЛИЗКО К ТОМУ, ЧТОБЫ
C       ПРЕВЫСИТЬ УСТАНОВЛЕННЫЙ ПРЕДЕЛ.
C
   60   NOFIN=2*NOFIN
        LEVMAX=LEVOUT
        FLAG=FLAG+(B-X0)/(B-A)
        GO TO 70
C
C       ТЕКУЩЕЕ ПРЕДЕЛЬНОЕ ЗНАЧЕНИЕ ГЛУБИНЫ ДЕЛЕНИЯ
C       ПОПОЛАМ РАВНО LEVMAX
C
   62   FLAG=FLAG+1.0
C
C       ***ЭТАП 7*** СХОДИМОСТЬ ДЛЯ ИНТЕРВАЛА ИМЕЕТ МЕСТО
C       ПРИБАВИТЬ ОЧЕРЕДНЫЕ СЛАГАЕМЫЕ К ПЕРЕМЕННЫМ
C       СУММАМ.
C
   70   RESULT=RESULT+QNOW
        ERREST=ERREST+ESTERR
        COR11=COR11+QDIFF/1023.0
C
C       УСТАНОВИТЬ СЛЕДУЮЩИЙ ИНТЕРВАЛ.
C
   72   IF(NIM .EQ. 2*(NIM/2)) GO TO 75
        NIM=NIM/2
        LEV=LEV-1
        GO TO 72
   75   NIM=NIM+1
        IF(LEV .LE. 0) GO TO 80
```

```
C
C       СОБРАТЬ ЭЛЕМЕНТЫ, НЕОБХОДИМЫЕ ДЛЯ СЛЕДУЮЩЕГО
C       ИНТЕРВАЛА.
C
        QPREV=QRIGHT(LEV)
        X0=X(16)
        F0=F(16)
        DO 78 I=1, 8
          F(2*I)=FSAVE(I, LEV)
          X(2*I)=XSAVE(I, LEV)
   78   CONTINUE
        GO TO 30
C
C       ***ЭТАП 8*** ЗАКЛЮЧИТЕЛЬНЫЕ ОПЕРАЦИИ И ВЫХОД
C
   80   RESULT=RESULT+COR11
C
C       ОБЕСПЕЧИТЬ, ЧТОБЫ ЗНАЧЕНИЕ ПЕРЕМЕННОЙ ERREST
C       БЫЛО НЕ МЕНЬШЕ УРОВНЯ ОКРУГЛЕНИЙ.
C
        IF(ERREST .EQ. 0.0)RETURN
   82   TEMP=ABS(RESULT)+ERREST
        IF (TEMP .NE. ABS(RESULT))RETURN
        ERREST=2.0*ERREST
        GO TO 82
        END
```

УПРАЖНЕНИЯ

5.1. Интеграл, определяющий функцию ошибок

$$\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt,$$

очень просто находится с помощью численных квадратур. Напишите программу, которая, используя QUANC8, печатает таблицу значений функции $\operatorname{erf}(x)$ для $x=0.0,\ 0.1,\ 0.2,\ \ldots,\ 1.9,\ 2.0$. Сравните вашу таблицу с опубликованными значениями или со значениями, полученными по надежной подпрограмме, имеющейся на вашей машине.

Вычисление каждого значения в вашей таблице производится независимо от прочих значений, и это не особенно эффективный способ генерирования подобных таблиц. Относительно более совершенного метода см. упр. 6.1.

5.2. Используя равенство

$$\pi=\int_0^1 \frac{4}{1+x^2}\,dx,$$

найдите с помощью численного интегрирования приближения к числу π.

а) Используйте формулу трапеций и формулу прямоугольников с элементарными отрезками одинаковой длины $h=1/n$, взяв $n=8,\ 32,\ 128$. Отметьте, что ошибка приблизительно пропорциональна h^2.

б) Примените сплайн-квадратуры с теми же значениями h. Будет ли ошибка приблизительно пропорциональна какой-либо степени h?

в) Используйте программу QUANC8 с различными значениями допустимой погрешности. Напечатайте соответствующие значения параметра NOFUN и сравните число точек, необходимых для получения определенной точности, с аналогичными числами в а) и б). Поскольку данная задача довольно проста, задаваемые границы погрешностей не должны влиять на число точек.

5.3. Используя определение интеграла Римана и формулу среднего значения, докажите, что приближения, получаемые по составным формулам прямоугольников и трапеций, сходятся при $h \to 0$ к интегралу. Выделите отчетливо те предположения, которые вы делаете относительно подынтегральной функции.

5.4. Предположим, что программу QUANC8 использовали для интегрирования функции, тождественно равной нулю, скажем, на интервале [0,1]. Результат, конечно, будет равен нулю. В каких точках QUANC8 потребует вычисления подынтегральной функции? Попробуйте ответить на этот вопрос, просто читая QUANC8. Затем проверьте ваш ответ, пропустив программу со следующей подпрограммой-функцией:

```
      REAL FUNCTION FUN(X)
      REAL X
      WRITE(6,1) X
1     FORMAT(F12.8)
      FUN = 0.0
      RETURN
      END
```

5.5. (а) Какой результат будет получен по программе QUANC8 для

$$\int_0^{2\pi} [1 - \cos 32x]\, dx?$$

Каков правильный ответ? Как можно было бы применить QUANC8, чтобы получить лучший результат? Опять-таки, попробуйте ответить на эти вопросы до реального выполнения программы.

(б) Предположим, что вам дана какая-либо другая квадратурная программа. Объясните, как найти гладкую подынтегральную функцию без особенностей, которая сможет одурачить эту программу.

5.6. Модифицируйте программу QUANC8 так, чтобы она давала в качестве выходных параметров или печатала общее количество вычислений функции и длину наименьшего элементарного отрезка, который она использует. Пропустите эту программу с какой-нибудь достаточно трудной задачей, например $f(x) = \sqrt{x}$ или $f(x) = 1/((x-c)^2 + \varepsilon)$, где c принадлежит интервалу интегрирования, а ε очень мало. Сравните количество реально потребовавшихся вычислений функции с тем, которое понадобилось бы, если бы наименьший элементарный отрезок был использован по всему интервалу интегрирования.

5.7. Найдите веса формулы Ньютона — Котеса 8-го порядка, потребовав, чтобы приближенное равенство

$$\int_0^1 f(x)\, dx \approx \sum_{k=0}^{8} w_k f\left(\frac{k}{8}\right)$$

было точным для $f(x) = x^p$, $p = 0, 1, \ldots, 8$. Используйте для этого DECOMP и SOLVE. Покажите, что формула точна и для $f(x) = x^9$. Покажите, что использо-

вание COR11 в QUANC8 делает результат точным и в случае $f(x)=x^{10}$ или x^{11}.

5.8. Напишите подпрограмму SPLINT, входом которой являются N узлов X(I), Y(I), I=1, . . ., N, а выходом — сплайн-приближение к интегралу

$$\int_{X(1)}^{X(N)} y(x)\,dx,$$

описанное в § 5.2. Ваша подпрограмма должна содержать обращение к SPLINE.

5.9. Какая из следующих задач окажется трудна, т. е. потребует большого количества вычислений функции, для программы QUANC8 при значении параметра RELERR=10^{-8}? Почему? Попробуйте ответить без реального вычисления интегралов.

(а) $\int_0^1 e^{x^2}\,dx.$

(б) $\int_0^2 \sin 10x\,dx.$

(в) $\int_0^1 \frac{\sin x}{x-3}\,dx.$

(г) $\int_1^5 \frac{(x-1)^{1/5}}{x^2+1}\,dx.$

(д) $\int_1^{10} \ln x\,dx.$

(е) $\int_0^4 f(x)\,dx,$ где $fx=\begin{cases} 1, & x=0, \\ \dfrac{(e^x-1)^5}{x^5}, & x\neq 0. \end{cases}$

5.10. Опишите эффективный и точный метод вычисления $\int_0^4 f(x)dx$, где

$$f(x)=\begin{cases} e^{x^2}, & 0\leqslant x\leqslant 2, \\ \dfrac{1}{4-\sin 16\pi x}, & 2<x\leqslant 4. \end{cases}$$

5.11. Характеристики адаптивной квадратурной программы можно несколько улучшить посредством приема, предложенного Дэвидом Каханером и называемого «банкированием». Банкирование использует то обстоятельство, что когда подынтервал приемлем, оценка ошибки для него обычно несколько меньше, чем заданная граница. Разность между границей и оценкой может быть «вложена в банк» и использована впоследствии для увеличения эффективной границы ошибок других подынтервалов. Критерий приемлемости подынтервала заменяется на

ESTERR.LE.TOLERR+THETA*BANK.

Значение параметра THETA выбирается между 0 и 1, чтобы ограничить максимальное «изъятие» из переменной BANK.

Модифицируйте программу QUANC8, включив в нее банкирование. Проведите эксперименты с различными значениями THETA и различными подынтегральными функциями.

Для значений ABSERR=0. и RELERR=1. D—15 пропустите вашу модифицированную подпрограмму QUANC8 в варианте с двойной точностью для интегралов

$$\int_0^1 \frac{dx}{(x-0.1)^2+0.0001}$$

и

$$\int_1^0 \frac{dx}{(x-0.1)^2+0.0001}.$$

Вычислите каждый интеграл для следующих значений THETA: 0.,.1,.2, . . ., 1. Заметьте, что в случае THETA=0. критерий приемлемости подынтервала тот же, что и для немодифицированной программы QUANC8. Почему банкирование дает для второго интеграла лучшие результаты, чем для первого?

Значения THETA, бóльшие 1, можно использовать для «кредитования», которое должно быть «погашено» банку последующими подынтервалами. Попробуйте использовать кредитование в указанных выше задачах.

5.12. Типовой задачей прикладной математики является решение интегрального уравнения

$$f(x)+\int_a^b K(x,\ t)\,y(t)\,dt=y(x),$$

где функции $f(x)$ и $K(x,\ t)$ заданы и задача состоит в вычислении $y(x)$.

Если мы приблизим интеграл посредством квадратурной формулы

$$\int_a^b K(x,\ t)y(t)\,dt \approx \sum_{i=1}^{n} \alpha_i K(x,\ t_i)\,y(t_i),$$

то интегральное уравнение превратится в систему линейных алгебраических уравнений:

$$f(t_j)+\sum_{i=1}^{n} \alpha_i K(t_j,\ t_i)\,y(t_i)=y(t_j), \quad j=1,\ 2,\ \ldots,\ n.$$

Ее решение $y(t_i)$, $i=1, \ldots, n$, есть искомое дискретное приближение к функции $y(t)$.

Используя формулу Симпсона, найдите приближенное решение интегрального уравнения

$$\frac{4x^3+5x^2-2x+5}{8(x+1)^2}+\int_0^1 \left(\frac{1}{1+t}-x\right) y(t)\,dt=y(x).$$

Для полученного дискретного приближения к функции $y(t)$ постройте сплайн-функцию посредством подпрограммы SPLINE из гл. 4. Сравните вычисленную сплайн-функцию с точным решением

$$y(x)=(1+x)^{-2}$$

в различных (табулированных и нетабулированных) точках интервала [0,1].

6. ЗАДАЧА КОШИ ДЛЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

6.1. Постановка задачи

Дифференциальное уравнение первого порядка можно записать в виде

$$y' = f(y, t).$$

Это уравнение имеет семейство решений $y(t)$. Например, если $f(y, t)=y$, то для произвольной константы C функция $y(t)=Ce^t$ является решением. Выбор начального значения, скажем $y(0)$, служит для выделения одной из кривых семейства, как показано на рис. 6.1. Начальное значение зависимого переменного

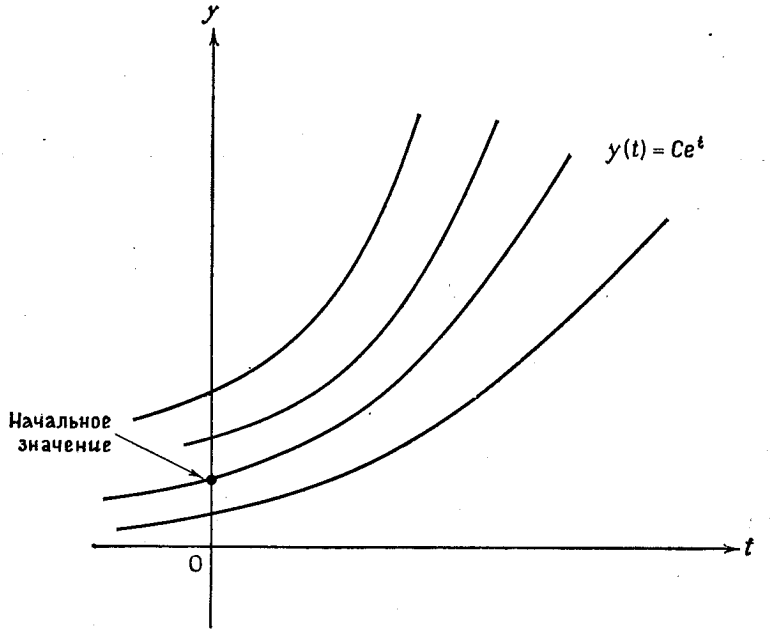


Рис. 6.1. Семейство решений уравнения $y'=y$.

может быть задано для любого значения t_0 независимого переменного. Однако часто считают, что выполнено преобразование, обеспечивающее, чтобы $t_0=0$. Это не влияет на решение или методы, используемые для приближения решения.

Зачастую имеется более чем одно зависимое переменное, и тогда задача заключается в решении системы уравнений первого порядка; например,

$$y' = f(y,\ z,\ t),$$
$$z' = g(y,\ z,\ t).$$

Предположим, что производные df/dy, df/dz, dg/dy, dg/dz существуют во всем интервале интегрирования. Решение этой системы содержит две постоянные интегрирования, и, следовательно, нужны два дополнительных условия, чтобы определить эти константы. Если значения y и z указаны при одном и том же значении независимой переменной t_0, то система будет иметь единственное решение. Задача определения значений y и z для (будущих) значений $t>t_0$ называется *задачей Коши*.

Любое обыкновенное дифференциальное уравнение порядка n, которое можно записать так, что его левая часть есть производная наивысшего порядка, а в правой части эта производная не встречается, может быть записано и в виде системы из n уравнений первого порядка путем введения $n-1$ новых переменных. Например,

$$u'' = g(u,\ u',\ t)$$

можно записать как систему

$$z' = g(u,\ z,\ t),$$
$$u' = z,$$

где $z'(t)=u''(t)$. В векторных обозначениях это выглядит так:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y},\ t),$$

где

$$\mathbf{y} = \begin{pmatrix} z \\ u \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$
$$\mathbf{f}(\mathbf{y},\ t) = \begin{pmatrix} g(y_2,\ y_1,\ t) \\ y_1 \end{pmatrix}.$$

При обсуждении методов для задачи Коши удобно представлять себе единственное уравнение

$$y' = f(y,\ t),$$
$$y(t_0) = y_0.$$

Однако методы с равным успехом применимы и к системам уравнений.

6.2. Численные решения

Лишь очень немногие дифференциальные уравнения могут быть решены точно, и потому обычно необходимо приближать решения численными методами. Это требует задания следующей дополнительной информации:

1. Указание ошибки, которую пользователь готов допустить в решении. Если потребовать бесконечной точности, то мы не сможем, за исключением немногих игрушечных задач, ничего сделать.
2. Указание того, что пользователь готов заплатить за такое решение. Это зачастую функция допускаемой ошибки.

Подход, который мы опишем, связан с *пошаговыми методами* (называемыми также *разностными методами* или *методами дискретного переменного*). Генерируется последовательность точек $t_0, t_1, t_2, \ldots$, возможно, с переменной длиной шага $h_n=t_{n+1}-t_n$. В каждой точке t_n решение $y(t_n)$ аппроксимируется числом y_n, которое вычисляется по предыдущим значениям. Разностный метод, дающий формулу для вычисления y_{n+1} по k предыдущим значениям $y_n, y_{n-1}, \ldots, y_{n-k+1}$, называется *k-шаговым методом*. Если $k=1$, то это *одношаговый метод*, а при $k>1$ это *многошаговый* метод.

Примером одношагового метода является метод *Эйлера*. В методе Эйлера значение y_{n+1} вычисляется посредством прямолинейной экстраполяции из предыдущей точки y_n. Рассмотрим уравнение

$$y'=f(y, t),$$

и пусть задано $y(t_0)=y_0$. Наклон решения $y(t)$ в начальной точке можно вычислить по формуле $y'_0=f(y_0, t_0)$. Тогда приближение y_1 к $y(t_1)$ можно найти, беря два первых члена ряда Тейлора:

$$y(t_1)\approx y_1=y_0+h_0 f(y_0, t_0).$$

Полагаем затем $t_2=t_1+h_1$ и находим

$$y(t_2)\approx y_2=y_1+h_1 f(y_1, t_1)$$

и т. д. В общем случае

$$y_{n+1}=y_n+h_n f(y_n, t_n).$$

Как видно из рис. 6.2, обычно на каждом новом шаге приближенное решение переходит на другой член семейства решений. Для некоторых дифференциальных уравнений это явление может привести к большим ошибкам. Например, при решении уравнения $y'=y$ методом Эйлера ошибки, сделанные на ранних этапах, умножаются с ростом времени на множитель e^t (рис. 6.1). Это явле-

ние называется *неустойчивостью дифференциального уравнения.* Иногда можно обойти эту трудность, решая задачу с обращенным временем. Однако при решении системы уравнений факт неустойчивости часто не зависит от направления решения.

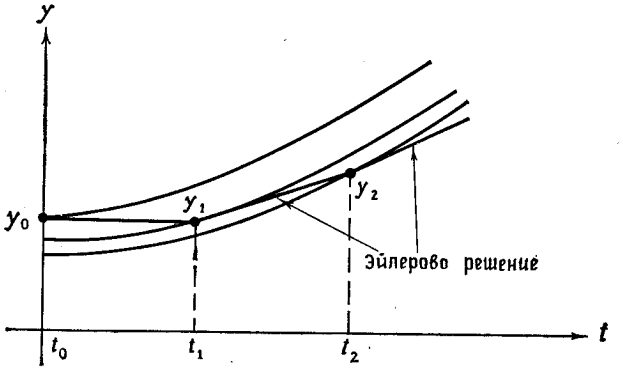


Рис. 6.2. Метод Эйлера.

Если, с другой стороны, мы имеем уравнение $y'=-y$, то получаем семейство кривых, изображенное на рис. 6.3. Здесь начальные ошибки убывают с ростом t. Это называется *устойчивостью дифференциального уравнения.* Вообще для дифференци-

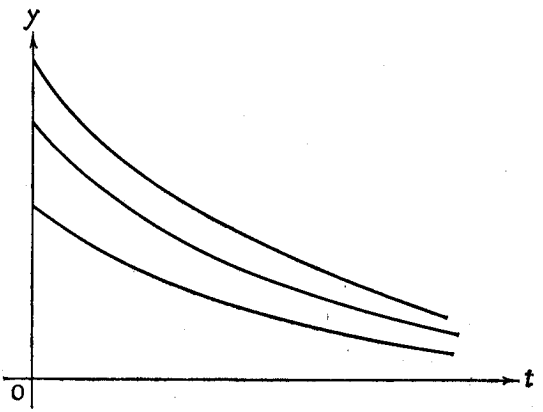


Рис. 6.3. Семейство решений уравнения $y'=-y$.

ального уравнения $y'=\lambda y$, где λ задано, начальные ошибки в решении умножаются при возрастающем t на $e^{\lambda t}$. Если $\lambda\leqslant 0$, то начальные ошибки не возрастают, так что уравнение устойчиво. Если $\lambda>0$, уравнение неустойчиво. Как мы увидим впоследствии,

устойчивое дифференциальное уравнение не обязательно легко решается численными методами. Если нужна малая *относительная ошибка*, то устойчивость дифференциального уравнения может не иметь значения.

6.3. Ошибки

Ошибки в численном решении задачи Коши проистекают из двух источников:

1. *Ошибка дискретизации* (иногда называемая ошибкой *усечения*).
2. *Ошибка округления.*

Ошибка дискретизации есть свойство используемого метода. Это значит, что если бы все арифметические вычисления могли выполняться с бесконечной точностью, то других ошибок, кроме ошибки дискретизации, не было бы. Ошибка округления есть свойство машины и программы. Поскольку точное решение, вообще говоря, неизвестно и не может быть вычислено, то обычно ошибка дискретизации так или иначе оценивается.

Важно различать между собой две меры ошибок дискретизации: *локальную* ошибку дискретизации и *глобальную* ошибку дискретизации. Локальная ошибка дискретизации — это ошибка, сделанная на данном шаге, при условии, что предыдущие значения точны и что нет ошибок округления. Более точно, пусть $u_n(t)$ — функция от t, описываемая условиями

$$u_n' = f(u_n, t),$$
$$u_n(t_n) = y_n.$$

Таким образом, $u_n(t)$ — решение дифференциального уравнения, определяемое не исходным начальным условием в точке t_0, а значением вычисленного решения в точке t_n. Локальная ошибка дискретизации d_n есть

$$d_n = y_{n+1} - u_n(t_{n+1}).$$

Это разность между вычисленным решением и теоретическим решением, определяемыми одними и теми же данными в точке t_n.

Глобальная ошибка дискретизации есть разность между вычисленным решением (ошибки округления игнорируются) и точным решением, определяемым исходным условием в точке t_0, т. е.

$$e_n = y_n - y(t_n).$$

Различие между локальной и глобальной ошибками дискретизации легко видеть в специальном случае, когда $f(y, t)$ не зависит от y. В этом случае решение есть просто интеграл $y(t)=\int_{t_0}^{t} f(\tau)\,d\tau$. Метод Эйлера превращается в схему численного интегрирования, которую можно было бы назвать «формулой прямоугольников для лентяя»; эта схема использует значения функции в концах подынтервалов, а не в их средних точках:

$$\int_{t_0}^{t_N} f(\tau)\,d\tau \approx \sum_{n=0}^{N-1} h_n f(t_n).$$

Локальная ошибка дискретизации есть ошибка на одном подынтервале

$$d_n = h_n f(t_n) - \int_{t_n}^{t_{n+1}} f(\tau)\,d\tau,$$

а глобальная ошибка дискретизации — общая ошибка

$$e_N = \sum_{n=0}^{N-1} h_n f(t_n) - \int_{t_0}^{t_N} f(\tau)\,d\tau.$$

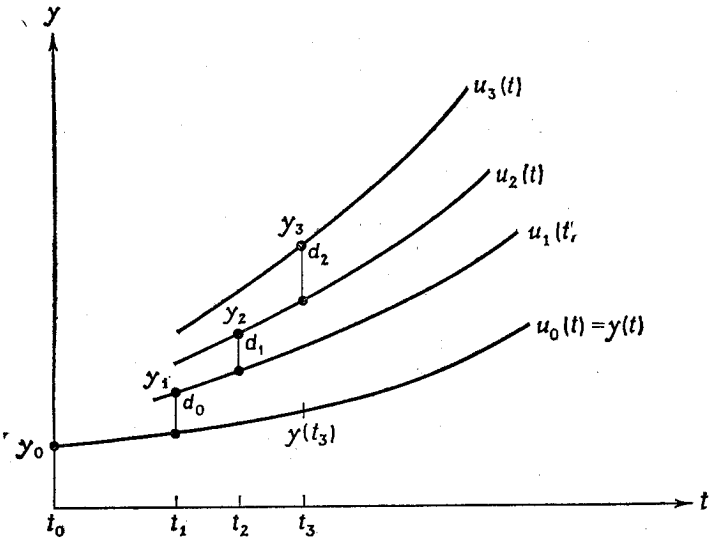
В данном специальном случае каждый подынтервал независим от других (сумма может вычисляться в любом порядке), так что глобальная ошибка есть сумма локальных ошибок:

$$e_N = \sum_{n=0}^{N-1} d_n.$$

В случае настоящего дифференциального уравнения, где $f(y, t)$ зависит от y, ошибка на любом интервале зависит от решений, вычисленных для предыдущих интервалов. Вследствие этого глобальная ошибка в общем случае будет больше суммы локальных ошибок, если дифференциальное уравнение неустойчиво, но меньше этой суммы, если дифференциальное уравнение устойчиво. Внимательное изучение рис. 6.4 и 6.5 должно прояснить эти утверждения.

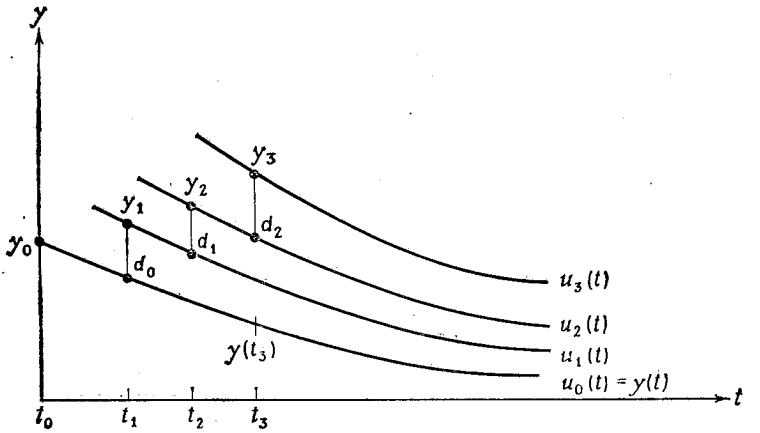
Устойчивость дифференциального уравнения и, следовательно, влияние локальной ошибки на глобальную управляются знаком производной $\partial f/\partial y$. Для нелинейных уравнений $\partial f/\partial y$ может изменять знак, так что уравнение может быть неустойчиво в одних областях и устойчиво в других. Для системы нелинейных уравнений ситуация еще более сложна.

Фундаментальным понятием при оценке точности численного метода является его *порядок*. Порядок определяется в терминах



$$e_3 = y_3 - y(t_3) > d_0 + d_1 + d_2$$

Рис. 6.4. Локальная и глобальная ошибки дискретизации для неустойчивого дифференциального уравнения.



$$e_3 = y_3 - y(t_3) < d_0 + d_1 + d_2$$

Рис. 6.5. Локальная и глобальная ошибки дискретизации для устойчивого дифференциального уравнения.

локальной ошибки дискретизации, получаемой при применении метода к задачам с гладкими решениями. Говорят, что метод имеет порядок p, если существует число C, такое, что

$$|d_n| \leqslant Ch_n^{p+1}.$$

Число C может зависеть от производных функции, определяющей дифференциальное уравнение, и от длины интервала, на котором ищется решение, но оно не должно зависеть от номера шага n и величины шага h_n. Указанное неравенство с помощью O-символики может быть записано более компактно:

$$d_n = O(h_n^{p+1}).$$

Например, рассмотрим метод Эйлера

$$y_{n+1} = y_n + h_n f(y_n, t_n).$$

Предположим, что локальное решение $u_n(t)$ имеет непрерывную вторую производную. Тогда, согласно разложению Тейлора, в окрестности точки t_n

$$u_n(t) = u_n(t_n) + (t - t_n) u_n'(t_n) + O((t - t_n)^2).$$

Из дифференциального уравнения и начального условия, определяющего $u_n(t)$, следует

$$u_n(t_{n+1}) = y_n + h_n f(y_n, t_n) + O(h_n^2).$$

Поэтому

$$d_n = y_{n+1} - u_n(t_{n+1}) = O(h_n^2).$$

Отсюда заключаем, что для метода Эйлера $p=1$, т. е. метод имеет первый порядок.

Рассмотрим теперь глобальную ошибку дискретизации в фиксированной конечной точке $t=t_f$. По мере повышения требований к точности длины шагов h_n будут убывать, а общее их число N, необходимое для достижения t_f, будет возрастать. Грубо говоря,

$$N = \frac{t_f - t_0}{h},$$

где h — средняя длина шага. Далее, глобальная ошибка e_N может быть представлена как сумма N локальных ошибок с множителями, описывающими устойчивость уравнения. Эти множители лишь слабо зависят от величин шагов, и потому мы можем, огрубляя, сказать, что если локальная ошибка есть $O(h^{p+1})$, то глобальная ошибка будет $N \cdot O(h^{p+1}) = O(h^p)$. Вот почему в определении порядка в показателе было взято $p+1$, а не p.

Для метода Эйлера $p=1$, так что уменьшение средней длины шага в 2 раза уменьшит среднюю локальную ошибку примерно в $2^{p+1}=4$ раза; но так как для достижения t_f теперь потребуется

приблизительно вдвое больше шагов, то глобальная ошибка уменьшится лишь в $2^p=2$ раза. Для методов более высокого порядка глобальная ошибка для гладких решений уменьшилась бы гораздо ощутимей.

Нужно отметить, что при обсуждении численных методов для обыкновенных дифференциальных уравнений слово *порядок* может иметь несколько различных значений. Порядок дифференциального уравнения — это индекс наивысшей встречающейся в нем производной. Например, $y''=t+\sin y$ — дифференциальное уравнение второго порядка. Термин «порядок системы уравнений» иногда относится к числу уравнений системы. Например, $y'=y+z$, $z'=y-z$ есть система второго порядка. Порядок численного метода для решения обыкновенного дифференциального уравнения — это как раз то, что мы здесь обсуждаем. Это степень длины шага, которая появляется в выражении для глобальной ошибки. Например, метод Эйлера имеет первый порядок.

На практике проведенный анализ должен быть дополнен, чтобы учесть ошибки округлений. Если на каждом шаге метода Эйлера делается наихудшая возможная ошибка округления ε, то (предполагая, для простоты, фиксированной длину шага)

$$y_{n+1}=y_n+hf(y_n, t_n)+\varepsilon$$

и полная ошибка вследствие округлений равна $N\varepsilon$ или $b\varepsilon/h$, где $b=t_f-t_0$. Можно надеяться, однако, что вместо того, чтобы быть константой, ε является в известной степени случайной величиной со средним значением 0. На практике общая ошибка, накопившаяся от округлений, обычно ведет себя как $\varepsilon\sqrt{N}$, если машина округляет. На машинах типа IBM 360, которые после арифметических операций просто отбрасывают лишние разряды (это часто называется усечением [1]), и вот почему мы предпочли выше термин *ошибка дискретизации*), ошибка округлений имеет тенденцию расти линейно по N.

Суммируя глобальную ошибку дискретизации и ошибку округлений, получаем

$$\text{общая ошибка} \approx b\left(Ch+\frac{\varepsilon}{h}\right).$$

Следовательно, общая ошибка минимизируется для некоторого оптимального значения h, выражаемого приближенной формулой

$$h_{\text{opt}} \approx \sqrt{\frac{\varepsilon}{c}}.$$

Общая ошибка на конце интервала интегрирования изображается на рис. 6.6 как функция от длины шага h. По мере убывания

[1]) В оригинале — truncation.— *Прим. перев.*

h ошибка дискретизации убывает (линейно) и численное решение начинает сходиться к точному решению. Однако, когда h становится слишком мало, численное решение начинает расходиться вследствие ошибок округлений. Оптимальное значение h

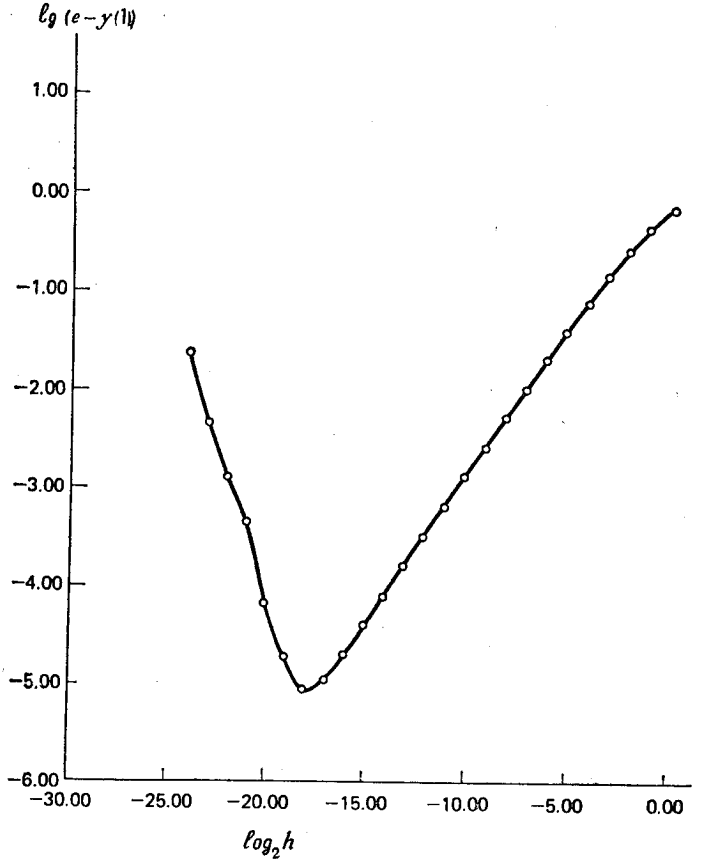


Рис. 6.6. Общая ошибка в $y(1)$ при использовании метода Эйлера для решения задачи $y'=y$, $y(0)=1$. Машина Honeywell 6050, обычная точность ($\beta=2$, $t=27$).

для этой задачи, очевидно, равно приблизительно 2^{-18}. При решении обыкновенных дифференциальных уравнений обычно нужно значительное машинное время, чтобы ошибки округлений могли ощутимо накопиться. Например, вычисления для рис. 6.6 потребовали около 15 минут времени процессора машины Honeywell 6050. Однако, довольно часто приходится интегрировать системы уравнений, для которых необходимы часы машинного времени.

6.4. Методы

Хотя для численного решения обыкновенных дифференциальных уравнений существует множество различных пошаговых методов, каждый попадает в одну из четырех общих категорий.

Методы рядов Тейлора

Если $y(t)$ — гладкое решение, то оно имеет разложение по Тейлору

$$y(t+h)=y(t)+hy'(t)+\frac{h^2}{2!}y''(t)+\dots .$$

Метод Эйлера можно рассматривать как приближение, использующее первые два члена ряда Тейлора. Если могут быть вычислены производные функции y более высокого порядка, то можно получить метод порядка p, полагая

$$y_{n+1}=y_n+hy'_n+\frac{h^2}{2!}y''_n+\frac{h^3}{3!}y'''_n+\dots+\frac{h^p}{p!}y_n^{(p)}.$$

Первый отброшенный член дает оценку локальной ошибки дискретизации и потому может быть использован для выбора величины шага. Производные $y'(x)$, $y''(x)$ и т. д. можно выразить через частные производные функции f. При этом важно различать функцию $f(y, t)$ двух независимых переменных и функцию $f(y(t), t)$ одного независимого переменного, полученную подстановкой решения y в f. Исходя из основного дифференциального уравнения

$$y'(t)=f(y(t), t),$$

можно продифференцировать обе части по t, что дает

$$y''=f_y y'+f_t=f_y\cdot f+f_t;$$

затем

$$y'''=f_{yy}\cdot f^2+f_y^2\cdot f+f_y\cdot f_t+2f_{ty}\cdot f+f_{tt}$$

и т. д. Этот процесс называется полным дифференцированием.

Например, если $f(y, t)=y^2+t^2$, то

$$\begin{aligned} y' &= y^2+t^2,\\ y'' &= 2y^3+2yt^2+2t,\\ y''' &= 6y^4+8y^2t^2+4yt+2t^4+2. \end{aligned}$$

Заметьте, что даже для многочлена f от двух переменных выражения полных производных усложняются по мере роста порядка.

Можно составить программу, которая вычисляет полные производные для некоторых классов функций. Для дифференциальных уравнений, определяемых такими функциями, методы рядов

Тейлора могут быть очень эффективны. Однако в общем случае применение этих методов слишком ограничено, чтобы представлять для нас дальнейший интерес.

Методы Рунге — Кутта

Метод Рунге—Кутта предназначен для аппроксимации методов, основанных на рядах Тейлора, но без явного вычисления производных, за исключением первой. Приближение получается ценой нескольких вычислений функции. Классический метод Рунге — Кутта четвертого порядка выражается формулой

$$y_{n+1} = y_n + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3),$$

где

$$\begin{aligned} k_0 &= hf(y_n,\ t_n),\\ k_1 &= hf\left(y_n + \frac{1}{2}k_0,\ t_n + \frac{h}{2}\right),\\ k_2 &= hf\left(y_n + \frac{1}{2}k_1,\ t_n + \frac{h}{2}\right),\\ k_3 &= hf(y_n + k_2,\ t_n + h). \end{aligned}$$

Заметьте, что на каждом шаге требуется четыре вычисления функции $f(y,\ t)$.

Классический метод Рунге — Кутта может рассматриваться как обобщение на дифференциальные уравнения квадратурной формулы Симпсона:

$$\int_{t_n}^{t_{n+1}} f(t)\,dt \approx \frac{h_n}{6}\left[f(t_n) + 4f\left(\frac{t_n + t_{n+1}}{2}\right) + f(t_{n+1})\right].$$

Если $f(y,\ t)$ есть функция только от t, то обе формулы совпадают.

При $h \to 0$ классический метод Рунге — Кутта четвертого порядка согласуется с рядом Тейлора асимптотически до членов порядка h^4. Однако здесь нет легко определяемой оценки для локальной ошибки дискретизации, помогающей в выборе величины шага. В § 6.8 будет подробно описана современная модификация метода Рунге — Кутта, которая включает контроль длины шага. Методы Рунге — Кутта имеют несколько достоинств. Их легко программировать; они численно устойчивы для широкого класса задач. Поскольку для вычисления y_{n+1} нужно лишь одно значение решения y_n, то метод является *самостартующим*. Величину шага h_n можно изменить на любом этапе вычислений.

Многошаговые методы

В методах, рассматривавшихся до сих пор, значение y_{n+1} вычисляется посредством функции, которая зависит лишь от t_n, y_n и длины шага h_n. Видимо, логично предположить, что можно получить бóльшую точность, используя информацию в предыдущих точках, именно y_{n-1}, y_{n-2}, ... и f_{n-1}, f_{n-2}, ..., где $f_i = f(y_i, t_i)$. Многошаговые методы, основанные на этой идее, весьма эффективны. Если требуется высокая точность, то они обычно более экономичны, чем одношаговые методы, и часто можно тривиально получить оценку ошибки усечения. Запрограммированные соответствующим образом, многошаговые методы могут эффективно выдавать значения численного решения в произвольных точках, не изменяя значение h. Порядок метода может выбираться автоматически и динамически изменяться, тем самым получаются методы, работающие для очень широкого круга задач. В процессе решения могут выявляться некоторые виды ошибок. Жесткие уравнения (обсуждаемые ниже) могут решаться некоторыми многошаговыми методами, и уравнения можно автоматически классифицировать как жесткие или нежесткие. Эти достоинства приобретаются ценой усложнения программы и в некоторых случаях за счет возможной численной неустойчивости.

Линейные многошаговые методы можно рассматривать как специальные случаи формулы

$$y_{n+1} = \sum_{i=1}^{k} \alpha_i y_{n+1-i} + h \sum_{i=0}^{k} \beta_i f_{n+1-i},$$

где k — фиксированное целое число и α_k либо β_k отлично от нуля. Эта формула определяет *общий линейный k-шаговый метод*. Метод называется линейным, потому что каждое f_i входит в формулу линейно; при этом f может быть, а может не быть линейной функцией своих аргументов.

После того как метод «стартовал», каждый шаг требует вычисления y_{n+1} из известных значений: y_n, y_{n-1}, ..., y_{n-k+1}, f_n, f_{n-1}, ..., f_{n-k+1}. Если $\beta_0 = 0$, метод называется *явным* и вычисление проводится очевидным образом. Если же $\beta_0 \neq 0$, то метод называется *неявным*, потому что для нахождения y_{n+1} требуется f_{n+1}. Бóльшие трудности при использовании неявных методов окупаются другими их качествами.

Обычно на каждом шаге решения совместно используются два многошаговых метода. Явный метод, называемый *предиктором*, сопровождается одним или более применениями неявного метода, называемого *корректором* — отсюда название «методы *предиктор-корректор*.»

Примером хорошего метода предиктор-корректор является метод Адамса четвертого порядка:

$$\text{предиктор: } y_{n+1}=y_n+\frac{h}{24}(55f_n-59f_{n-1}+37f_{n-2}-9f_{n-3}),$$

$$\text{корректор: } y_{n+1}=y_n+\frac{h}{24}(9f_{n+1}+19f_n-5f_{n-1}+f_{n-2}).$$

Обе формулы имеют четвертый порядок. Это частные примеры семейства предикторов и корректоров с различными порядками. Схема предиктор-корректор для вычисления y_{n+1} такова:

1. Использовать предиктор для вычисления $y_{n+1}^{(0)}$ — начального приближения к y_{n+1}. Положить $i=0$.
2. Вычислить функцию и положить $f_{n+1}^{(i)}=f(y_{n+1}^{(i)},\ t_{n+1})$.
3. Вычислить лучшее приближение $y_{n+1}^{(i+1)}$ по формуле корректора, полагая $f_{n+1}=f_{n+1}^{(i)}$.
4. Если $|y_{n+1}^{(i+1)}-y_{n+1}^{(i)}|>$ заданного допуска, то увеличить i на 1 и перейти к шагу 2; в противном случае положить $y_{n+1}=y_{n+1}^{(i+1)}$.

Здесь три различных процесса: шаг-предсказание 1, который мы обозначим через P; шаг-вычисление 2, назовем его E; и шаг-коррекция 3, обозначаемый C. Шаг 4 может быть заменен требованием выполнения ровно m итераций корректора. Получающийся метод обозначают $P(EC)^m$. Имеются веские причины полагать, что разумно заключительное вычисление f, дающее более точное значение для следующего шага; подобные методы обозначают $P(EC)^mE$. Если порядок предиктора не меньше, чем порядок корректора, или меньше только на единицу, то более чем двукратная коррекция уже существенно не влияет на результат. Для методов Адамса известно, что если принимать во внимание и устойчивость, и ошибки дискретизации, то метод $P(EC)^m$ уступает методу $P(EC)^{m-1}E$, а метод $P(EC)^1E$ более устойчив, чем $P(EC)^2E$. Таким образом, наиболее разумной схемой предиктор-корректор для методов Адамса является, видимо, PECE.

Методы экстраполяции

Формулы предсказания, обсуждавшиеся в предыдущем разделе, можно рассматривать как методы экстраполяции, где y_{n+1} экстраполируется по известным предыдущим значениям y_i и f_i. Метод, использующий другой тип экстраполяции, предложил в 1927 г. Гонт (J. A. Gaunt); его развил в докторской диссертации 1964 г. Грэг (W. B. Gragg). Мы вкратце опишем этот численный метод. По поводу некоторых тонких деталей отсылаем читателя к известной статье Булирш, Стоэр (1966).

Дифференциальное уравнение интегрируется на заданном интервале, например $0 \leqslant t \leqslant T$, посредством одношагового метода для различных длин шага h_j; полученные результаты обозначим через $Y(h_j)$. Это дает дискретное приближение к функции $Y(h)$, где $Y(0)=y(T)$. По вычисленным узлам строится интерполяционный полином или рациональная функция $\hat{Y}(h)$. Тогда экстраполированное значение $\hat{Y}(0)$ является обычно очень хорошим приближением к $Y(0)$. Это еще один пример упомянутого в гл. 5 приема Ричардсона «экстраполяции к пределу».

6.5. Жесткие уравнения

Постоянная времени для решения дифференциального уравнения — это время, необходимое для его уменьшения, выражаемого коэффициентом $1/e$. Например, уравнение $y'=-\lambda y$ имеет решение $Ce^{-\lambda t}$. Если λ положительно, то y уменьшится в $1/e$ раз за время $1/\lambda$. Устойчивое дифференциальное уравнение называется *жестким*, если оно имеет частное решение в виде убывающей экспоненты, постоянная времени которого очень мала в сравнении с длиной интервала, на котором разыскивается решение. Для системы

$$\mathbf{y}' = \mathbf{f}(\mathbf{y})$$

скорости убывания связаны, по крайней мере локально, с собственными значениями матрицы $\partial f/\partial y = (\partial f_i/\partial y_j)$. Рассмотрим, например, систему

$$u' = 998u + 1998v,$$
$$v' = -999u - 1999v.$$

Собственные значения матрицы коэффициентов

$$\begin{pmatrix} 998 & 1998 \\ -999 & -1999 \end{pmatrix}$$

суть -1 и -1000. Если $u(0)=v(0)=1$, то решением будет

$$u = 4e^{-t} - 3e^{-1000t},$$
$$v = -2e^{-t} + 3e^{-1000t};$$

оно показано на рис. 6.7. После очень небольшого промежутка времени решение весьма близко к вектор-функции

$$u = 4e^{-t},$$
$$v = -2e^{-t}.$$

Предположим теперь, что мы должны решать эту систему посредством метода Эйлера. Дискретное решение можно записать

формулами

$$u_{n+1} = u_n + h(998u_n + 1998v_n),$$
$$v_{n+1} = v_n + h(-999u_n - 1999v_n),$$

где $u_0 = v_0 = 1$. Если выбрать $h = 0.01$, то

$$u_1 = 1 + 0.01(998 + 1998) = 30.96,$$
$$v_1 = 1 + 0.01(-999 - 1999) = -28.98.$$

Попробуйте выполнить еще несколько шагов интегрирования; вы увидите, какой катастрофический характер примут результаты. Положение дел может быть поправлено, по крайней мере временно, использованием меньшего значения h. Однако посте-

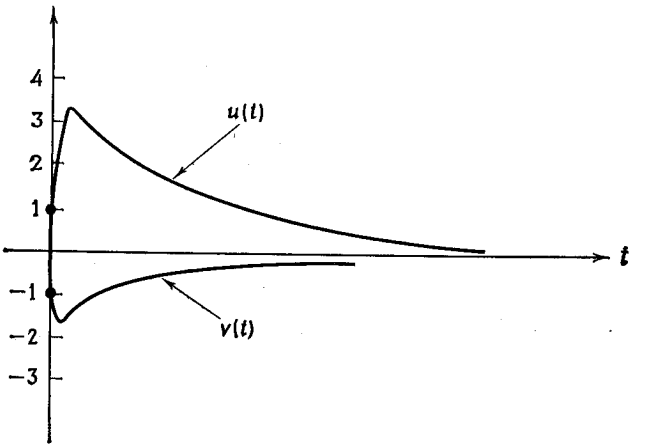


Рис. 6.7. Решение жесткого уравнения.

пенно ошибки округления и дискретизации накопятся в достаточной степени для того, чтобы снова вызвать неустойчивость. Это явление можно представить себе визуально, рассматривая семейство решений, ассоциированных с $u(t)$, показанное на рис. 6.8. Переходная часть решения, которая, казалось бы, давно уже практически исчезла, тем не менее мешает увеличить длину шага и ограничивает область значений независимого переменного, для которой можно вычислить решение. Это особенно досадно потому, что на данном этапе вычислений решение очень гладко, и хотелось бы увеличить длину шага.

Большинство стандартных методов не приспособлено для решения жестких уравнений. Очень часто, прежде чем программа квалифицирует уравнение как жесткое, происходит многократное автоматическое уменьшение величины h, стоящее больших денег. К счастью, были изобретены специальные методы,

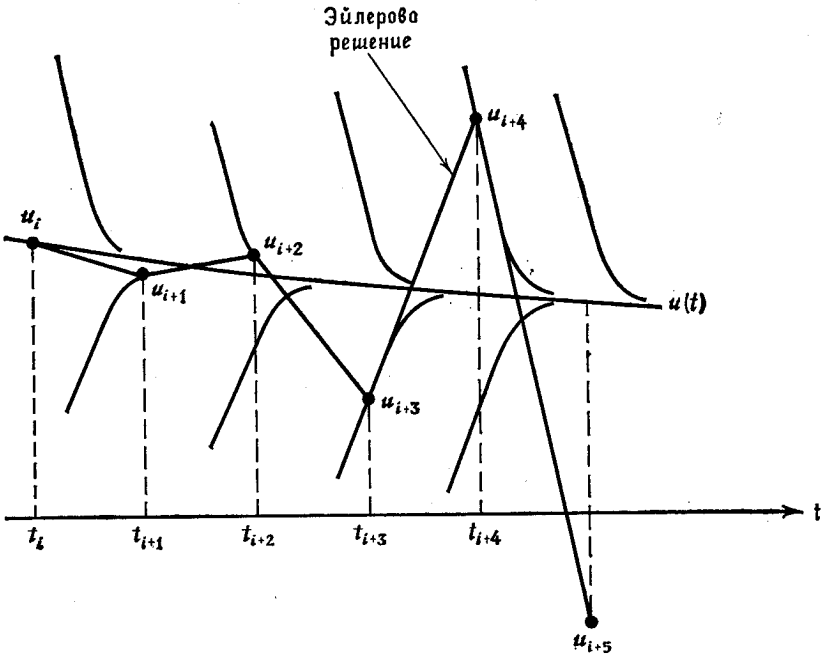


Рис. 6.8. Решение жесткого уравнения методом Эйлера.

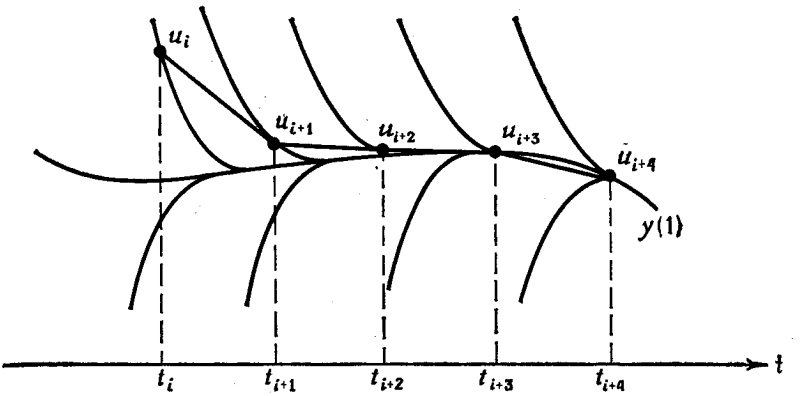


Рис. 6.9. Решение жесткого уравнения обратным методом Эйлера.

оказавшиеся весьма эффективными. Вероятно простейшим из них является так называемый *обратный метод Эйлера*, выражаемый неявной формулой:

$$y_{n+1}=y_n+hf(y_{n+1},\ t_{n+1}).$$

Работа обратного метода Эйлера иллюстрируется рис. 6.9.

Конструирование эффективных методов для жестких уравнений является областью активных исследований. Обсуждение имеющихся в настоящее время методов выходит за рамки этой книги; кроме того, всякое такое обсуждение очень быстро устарело бы. Мы отсылаем читателя к книге Гир (1968) и текущей литературе по численному анализу и математическому обеспечению.

6.6. Краевые задачи

До сих пор мы обсуждали лишь задачи с начальными условиями, в которых значения зависимых переменных задаются для одного и того же значения независимой переменной t. В некоторых случаях приходится решать систему обыкновенных дифференциальных уравнений при наличии условий на значения зависимых переменных, задаваемых для различных значений t. Подобная задача называется *краевой задачей*. Методы решения краевых задач, вообще говоря, очень отличаются от методов решения задач Коши; см. Келлер (1968). Однако мы кратко осветим популярный метод для решения краевой задачи, который сводит ее к последовательности задач Коши.

Предположим, что нужно решить систему

$$y'=f(y,\ t),$$

где

$$y=\begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$

при наличии условий $y_1(0)=\alpha$ и $y_2(b)=\beta$, $b\neq 0$. Можно применить следующий итерационный метод:

1. Выбрать значение ξ, аппроксимирующее $y_2(0)$.
2. Решить задачу Коши:

$$\hat{y}'=f(\hat{y},t),\quad \hat{y}(0)=\begin{pmatrix} \alpha \\ \xi \end{pmatrix}.$$

3. Если $|\hat{y}_2(b)-\beta|$ мало, положить $y(t)\approx\hat{y}(t)$; в противном случае изменить ξ тем или иным образом (см. гл. 7) и вернуться к шагу 2.

Этот метод решения краевых задач называется *методом стрельбы* — вполне подходящее название!

6.7. Выбор подпрограммы

Трудно сконструировать такую программу, которая бы эффективно решала любую задачу Коши для обыкновенных дифференциальных уравнений. Помимо прочего, нужно принимать во внимание специальные свойства конкретных задач, жесткость, сложность и стоимость вычисления правых частей, ожидаемую точность, желаемый вид выходной информации, легкость, с которой пользователь может читать, использовать и, если необходимо, модифицировать программу.

Из многих имеющихся подпрограмм мы решили включить в книгу фортран-подпрограмму RKF45, авторы Уотс и Шампэнь. Это был нелегкий выбор, и в действительности два предварительных варианта книги содержали две другие подпрограммы. В этом параграфе будет объяснен данный выбор, а также то, почему в других ситуациях можно предпочесть иное решение.

Несколько недавних исследований сравнивают различные методы и программы для решения задач Коши. Читатель, желающий больше информации относительно выбора подпрограммы, должен обратиться к статьям: Хал и др. (1972), Крог (1973), Шампэнь и др. (1976), Энрайт, Хал (1976).

Важно различать метод и подпрограмму. Метод обычно может быть описан несколькими очень простыми формулами. В самом деле, в предыдущем параграфе было описано несколько методов. Программа, полностью решающая задачу, является, с другой стороны, гораздо более сложным объектом. Она должна предусматривать множество деталей, таких, как выбор величины шага, контроль ошибки, управление временной памятью, взаимодействие с другими программами, выявление ошибок в вызывающей последовательности и возможных разрывов и особенностей. Две различные программы, реализующие один и тот же основной метод, могут поэтому существенно различаться своими характеристиками.

Пользователь, возможно, знает точность, требуемую в решении, но он может не иметь представления о величине шага, необходимой, чтобы достигнуть ее. Более того, требования к величине шага обычно меняются в процессе решения. Поэтому мы считаем существенным, чтобы подпрограмма общего назначения включала автоматический выбор длины шага и контроль ошибки дискретизации.

Мы рассматриваем главным образом нежесткие задачи. Качественная подпрограмма с автоматическим контролем ошибки обычно работает и для жестких задач, но она может быть очень неэффективной в этом случае. Некоторые недавно разработанные подпрограммы пытаются выявить чрезмерную жесткость и пре-

дупредить пользователя, что другой подход может оказаться менее дорогим.

Методы рядов Тейлора приложимы лишь к специальным задачам, в которых $f(y, t)$ может быть задана в символической, легко дифференцируемой форме. Поскольку нам нужны методы, в которых от пользователя требуется лишь подготовить подпрограмму вычисления $f(y, t)$, то будем рассматривать только методы Рунге — Кутта, многошаговые и экстраполяционные.

Конкретный метод Рунге — Кутта обязательно имеет фиксированный порядок. Например, классический метод имеет порядок 4. Для различных порядков нужны методы с различными коэффициентами. Кроме того, одинокий метод не имеет средств для поддержания на заданном уровне ошибки дискретизации и, следовательно, не может выбирать величину шага. Для того чтобы достигнуть автоматического контроля длины шага, необходимо комбинировать два различных метода. Методы Рунге — Кутта, которые эффективно добиваются этого, были построены лишь недавно. Эти комбинированные методы имеют все же фиксированный порядок и потому эффективны только в ограниченном диапазоне требований к точности.

Хорошо написанную подпрограмму, основанную на методах Рунге — Кутта, можно довольно легко читать, понимать, использовать и модифицировать. Это главные причины, по которым мы выбрали подобную подпрограмму для данной книги. Подпрограмма RKF45, описываемая в следующем параграфе, является наилучшей универсальной реализацией методов Рунге — Кутта, которая нам известна.

Методы Адамса — это семейство многошаговых методов различных порядков. Хорошо составленная подпрограмма Адамса с переменным порядком и переменным шагом обычно более эффективна в широком диапазоне требований к точности, чем подпрограмма Рунге — Кутта фиксированного порядка с переменным шагом. Среди примеров таких подпрограмм отметим DVDQ и DVOA (Fred Krogh), DIFSUB (Гир (1971)), STEP и DE (Шампэнь, Гордон (1975)), VOAS (Седгвик (1973)). Из-за переменного порядка все эти подпрограммы более сложны. Поэтому они требуют большей памяти и более трудны для чтения и понимания, чем RKF45. Кроме того, при скромных запросах к точности (локальная ошибка меньше чем 10^{-5} или 10^{-6}) RKF45 столь же эффективна, как и более сложные методы Адамса.

Имеющиеся в настоящее время подпрограммы, использующие экстраполяционные методы, несколько менее гибки, чем подпрограммы, основанные на других методах. К тому же они по-видимому менее эффективны, особенно при высоких требованиях к точности или если выходные результаты нужны по всему интервалу интегрирования, а не только в его конце. Работа по реали-

зации экстраполяционных методов, однако, продолжается, и потому они должны учитываться при любом сравнении.

6.8. Подпрограмма RKF45

RKF45 — это подпрограмма решения задач Коши для обыкновенных дифференциальных уравнений, основанная на формулах Рунге — Кутта, которые предложил в 1970 г. Фельберг; она была составлена в 1974 г. Шампэнем и Уотсом. Подпрограмма требует шести вычислений функции за шаг. Четыре из этих значений берутся с одним набором коэффициентов, приводя к методу четвертого порядка, и все шесть значений комбинируются с другими коэффициентами, что дает метод пятого порядка. Сравнение двух вычисленных величин позволяет получить оценку ошибки, которая используется для контроля длины шага.

Мы не станем подробно выводить полные формулы, а рассмотрим вместо этого более простой метод второго порядка. Этот метод использует лишь два вычисления функции за шаг. Первое из них

$$k_1 = h_n f(y_n,\ t_n).$$

Затем предпринимается дробный шаг, основанный на k_1. Введем два пока неопределенных коэффициента α и β. Пусть

$$k_2 = h_n f(y_n + \beta k_1,\ t_n + \alpha h_n).$$

Наконец, для полного шага берется комбинация двух значений функции с еще двумя неопределенными коэффициентами:

$$y_{n+1} = y_n + \gamma_1 k_1 + \gamma_2 k_2.$$

Чтобы определить коэффициенты, разложим k_1 и k_2 в ряды Тейлора (с двумя переменными) относительно точки $(t_n,\ y_n)$, что дает

$$\begin{aligned} k_1 &= h_n f_n, \\ k_2 &= h_n (f_n + \beta k_1 f_{y,\,n} + \alpha h_n f_{t,\,n} + \ldots), \\ y_{n+1} &= y_n + (\gamma_1 + \gamma_2)\, h_n f_n + \gamma_2 \beta h_n^2 f_{y,\,n} f_n + \gamma_2 \alpha h_n^2 f_{t,\,n} + \ldots . \end{aligned}$$

Разложение y_{n+1} можно теперь сравнить с разложением точного *локального* решения $u_n(t)$, определенного в § 6.3:

$$\begin{aligned} u_n(t_{n+1}) &= u_n(t_n) + h_n u_n'(t_n) + \frac{h_n^2}{2} u_n''(t_n) + \ldots \\ &= y_n + h_n f_n + \frac{h_n^2}{2} (f_{y,\,n} f_n + f_{t,\,n}) + \ldots . \end{aligned}$$

Приравнивая коэффициенты при одинаковых степенях h_n, получаем три уравнения относительно четырех неизвестных коэффи-

циентов:

$$\gamma_1+\gamma_2=1,$$
$$\gamma_2\beta=\frac{1}{2},$$
$$\gamma_2\alpha=\frac{1}{2}.$$

Выбрав один из этих коэффициентов, например α, за параметр, приходим к однопараметрическому семейству методов Рунге—Кутта:

$$k_1=h_nf(y_n,\ t_n),$$
$$k_2=h_nf(y_n+\alpha k_1,\ t_n+\alpha h_n),$$
$$y_{n+1}=y_n+\left(1-\frac{1}{2\alpha}\right)k_1+\frac{1}{2\alpha}k_2.$$

Два очевидных выбора для α — это $\frac{1}{2}$ и 1. Они определяют методы, тесно связанные с квадратурными формулами соответственно прямоугольников и трапеций. Исследование первых членов, отброшенных в вышеуказанных разложениях, показывает, что ни при каком выборе α не удается исключить эти члены для всех функций $f(y,\ t)$; таким образом, метод имеет второй порядок при любом α.

Формулы, используемые в подпрограмме RKF45, предполагают шесть вычислений функции за шаг: $k_1,\ k_2,\ldots,\ k_6$. Они определяются соотношениями

$$k_i=h_nf\left(y_n+\sum_{j=1}^{i-1}\beta_{ij}k_i,\ t_n+\alpha_ih_n\right),\quad i=1,\ \ldots,\ 6.$$

Новое значение y_{n+1} вычисляется затем как взвешенная комбинация шести величин k_i:

$$y_{n+1}=y_n+\sum_{i=1}^{6}\gamma_ik_i.$$

Всего здесь 27 коэффициентов: 6 коэффициентов α_i, 15 — β_{ij} и 6—γ_i. Заметьте, что коэффициенты β_{ij} образуют нижний треугольный массив, так что каждое k_i получается по предыдущим значениям k_j.

Чтобы определить коэффициенты, все k_i раскладываются в ряды Тейлора, эти разложения подставляются в формулу для y_{n+1} и результат сравнивается с рядом Тейлора для $u_n(t_{n+1})$. Можно найти такие значения коэффициентов, что разность между двумя разложениями начинается с члена порядка h_n^6, а не h_n^5. Получающийся метод имеет, следовательно, пятый порядок. Хотя это и может показаться удивительным, но в то время как для $p=1,\ 2,\ 3$ и 4 можно построить метод порядка p за p вычисле-

Таблица 6.1

α_i	β_{ij}					γ_i	γ_i^*
0						$\frac{16}{135}$	$\frac{25}{216}$
$\frac{1}{4}$	$\frac{1}{4}$					0	0
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$				$\frac{6656}{12825}$	$\frac{1408}{2565}$
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$			$\frac{28561}{56430}$	$\frac{2197}{4104}$
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$		$-\frac{9}{50}$	$-\frac{1}{5}$
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	$\frac{2}{55}$	0

ний функции, для $p=5$ или 6 p вычислений функции дают лишь метод порядка $p-1$. (Это отчасти объясняет популярность классического метода четвертого порядка, упомянутого в предыдущем параграфе; чтобы получить еще один дополнительный порядок точности, требуются два дополнительных вычисления функции.)

Возможны различные способы выбора коэффициентов, приводящие к методам пятого порядка. Значения, предложенные Фельбергом, указаны в табл. 6.1.

«Лишнее» вычисление функции, однако, также не пропадает. Фельберг нашел еще один набор коэффициентов γ_i^*, такой, что четыре из шести коэффициентов k_i можно скомбинировать и получить второе значение y_{n+1}^*, точное до четвертого порядка. Таким образом,

$$y_{n+1}^* = y_n + \sum_{i=1}^{6} \gamma_i^* k_i.$$

Коэффициенты γ_i^* также приведены в таблице. Программа в действительности не вычисляет y_{n+1}^*, а находит вместо этого оценку ошибки $\sum_{i=1}^{6}(\gamma_i - \gamma_i^*)k_i$, используемую для контроля величины шага.

Иллюстрирующая основная программа в конце этого параграфа с помощью подпрограммы RKF45 определяет движение двух тел под действием взаимного гравитационного притяжения. Пусть $x(t)$ и $y(t)$ обозначают положение одного тела в координатной системе, начало которой зафиксировано в другом теле. Из ньютоновых законов движения вытекают следующие дифференци-

альные уравнения:

$$x''(t) = \frac{-\alpha^2 x(t)}{R(t)},$$

$$y''(t) = \frac{-\alpha^2 y(t)}{R(t)},$$

где

$$R(t) = [x(t)^2 + y(t)^2]^{3/2},$$

а α — константа, зависящая от гравитационной постоянной, масс обоих тел и выбранных единиц измерения.

Если взять начальные условия

$$x(0) = 1-e, \quad x'(0) = 0,$$

$$y(0) = 0, \qquad y'(0) = \alpha\left(\frac{1+e}{1-e}\right)^{1/2},$$

где e — некоторый параметр, такой, что $0 \leqslant e < 1$, то решение оказывается периодическим с периодом $2\pi/\alpha$. Орбита будет эллипсом с эксцентриситетом e, один фокус которого находится в начале координат.

Чтобы записать это как систему из четырех уравнений первого порядка, положим

$$y_1 = x, \quad y_2 = y, \quad y_3 = x', \quad y_4 = y'.$$

Уравнения и начальные условия приобретают теперь вид

$$R = \frac{(y_1^2+y_2^2)^{3/2}}{\alpha^2},$$

$$y_1' = y_3, \qquad y_1(0) = 1-e,$$

$$y_2' = y_4, \qquad y_2(0) = 0,$$

$$y_3' = -\frac{y_1}{R}, \qquad y_3(0) = 0,$$

$$y_4' = -\frac{y_2}{R}, \qquad y_4(0) = \alpha\left(\frac{1+e}{1-e}\right)^{1/2}.$$

Нормируя t, можно исключить α; но мы не делали этого, желая проиллюстрировать использование фортранной конструкции COMMON для передачи параметров типа α от основной программы к подпрограмме, определяющей уравнения.

Параметр IFLAG является важной контрольной переменной. При первом обращении к RKF45 ему следует присвоить значение 1. Обычно RKF45 изменяет его значение на 2, и при последующих обращениях нужно сохранить это значение. Значения, не равные 2, полученные на выходе RKF45, сигнализируют о наличии различных нерегулярностей или ошибок, подробно описываемых в комментариях. IFLAG=4 и IFLAG=7 — это предупреждения, что программе будет очень трудно получить требуе-

Таблица 6.2

Результаты, полученные иллюстрирующей программой.

0.0	0.750000000	0.000000000
0.5	0.619768032	0.477791373
1.0	0.294417538	0.812178519
1.5	−0.105176382	0.958038092
2.0	−0.490299793	0.939874996
2.5	−0.813942832	0.799590802
3.0	−1.054031517	0.575706078
3.5	−1.200735042	0.300160708
4.0	−1.250000001	−0.000000001
4.5	−1.200735042	−0.300160709
5.0	−1.054031517	−0.575706079
5.5	−0.813942832	−0.799590803
6.0	−0.490299793	−0.939874996
6.5	−0.105176383	−0.958038092
7.0	0.294417537	−0.812178518
7.5	0.619768031	−0.477791370
8.0	0.749999996	0.000000006
8.5	0.619768024	0.477791379
9.0	0.294417526	0.812178522
9.5	−0.105176395	0.958038091
10.0	−0.490299806	0.939874991
10.5	−0.813942843	0.799590794
11.0	−1.054031524	0.575706068
11.5	−1.200735047	0.300160697
12.0	−1.250000002	−0.000000011

мую точность. Пользователь может продолжать процесс, либо увеличить границы погрешностей, либо перейти к подпрограмме многошагового метода. IFLAG=3 указывает, что затребована слишком высокая относительная точность, а IFLAG=5 или 6 означают, что прежде, чем продолжать, нужно изменить допуски на ошибку. IFLAG=8 — указание на неправильность вызова RKF45. Мы настоятельно советуем пользователю включить в свою основную программу проверку параметра IFLAG [1]).

В данном иллюстративном счете было взято $e=0.25$ и $\alpha=\pi/4$; были напечатаны координаты тела для $0\leqslant t\leqslant 12$ с шагом 0.5. Эти результаты приведены в табл. 6.2. Отметьте, что орбита периодична и период равен 8.

[1]) Отметим уже здесь, что часто встречающиеся в комментариях к подпрограмме RKF45 термины «число вычислений производных» или «число значений функции» следует понимать как число обращений к подпрограмме (одновременного) вычисления правых частей системы.— *Прим. перев.*

```
C  ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ RKF45
C
      SUBROUTINE ORBIT(T, Y, YP)
      REAL T, Y(4), YP(4), R, ALFASQ
      COMMON ALFASQ
      R=Y(1)*Y(1)+Y(2)*Y(2)
      R=R*SQRT(R)/ALFASQ
      YP(1)=Y(3)
      YP(2)=Y(4)
      YP(3)=-Y(1)/R
      YP(4)=-Y(2)/R
      RETURN
      END
C
      EXTERNAL ORBIT
      REAL T, Y(4), TOUT, RELERR, ABSERR
      REAL TFINAL, TPRINT, ECC, ALFA, ALFASQ, WORK(27)
      INTEGER IWORK(5), IFLAG, NEQN
      COMMON ALFASQ
      ECC=0.25
      ALFA=3.141592653589/4.0
      ALFASQ=ALFA*ALFA
      NEQN=4
      T=0.0
      Y(1)=1.0-ECC
      Y(2)=0.0
      Y(3)=0.0
      Y(4)=ALFA*SQRT((1.0+ECC)/(1.0-ECC))
      RELERR=1.0E-9
      ABSERR=0.0
      TFINAL=12.0
      TPRINT=0.5
      IFLAG=1
      TOUT=T
   10 CALL RKF45(ORBIT, NEQN, Y, T, TOUT, RELERR, ABSERR,
     1 IFLAG, WORK, IWORK)
      WRITE(6, 11) T, Y(1), Y(2)
      GO TO(80, 20, 30, 40, 50, 60, 70, 80), IFLAG
   20 TOUT=T+TPRINT
      IF(T.LT.TFINAL) GO TO 10
      STOP
   30 WRITE(6, 31) RELERR, ABSERR
      GO TO 10
   40 WRITE (6, 41)
      GO TO 10
   50 ABSERR=1.0E-9
      WRITE (6, 31) RELERR, ABSERR
      GO TO 10
   60 RELERR=10.0*RELERR
      WRITE (6, 31) RELERR, ABSERR
      IFLAG=2
      GO TO 10
   70 WRITE (6, 71)
      IFLAG=2
      GO TO 10
   80 WRITE (6, 81)
      STOP
```

```
C
   11 FORMAT (F5.1, 2F15.9)
   31 FORMAT (17H TOLERANCES RESET, 2E12.3)
   41 FORMAT (11H MANY STEPS)
   71 FORMAT (12H MUCH OUTPUT)
   81 FORMAT (14H IMPROPER CALL)
      END
      SUBROUTINE RKF45 (F, NEQN, Y, T, TOUT, RELERR,
     1  ABSERR, IFLAG, WORK, IWORK)
C
CC    МЕТОД РУНГЕ—КУТТА—ФЕЛЬБЕРГА ЧЕТВЕРТОГО-ПЯТОГО
      ПОРЯДКА
C
CCC      СОСТАВИТЕЛИ ПРОГРАММЫ—H. A. WATTS, L. F. SHAMPINE
CCC                    SANDIA LABORATORIES
CCC                  ALBUQUERQUE, NEW MEXICO
CCC
CCC      RKF45 ПРЕДНАЗНАЧЕНА ГЛАВНЫМ ОБРАЗОМ ДЛЯ РЕШЕНИЯ
CCC  НЕЖЕСТКИХ И СЛАБО ЖЕСТКИХ ДИФФЕРЕНЦИАЛЬНЫХ
CCC  УРАВНЕНИЙ, КОГДА ВЫЧИСЛЕНИЕ ПРОИЗВОДНЫХ НЕ
CCC  СЛИШКОМ ДОРОГОСТОЯЩЕЕ. RKF45, ВООБЩЕ ГОВОРЯ,
CCC  НЕ СЛЕДУЕТ ИСПОЛЬЗОВАТЬ, ЕСЛИ ПОЛЬЗОВАТЕЛЮ
CCC  ТРЕБУЕТСЯ ВЫСОКАЯ ТОЧНОСТЬ
CCC
CC    РЕЗЮМЕ
CCC
CCC      ПОДПРОГРАММА RKF45 ИНТЕГРИРУЕТ СИСТЕМУ ИЗ NEQN
CCC  ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ
CCC  ПЕРВОГО ПОРЯДКА СЛЕДУЮЩЕГО ВИДА:
CCC           DY(I)/DT=F(T, Y(1), Y(2), ..., Y(NEQN)),
CCC  ГДЕ Y(I) ЗАДАНЫ В T.
CCC  ОБЫЧНО ПОДПРОГРАММУ ПРИМЕНЯЮТ ДЛЯ ИНТЕГРИРО-
CCC  ВАНИЯ ОТ T ДО TOUT, ОДНАКО ЕЕ МОЖНО ИСПОЛЬЗОВАТЬ
CCC  И КАК ОДНОШАГОВЫЙ ИНТЕГРАТОР, ЧТОБЫ ПРОДОЛЖИТЬ
CCC  РЕШЕНИЕ НА ОДИН ШАГ В НАПРАВЛЕНИИ TOUT. НА
CCC  ВЫХОДЕ ПАРАМЕТРАМ, ФИГУРИРУЮЩИМ В СПИСКЕ
CCC  ВЫЗОВА, ПРИСВАИВАЮТСЯ ЗНАЧЕНИЯ, НЕОБХОДИМЫЕ
CCC  ДЛЯ ПРОДОЛЖЕНИЯ ИНТЕГРИРОВАНИЯ. ПОЛЬЗОВАТЕЛЮ
CCC  НУЖНО ЛИШЬ ЕЩЕ РАЗ ОБРАТИТЬСЯ К RKF45 (И, ВОЗ-
CCC  МОЖНО, ОПРЕДЕЛИТЬ НОВОЕ ЗНАЧЕНИЕ ДЛЯ TOUT).
CCC  В ДЕЙСТВИТЕЛЬНОСТИ RKF45—ЭТО ПРОГРАММА ИНТЕР-
CCC  ФЕЙСА, КОТОРАЯ ВЫЗЫВАЕТ ПОДПРОГРАММУ RKFS,
CCC  ОСУЩЕСТВЛЯЮЩУЮ ПРОЦЕСС РЕШЕНИЯ. RKFS В СВОЮ
CCC  ОЧЕРЕДЬ ВЫЗЫВАЕТ ПОДПРОГРАММУ FEHL, КОТОРАЯ
CCC  ВЫЧИСЛЯЕТ ПРИБЛИЖЕННОЕ РЕШЕНИЕ НА ОДИН ШАГ.
CCC
CCC      RKF45 ИСПОЛЬЗУЕТ МЕТОД РУНГЕ—КУТТА—ФЕЛЬБЕРГА,
CCC  ОПИСАННЫЙ В СЛЕДУЮЩЕЙ ПУБЛИКАЦИИ: E. FEHLBERG,
CCC  LOW-ORDER CLASSICAL RUNGE—KUTTA FORMULAS WITH
CCC  STEPSIZE CONTROL, NASA TR R-315
CCC
CCC      СТИЛЬ РАБОТЫ ПРОГРАММЫ RKF45 ИЛЛЮСТРИРУЕТСЯ В
CCC  СЛЕДУЮЩИХ ПУБЛИКАЦИЯХ: L. F. SHAMPINE, H. A. WATTS,
CCC  S.DAVENPORT, SOLVING NON-STIFF ORDINARY DIFFEREN-
CCC  TIAL EQUATIONS—THE STATE OF THE ART, SANDIA
CCC  LABORATORIES REPORT SAND75-0182, SIAM REVIEW, 18(1976),
CC    N3, 376—411.
```

```
C
C
C     ПАРАМЕТРЫ ПРОГРАММЫ:
C        F—ПОДПРОГРАММА F(T, Y, YP) ДЛЯ ВЫЧИСЛЕНИЯ
C        ПРОИЗВОДНЫХ YP(I)=DY(I)/DT
C        NEQN—ЧИСЛО ИНТЕГРИРУЕМЫХ УРАВНЕНИЙ
C        Y(*)—РЕШЕНИЕ В ТОЧКЕ T
C        T—НЕЗАВИСИМАЯ ПЕРЕМЕННАЯ
C        TOUT—ТОЧКА ВЫХОДА, В КОТОРОЙ НУЖНО ОПРЕДЕ-
C        ЛИТЬ ЗНАЧЕНИЕ РЕШЕНИЯ
C        RELERR, ABSERR—ГРАНИЦЫ АБСОЛЮТНОЙ И ОТНО-
C           СИТЕЛЬНОЙ ПОГРЕШНОСТИ ДЛЯ ТЕСТА ЛОКАЛЬНОЙ
C           ОШИБКИ. НА КАЖДОМ ШАГЕ ПРОГРАММА
C           ТРЕБУЕТ ВЫПОЛНЕНИЯ УСЛОВИЯ
C     ABS (LOCAL ERROR). LE. RELERR*ABS(Y)+ABSERR
C              ДЛЯ КАЖДОЙ КОМПОНЕНТЫ ВЕКТОРОВ
C              ЛОКАЛЬНОЙ ОШИБКИ И РЕШЕНИЯ
C        IFLAG—УКАЗАТЕЛЬ РЕЖИМА ИНТЕГРИРОВАНИЯ.
C        WORK(*)—МАССИВ, СОДЕРЖАЩИЙ ИНФОРМАЦИЮ,
C        ВНУТРЕННЮЮ ДЛЯ RKF45, КОТОРАЯ НЕОБХОДИМА
C        ПРИ ПОСЛЕДУЮЩИХ ВЫЗОВАХ. ЕГО РАЗМЕРНОСТЬ
C        ДОЛЖНА БЫТЬ НЕ МЕНЬШЕ 3+6*NEQN
C        IWORK(*)—ЦЕЛЫЙ МАССИВ, СОДЕРЖАЩИЙ ИНФОРМА-
C           ЦИЮ, ВНУТРЕННЮЮ ДЛЯ RKF45, КОТОРАЯ
C           НЕОБХОДИМА ПРИ ПОСЛЕДУЮЩИХ ВЫЗОВАХ.
C           ЕГО РАЗМЕРНОСТЬ ДОЛЖНА БЫТЬ НЕ МЕНЬШЕ 5.
C
C  ПЕРВОЕ ОБРАЩЕНИЕ К RKF45
C
C     ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН ПРЕДУСМОТРЕТЬ В СВОЕЙ
C     ВЫЗЫВАЮЩЕЙ ПРОГРАММЕ ПАМЯТЬ ДЛЯ СЛЕДУЮЩИХ
C     МАССИВОВ, ФИГУРИРУЮЩИХ В СПИСКЕ ВЫЗОВА—
C     Y(NEQN), WORK(3+6*NEQN), IWORK(5); КРОМЕ ТОГО, ОН
C     ДОЛЖЕН ОБЪЯВИТЬ F В ОПЕРАТОРЕ EXTERNAL, ПОДГО-
C     ТОВИТЬ ПОДПРОГРАММУ F(T, Y, YP) И ПРИСВОИТЬ
C     НАЧАЛЬНЫЕ ЗНАЧЕНИЯ ПАРАМЕТРАМ—
C        NEQN—ЧИСЛО ИНТЕГРИРУЕМЫХ УРАВНЕНИЙ (NEQN.
C        GE. 1)
C        Y(*)—ВЕКТОР НАЧАЛЬНЫХ УСЛОВИЙ
C        T—НАЧАЛЬНАЯ ТОЧКА ИНТЕГРИРОВАНИЯ, T ДОЛЖНО
C        БЫТЬ ПЕРЕМЕННОЙ
C        TOUT—ТОЧКА ВЫХОДА, В КОТОРОЙ НУЖНО НАЙТИ
C              ЗНАЧЕНИЕ РЕШЕНИЯ. T=TOUT ВОЗМОЖНО
C              ЛИШЬ ПРИ ПЕРВОМ ОБРАЩЕНИИ. В ЭТОМ
C              СЛУЧАЕ ВЫХОД ИЗ RKF45 ПРОИСХОДИТ СО
C              ЗНАЧЕНИЕМ ПАРАМЕТРА IFLAG=2, ЕСЛИ
C              МОЖНО ПРОДОЛЖАТЬ ИНТЕГРИРОВАНИЕ.
C        RELERR, ABSERR—ГРАНИЦЫ ДЛЯ ОТНОСИТЕЛЬНОЙ И
C              АБСОЛЮТНОЙ ЛОКАЛЬНЫХ ПОГРЕШНОСТЕЙ.
C              ЭТИ ГРАНИЦЫ ДОЛЖНЫ БЫТЬ НЕОТРИЦА-
C              ТЕЛЬНЫ. RELERR ДОЛЖНА БЫТЬ ПЕРЕМЕННОЙ,
C              А ABSERR МОЖЕТ БЫТЬ И КОНСТАНТОЙ.
C              ПРОГРАММЕ, ВООБЩЕ ГОВОРЯ, НЕ СЛЕДУЕТ
C              ЗАДАВАТЬ ГРАНИЦУ ДЛЯ ОТНОСИТЕЛЬНОЙ
C              ОШИБКИ, МЕНЬШУЮ, ЧЕМ ПРИМЕРНО 1. E—8.
C              ДАБЫ ИЗБЕЖАТЬ ТРУДНОСТЕЙ, СВЯЗАННЫХ С
```

```
C               ОЧЕНЬ ВЫСОКИМИ ЗАПРОСАМИ К ТОЧНОСТИ,
C               ПРОГРАММА ТРЕБУЕТ, ЧТОБЫ RELERR БЫЛА
C               БОЛЬШЕ, ЧЕМ НЕКОТОРЫЙ ПАРАМЕТР ОТНО-
C               СИТЕЛЬНОЙ ОШИБКИ, ВЫЧИСЛЯЕМЫЙ ВНУТРИ
C               ЕЕ И ЗАВИСЯЩИЙ ОТ МАШИНЫ. В ЧАСТНОСТИ,
C               НЕ РАЗРЕШАЕТСЯ ЗАДАНИЕ ТОЛЬКО АБСО-
C               ЛЮТНОЙ ОШИБКИ. ЕСЛИ ЖЕ ЗАДАНО ЗНАЧЕНИЕ
C               RELERR, МЕНЬШЕЕ ДОПУСТИМОГО, ТО RKF45
C               УВЕЛИЧИВАЕТ RELERR НАДЛЕЖАЩИМ ОБРА-
C               ЗОМ И ВОЗВРАЩАЕТ УПРАВЛЕНИЕ ПОЛЬЗОВА-
C               ТЕЛЮ, ПРЕЖДЕ ЧЕМ ПРОДОЛЖАТЬ ИНТЕГРИ-
C               РОВАНИЕ
C       IFLAG—+1, —1. ЭТО УКАЗАТЕЛЬ НАСТРОЙКИ ПРО-
C               ГРАММЫ ДЛЯ КАЖДОЙ НОВОЙ ЗАДАЧИ.
C               НОРМАЛЬНОЕ ВХОДНОЕ ЗНАЧЕНИЕ РАВНО +1.
C               ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН ЗАДАВАТЬ IFLAG=—1
C               ЛИШЬ В ТОМ СЛУЧАЕ, КОГДА НЕОБХОДИМО
C               УПРАВЛЕНИЕ ОДНОШАГОВЫМ ИНТЕГРАТОРОМ.
C               В ЭТОМ СЛУЧАЕ RKF45 ПЫТАЕТСЯ ПРОДОЛЖИТЬ
C               РЕШЕНИЕ НА ОДИН ШАГ В НАПРАВЛЕНИИ
C               TOUT ПРИ КАЖДОМ ОЧЕРЕДНОМ ВЫЗОВЕ.
C               ПОСКОЛЬКУ ЭТОТ РЕЖИМ РАБОТЫ ВЕСЬМА
C               НЕЭКОНОМИЧЕН, ЕГО СЛЕДУЕТ ПРИМЕНЯТЬ
C               ЛИШЬ В СЛУЧАЕ КРАЙНЕЙ НЕОБХОДИМОСТИ.
C
C
C  ИНФОРМАЦИЯ НА ВЫХОДЕ
C       Y(*)—РЕШЕНИЕ В ТОЧКЕ T
C       T—ПОСЛЕДНЯЯ ТОЧКА, ДОСТИГНУТАЯ ПРИ ИНТЕГ-
C       РИРОВАНИИ.
C       IFLAG=2—ПРИ ИНТЕГРИРОВАНИИ ДОСТИГНУТО TOUT.
C               ЭТО ЗНАЧЕНИЕ ПАРАМЕТРА УКАЗЫВАЕТ НА
C               УСПЕШНЫЙ ВЫХОД И ЯВЛЯЕТСЯ НОРМАЛЬНЫМ
C               РЕЖИМОМ ДЛЯ ПРОДОЛЖЕНИЯ ИНТЕГРИРО-
C               ВАНИЯ.
C               =—2—БЫЛ ПРЕДПРИНЯТ ОДИН ШАГ В НАПРАВ-
C               ЛЕНИИ TOUT, ОКАЗАВШИЙСЯ УСПЕШНЫМ.
C               ЭТО НОРМАЛЬНЫЙ РЕЖИМ ДЛЯ ПРОДОЛЖЕНИЯ
C               ПОШАГОВОГО ИНТЕГРИРОВАНИЯ.
C               =3—ИНТЕГРИРОВАНИЕ НЕ БЫЛО ЗАКОНЧЕНО
C               ИЗ-ЗА ТОГО, ЧТО ЗАДАННОЕ ЗНАЧЕНИЕ ГРА-
C               НИЦЫ ДЛЯ ОТНОСИТЕЛЬНОЙ ОШИБКИ ОКАЗА-
C               ЛОСЬ СЛИШКОМ МАЛО. ДЛЯ ПРОДОЛЖЕНИЯ
C               ИНТЕГРИРОВАНИЯ RELERR БЫЛО НАДЛЕЖА-
C               ЩИМ ОБРАЗОМ УВЕЛИЧЕНО.
C               =4—ИНТЕГРИРОВАНИЕ НЕ БЫЛО ЗАКОНЧЕНО
C               ИЗ-ЗА ТОГО, ЧТО ПОТРЕБОВАЛОСЬ БОЛЕЕ 3000
C               ВЫЧИСЛЕНИЙ ПРОИЗВОДНОЙ. ЭТО СООТВЕТ-
C               СТВУЕТ ПРИБЛИЗИТЕЛЬНО 500 ШАГАМ.
C               =5—ИНТЕГРИРОВАНИЕ НЕ БЫЛО ЗАКОН-
C               ЧЕНО ИЗ-ЗА ТОГО, ЧТО РЕШЕНИЕ ОБРАТИЛОСЬ
C               В НУЛЬ, ВСЛЕДСТВИЕ ЧЕГО ТЕСТ ТОЛЬКО
C               ОТНОСИТЕЛЬНОЙ ОШИБКИ НЕ ПРОХОДИТ. ДЛЯ
C               ПРОДОЛЖЕНИЯ НЕОБХОДИМО НЕНУЛЕВОЕ
C               ЗНАЧЕНИЕ ПАРАМЕТРА ABSERR. ИСПОЛЬЗОВА-
C               НИЕ НА ОДИН ШАГ РЕЖИМА ПОШАГОВОГО
```

```
C                ИНТЕГРИРОВАНИЯ ЯВЛЯЕТСЯ РАЗУМНЫМ ВЫ-
C                ХОДОМ ИЗ ПОЛОЖЕНИЯ.
C                =6—ИНТЕГРИРОВАНИЕ НЕ БЫЛО ЗАКОНЧЕНО
C                ИЗ-ЗА ТОГО, ЧТО ТРЕБУЕМАЯ ТОЧНОСТЬ НЕ
C                МОГЛА БЫТЬ ДОСТИГНУТА ДАЖЕ ПРИ НАИ-
C                МЕНЬШЕЙ ДОПУСТИМОЙ ВЕЛИЧИНЕ ШАГА.
C                ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН УВЕЛИЧИТЬ ГРАНИЦУ
C                ПОГРЕШНОСТИ, ПРЕЖДЕ ЧЕМ МОЖНО БУДЕТ
C                ПОПЫТАТЬСЯ ПРОДОЛЖАТЬ ИНТЕГРИРОВАНИЕ.
C                =7—ПО ВСЕЙ ВИДИМОСТИ, RKF45 НЕЭФФЕК-
C                ТИВНА ПРИ РЕШЕНИИ ЭТОЙ ЗАДАЧИ. СЛИШ-
C                КОМ БОЛЬШОЕ ЧИСЛО ТРЕБУЕМЫХ ВЫХОД-
C                НЫХ ТОЧЕК ПРЕПЯТСТВУЕТ ВЫБОРУ ЕСТЕСТ-
C                ВЕННОЙ ВЕЛИЧИНЫ ШАГА. СЛЕДУЕТ ИСПОЛЬ-
C                ЗОВАТЬ РЕЖИМ ПОШАГОВОГО ИНТЕГРИРОВА-
C                НИЯ.
C                =8—НЕПРАВИЛЬНОЕ ЗАДАНИЕ ВХОДНЫХ ПА-
C                РАМЕТРОВ. ЭТО ЗНАЧЕНИЕ ПОЯВЛЯЕТСЯ, ЕСЛИ
C                ДОПУЩЕНА ОДНА ИЗ СЛЕДУЮЩИХ ОШИБОК —
C                    NEQN . LE. 0
C                    T=TOUT И IFLAG . NE. +1 ИЛИ —1
C                    RELERR ИЛИ ABSERR . LT. 0.
C                    IFLAG . EQ. 0 ИЛИ .LT. —2 ИЛИ .GT. 8
C           WORK(*), IWORK(*)—ИНФОРМАЦИЯ, КОТОРАЯ ОБЫЧНО
C                НЕ ПРЕДСТАВЛЯЕТ ИНТЕРЕСА ДЛЯ ПОЛЬЗОВА-
C                ТЕЛЯ, НО НЕОБХОДИМА ПРИ ПОСЛЕДУЮЩИХ
C                ВЫЗОВАХ. WORK(1), . . . , WORK(NEQN) СОДЕР-
C                ЖАТ ПЕРВЫЕ ПРОИЗВОДНЫЕ ВЕКТОРА РЕШЕ-
C                НИЯ Y В ТОЧКЕ T. WORK(NEQN+1) ХРАНИТ
C                ВЕЛИЧИНУ ШАГА H, С КОТОРОЙ МОЖНО ПО-
C                ПЫТАТЬСЯ ПРОВЕСТИ СЛЕДУЮЩИЙ ШАГ. В
C                IWORK(1) СОДЕРЖИТСЯ СЧЕТЧИК ЧИСЛА ВЫ-
C                ЧИСЛЕНИЙ ПРОИЗВОДНЫХ.
C     ПОСЛЕДУЮЩИЕ ОБРАЩЕНИЯ К RKF45
C        НА ВЫХОДЕ ПОДПРОГРАММЫ RKF45 ИМЕЕТСЯ ВСЯ ИН-
C        ФОРМАЦИЯ, НЕОБХОДИМАЯ ДЛЯ ПРОДОЛЖЕНИЯ ИНТЕГ-
C        РИРОВАНИЯ. ЕСЛИ ПРИ ИНТЕГРИРОВАНИИ ДОСТИГНУТО
C        TOUT, ТО ПОЛЬЗОВАТЕЛЮ ДОСТАТОЧНО ОПРЕДЕЛИТЬ НО-
C        ВОЕ ЗНАЧЕНИЕ TOUT И СНОВА ОБРАТИТЬСЯ К RKF45. В
C        РЕЖИМЕ ПОШАГОВОГО ИНТЕГРИРОВАНИЯ (IFLAG=—2)
C        ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН ИМЕТЬ В ВИДУ, ЧТО КАЖДЫЙ
C        ШАГ ВЫПОЛНЯЕТСЯ В НАПРАВЛЕНИИ ТЕКУЩЕГО ЗНАЧЕ-
C        НИЯ TOUT. ПО ДОСТИЖЕНИИ TOUT (СИГНАЛИЗИРУЕМОМ
C        ИЗМЕНЕНИЕМ IFLAG НА 2) ПОЛЬЗОВАТЕЛЬ ДОЛЖЕН ЗА-
C        ДАТЬ НОВОЕ ЗНАЧЕНИЕ TOUT И ПЕРЕОПРЕДЕЛИТЬ IFLAG
C        НА —2, ЧТОБЫ ПРОДОЛЖАТЬ В РЕЖИМЕ ПОШАГОВОГО
C        ИНТЕГРИРОВАНИЯ.
C        ЕСЛИ ИНТЕГРИРОВАНИЕ НЕ БЫЛО ЗАКОНЧЕНО, НО ПОЛЬ-
C        ЗОВАТЕЛЬ ХОЧЕТ ПРОДОЛЖАТЬ (СЛУЧАИ IFLAG=3,4), ОН
C        ПОПРОСТУ СНОВА ОБРАЩАЕТСЯ К RKF45. ПРИ IFLAG=3
C        ПАРАМЕТР RELERR БЫЛ ИЗМЕНЕН НАДЛЕЖАЩИМ ДЛЯ
C        ПРОДОЛЖЕНИЯ ИНТЕГРИРОВАНИЯ ОБРАЗОМ. В СЛУЧАЕ
C        IFLAG=4 СЧЕТЧИК ЧИСЛА ЗНАЧЕНИЙ ФУНКЦИИ БУДЕТ
C        ПЕРЕОПРЕДЕЛЕН НА 0, И БУДУТ РАЗРЕШЕНЫ ЕЩЕ 3000
C        ВЫЧИСЛЕНИЙ ФУНКЦИИ.
C        ОДНАКО В СЛУЧАЕ IFLAG=5, ПРЕЖДЕ ЧЕМ МОЖНО БУ-
C        ДЕТ ПРОДОЛЖАТЬ ИНТЕГРИРОВАНИЕ, ПОЛЬЗОВАТЕЛЬ
```

```
C     ДОЛЖЕН СНАЧАЛА ИЗМЕНИТЬ КРИТЕРИЙ ОШИБКИ, ЗАДАВ
C     ПОЛОЖИТЕЛЬНОЕ ЗНАЧЕНИЕ ДЛЯ ABSERR. ЕСЛИ ОН НЕ
C     СДЕЛАЕТ ЭТОГО, ВЫПОЛНЕНИЕ ПРОГРАММЫ БУДЕТ ПРЕ-
C     КРАЩЕНО.
C     ТОЧНО ТАК ЖЕ, В СЛУЧАЕ IFLAG=6, ПРЕЖДЕ ЧЕМ ПРО-
C     ДОЛЖАТЬ ИНТЕГРИРОВАНИЕ, ПОЛЬЗОВАТЕЛЮ НЕОБХО-
C     ДИМО ПЕРЕОПРЕДЕЛИТЬ IFLAG НА 2 (ИЛИ —2, ЕСЛИ ИС-
C     ПОЛЬЗУЕТСЯ РЕЖИМ ПОШАГОВОГО ИНТЕГРИРОВАНИЯ)
C     И УВЕЛИЧИТЬ ЗНАЧЕНИЕ ДЛЯ ABSERR ЛИБО RELERR,
C     ЛИБО И ДЛЯ ТОГО, И ДЛЯ ДРУГОГО. ЕСЛИ ЭТО НЕ БУДЕТ
C     СДЕЛАНО, ВЫПОЛНЕНИЕ ПРОГРАММЫ ПРЕКРАЩАЕТСЯ.
C     ПОЯВЛЕНИЕ IFLAG=6 УКАЗЫВАЕТ НА НЕРЕГУЛЯРНОСТЬ
C     (РЕШЕНИЕ БЫСТРО МЕНЯЕТСЯ ИЛИ, ВОЗМОЖНО, ИМЕЕТ-
C     СЯ ОСОБЕННОСТЬ), И ЧАСТО В ПОДОБНЫХ СЛУЧАЯХ НЕ
C     ИМЕЕТ СМЫСЛА ПРОДОЛЖАТЬ ИНТЕГРИРОВАНИЕ.
C     ЕСЛИ БУДЕТ ПОЛУЧЕНО ЗНАЧЕНИЕ IFLAG=7, ТО ПОЛЬ-
C     ЗОВАТЕЛЬ ДОЛЖЕН ПЕРЕЙТИ К РЕЖИМУ ПОШАГОВОГО
C     ИНТЕГРИРОВАНИЯ С ВЕЛИЧИНОЙ ШАГА, ОПРЕДЕЛЯЕМОЙ
C     ПРОГРАММОЙ, ИЛИ РАССМОТРЕТЬ ВОЗМОЖНОСТЬ ПЕРЕХО-
C     ДА НА ПРОГРАММЫ МЕТОДОВ АДАМСА. ЕСЛИ ВСЕ ЖЕ
C     ПОЛЬЗОВАТЕЛЬ ХОЧЕТ ПРОДОЛЖАТЬ ИНТЕГРИРОВАНИЕ
C     ПО ПОДПРОГРАММЕ RKF45, ОН ДОЛЖЕН ДО НОВОГО ОБ-
C     РАЩЕНИЯ К НЕЙ ПЕРЕОПРЕДЕЛИТЬ IFLAG НА 2. В ПРО-
C     ТИВНОМ СЛУЧАЕ ВЫПОЛНЕНИЕ ПРОГРАММЫ БУДЕТ
C     ПРЕКРАЩЕНО.
C     ЕСЛИ ПОЛУЧЕНО ЗНАЧЕНИЕ IFLAG=8, ТО ИНТЕГРИРОВА-
C     НИЕ НЕЛЬЗЯ ПРОДОЛЖАТЬ, ПОКА НЕ БУДУТ ИСПРАВЛЕ-
C     НЫ ОШИБОЧНЫЕ ВХОДНЫЕ ПАРАМЕТРЫ.
C     НУЖНО ОТМЕТИТЬ, ЧТО МАССИВЫ WORK И IWORK СО-
C     ДЕРЖАТ ИНФОРМАЦИЮ, НЕОБХОДИМУЮ ДЛЯ ДАЛЬНЕЙ-
C     ШЕГО ИНТЕГРИРОВАНИЯ. ПОЭТОМУ В ЭТИ МАССИВЫ НЕ-
C     ЛЬЗЯ ВНОСИТЬ ИЗМЕНЕНИЙ.
C     INTEGER NEQN, IFLAG, IWORK(5)
C     REAL Y(NEQN), T, TOUT, RELERR, ABSERR, WORK(1)
C     ЕСЛИ ТРАНСЛЯТОР ПРОВЕРЯЕТ ИНДЕКСЫ, ТО ЗАМЕНИТЬ
C     WORK(1) НА WORK(3+6*NEQN)
C     EXTERNAL F
C     INTEGER K1, K2, K3, K4, K5, K6, K1M
C     ВЫЧИСЛИТЬ ИНДЕКСЫ ДЛЯ РАСЩЕПЛЕНИЯ РАБОЧЕГО МАС-
C     СИВА
C     K1M=NEQN+1
C     K1=K1M+1
C     K2=K1+NEQN
C     K3=K2+NEQN
C     K4=K3+NEQN
C     K5=K4+NEQN
C     K6=K5+NEQN
C     ЭТА ПРОМЕЖУТОЧНАЯ ПРОГРАММА ПРОСТО СОКРАЩАЕТ ДЛЯ
C     ПОЛЬЗОВАТЕЛЯ ДЛИННЫЙ СПИСОК ВЫЗОВА ПУТЕМ РАСЩЕП-
C     ЛЕНИЯ ДВУХ РАБОЧИХ МАССИВОВ. ЕСЛИ ЭТО НЕ СОВМЕСТИ-
C     МО С ТРАНСЛЯТОРОМ, КОТОРЫЙ ИМЕЕТСЯ В РАСПОРЯЖЕНИИ
C     ПОЛЬЗОВАТЕЛЯ, ТО ОН ДОЛЖЕН ОБРАЩАТЬСЯ НЕПОСРЕД-
C     СТВЕННО К ПОДПРОГРАММЕ RKFS.
C     CALL RKFS(F, NEQN, Y, T, TOUT, RELERR, ABSERR, IFLAG,
     1          WORK(1), WORK(K1M), WORK(K1), WORK(K2),
     2          WORK(K3), WORK(K4), WORK(K5), WORK(K6),
```

```
     3              WORK(K6+1), IWORK(1), IWORK(2), IWORK(3),
                    IWORK(4), IWORK(5))
     4    RETURN
          END

          SUBROUTINE RKFS(F, NEQN, Y, T, TOUT, RELERR, ABSERR,
     1              IFLAG, YP, H, F1, F2, F3, F4, F5, SAVRE, SAVAE,
     2              NFE, KOP, INIT, JFLAG, KFLAG)
C         МЕТОД РУНГЕ—КУТТА—ФЕЛЬБЕРГА ЧЕТВЕРТОГО-ПЯТО-
C         ГО ПОРЯДКА
C         RKFS ИНТЕГРИРУЕТ СИСТЕМУ ОБЫКНОВЕННЫХ ДИФФЕ-
C         РЕНЦИАЛЬНЫХ УРАВНЕНИЙ ПЕРВОГО ПОРЯДКА (СМ. КОМ-
C         МЕНТАРИЙ К RKF45). МАССИВЫ YP, F1, F2, F3, F4 И F5
C         (РАЗМЕРНОСТИ ПО КРАЙНЕЙ МЕРЕ NEQN) И ПЕРЕМЕН-
C         НЫЕ H, SAVRE, SAVAE, NFE, KOP, INIT, JFLAG И KFLAG
C         ИСПОЛЬЗУЮТСЯ ВНУТРИ ПРОГРАММЫ И ВЫНЕСЕНЫ В
C         СПИСОК ВЫЗОВА, ЧТОБЫ СОХРАНИТЬ ИХ ОПРЕДЕЛЕН-
C         НОСТЬ ПРИ ПОВТОРНОМ ОБРАЩЕНИИ. ПОЭТОМУ ИХ ЗНА-
C         ЧЕНИЯ НЕ ДОЛЖНЫ ИЗМЕНЯТЬСЯ ПОЛЬЗОВАТЕЛЕМ.
C         ВОЗМОЖНЫЙ ИНТЕРЕС ПРЕДСТАВЛЯЮТ ПАРАМЕТРЫ
C         YP—ПРОИЗВОДНАЯ ВЕКТОРА РЕШЕНИЯ В ТОЧКЕ T
C         H—ПРЕДПОЛАГАЕМЫЙ РАЗМЕР ШАГА ДЛЯ ОЧЕРЕДНОГО
C         ЭТАПА
C         NFE—СЧЕТЧИК ЧИСЛА ВЫЧИСЛЕНИЙ ФУНКЦИИ
          LOGICAL HFAILD, OUTPUT
C
          INTEGER NEQN, IFLAG, NFE, KOP, INIT, JFLAG, KFLAG
          REAL Y(NEQN), T, TOUT, RELERR, ABSERR, H, YP(NEQN),
     1       F1(NEQN), F2(NEQN), F3(NEQN), F4(NEQN), F5(NEQN),
     2       SAVRE, SAVAE
C
          EXTERNAL F
C
          REAL A, AE, DT, EE, EEOET, ESTTOL, ET, HMIN, REMIN,
     1       RER, S, SCALE, TOL, TOLN, U26, EPSP1, EPS, YPK
C
          INTEGER K, MAXNFE, MFLAG
C
          REAL AMAX1, AMIN1
C         REMIN—ЭТО МИНИМАЛЬНОЕ ДОПУСТИМОЕ ЗНАЧЕНИЕ ДЛЯ
C         RELERR. ПОПЫТКИ ПОЛУЧИТЬ ПО ЭТОЙ ПОДПРОГРАММЕ
C         БОЛЕЕ ВЫСОКУЮ ТОЧНОСТЬ ОБЫЧНО СТОЯТ ОЧЕНЬ
C         ДОРОГО И ЗАЧАСТУЮ БЕЗУСПЕШНЫ.
          DATA REMIN /1. E—12/
C         СТОИМОСТЬ СЧЕТА КОНТРОЛИРУЕТСЯ ТРЕБОВАНИЕМ,
C         ЧТОБЫ КОЛИЧЕСТВО ВЫЧИСЛЕНИЙ ФУНКЦИИ БЫЛО ОГ-
C         РАНИЧЕНО ВЕЛИЧИНОЙ, ПРИБЛИЗИТЕЛЬНО РАВНОЙ ЗНА-
C         ЧЕНИЮ ПАРАМЕТРА MAXNFE. ПРИНЯТОЕ ЗДЕСЬ ЗНАЧЕ-
C         НИЕ ПРИМЕРНО СООТВЕТСТВУЕТ 500 ШАГАМ.
          DATA MAXNFE /3000/
C         ПРОВЕРИТЬ ВХОДНЫЕ ПАРАМЕТРЫ
          IF(NEQN .LT. 1) GO TO 10
          IF((RELERR. LT. 0. 0).OR.(ABSERR. LT. 0.0)) GO TO 10
          MFLAG= IABS(IFLAG)
          IF((MFLAG .EQ. 0) .OR. (MFLAG .GT. 8)) GO TO 10
          IF(MFLAG .NE. 1) GO TO 20
C         ПЕРВЫЙ ВЫЗОВ, ВЫЧИСЛИТЬ МАШИННОЕ ЭПСИЛОН
          EPS  =1.0
```

```
    5   EPS   =EPS/2.0
        EPSP1=EPS+1.0
        IF (EPSP1 .GT. 1.0) GO TO 5
        U26=26.0 *EPS
        GO TO 50
C
C       ОШИБКА ВО ВХОДНОЙ ИНФОРМАЦИИ
   10   IFLAG=8
        RETURN
C
C       ПРОВЕРИТЬ ВОЗМОЖНОСТЬ ПРОДОЛЖЕНИЯ
C
   20   IF((T .EQ. TOUT) .AND. (KFLAG .NE. 3)) GO TO 10
        IF(MFLAG .NE. 2) GO TO 25
C
C       IFLAG=+2 ИЛИ -2
        IF((KFLAG .EQ. 3) .OR. (INIT .EQ. 0)) GO TO 45
        IF(KFLAG .EQ. 4) GO TO 40
        IF((KFLAG .EQ. 5) .AND. (ABSERR .EQ. 0.0)) GO TO 30
        IF((KFLAG .EQ. 6) .AND. (RELERR .LE. SAVRE) .AND.
        1       (ABSERR .LE. SAVAE)) GO TO 30
        GO TO 50
C
C   IFLAG=3,4,5,6,7 ИЛИ 8
   25   IF(IFLAG .EQ. 3) GO TO 45
        IF(IFLAG .EQ. 4) GO TO 40
        IF((IFLAG .EQ. 5) .AND. (ABSERR. GT. 0.0)) GO TO 45
C
C   ИНТЕГРИРОВАНИЕ НЕЛЬЗЯ ПРОДОЛЖАТЬ, ПОСКОЛЬКУ ПОЛЬ-
C   ЗОВАТЕЛЬ НЕ ВЫПОЛНИЛ ИНСТРУКЦИЙ, СООТВЕТСТВУЮЩИХ
C   ЗНАЧЕНИЯМ IFLAG=5, 6, 7 ИЛИ 8
   30   STOP
C
C   ПЕРЕОПРЕДЕЛИТЬ СЧЕТЧИК ЧИСЛА ВЫЧИСЛЕНИЙ ФУНКЦИИ
   40   NFE=0
        IF(MFLAG .EQ. 2) GO TO 50
C
C       ПЕРЕОПРЕДЕЛИТЬ ЗНАЧЕНИЕ FLAG, УСТАНОВЛЕННОЕ
C       ПРИ ПРЕДЫДУЩЕМ ОБРАЩЕНИИ
   45   IFLAG=JFLAG
        IF(KFLAG .EQ. 3) MFLAG=IABS(IFLAG)
C
C       СОХРАНИТЬ ВХОДНОЕ ЗНАЧЕНИЕ IFLAG И УСТАНОВИТЬ
C       ЗНАЧЕНИЕ FLAG, СООТВЕТСТВУЮЩЕЕ ПРОДОЛЖЕНИЮ,
C       ДЛЯ БУДУЩЕЙ ВХОДНОЙ ПРОВЕРКИ
   50   JFLAG=IFLAG
        KFLAG=0
C
C       СОХРАНИТЬ ЗНАЧЕНИЯ RELERR И ABSERR ДЛЯ ВХОДНОЙ
C       ПРОВЕРКИ ПРИ ПОСЛЕДУЮЩИХ ОБРАЩЕНИЯХ
        SAVRE=RELERR
        SAVAE=ABSERR
C
C       УСТАНОВИТЬ ЗНАЧЕНИЕ ГРАНИЦЫ ДЛЯ ОТНОСИТЕЛЬ-
C       НОЙ ПОГРЕШНОСТИ, РАВНОЕ КАК МИНИМУМ 2*EPS+
C       +REMIN, ЧТОБЫ ИЗБЕЖАТЬ ТРУДНОСТЕЙ, СВЯЗАННЫХ
C       С ТРЕБОВАНИЕМ НЕДОСТИЖИМОЙ ТОЧНОСТИ
```

```
C
      RER=2.0*EPS+REMIN
      IF(RELERR .GE. RER) GO TO 55
C
C     ЗАДАННАЯ ГРАНИЦА ОТНОСИТЕЛЬНОЙ ПОГРЕШНОСТИ
C     СЛИШКОМ МАЛА
      RELERR=RER
      IFLAG  =3
      KFLAG  =3
      RETURN
C
   55 DT=TOUT-T
C
      IF(MFLAG .EQ. 1) GO TO 60
      IF(INIT .EQ. 0) GO TO 65
      GO TO 80
C
C     ПРИСВОЕНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ (ИНИЦИИРОВА-
C                 НИЕ)—УСТАНОВИТЬ ЗНАЧЕНИЕ УКАЗАТЕЛЯ
C                 ОКОНЧАНИЯ ИНИЦИИРОВАНИЯ, INIT
C                 УСТАНОВИТЬ ЗНАЧЕНИЕ УКАЗАТЕЛЯ СЛИШ-
C                 КОМ БОЛЬШОГО ЗАТРЕБОВАННОГО ЧИСЛА ВЫ-
C                 ХОДНЫХ ТОЧЕК, KOP
C                 ВЫЧИСЛИТЬ НАЧАЛЬНЫЕ ПРОИЗВОДНЫЕ
C                 УСТАНОВИТЬ ЗНАЧЕНИЕ СЧЕТЧИКА ЧИСЛА
C                 ВЫЧИСЛЕНИЙ ФУНКЦИИ, NFE
C                 ОЦЕНИТЬ НАЧАЛЬНУЮ ВЕЛИЧИНУ ШАГА
C
   60 INIT=0
      KOP=0
C
      A=T
      CALL F(A, Y, YP)
      NFE=1
      IF(T .NE. TOUT) GO TO 65
      IFLAG=2
      RETURN
C
C
   65 INIT=1
      H=ABS(DT)
      TOLN=0.
      DO 70 K=1, NEQN
         TOL=RELERR*ABS(Y(K))+ABSERR
         IF(TOL .LE. 0.) GO TO 70
         TOLN=TOL
         YPK=ABS(YP(K))
         IF(YPK*H**5 .GT. TOL) H=(TOL/YPK) **0.2
   70 CONTINUE
      IF(TOLN .LE. 0.0) H=0.0
      H=AMAX1 (H, U26*AMAX1 (ABS(T), ABS(DT)))
      JFLAG=ISIGN (2, IFLAG)
C
C
C     ПРИСВОИТЬ ВЕЛИЧИНЕ ШАГА ЗНАК, СООТВЕТСТВУЮЩИЙ
C     ИНТЕГРИРОВАНИЮ В НАПРАВЛЕНИИ ОТ T К TOUT
   80 H=SIGN (H, DT)
```

```
C
C       ПРОВЕРКА, НАСКОЛЬКО СЕРЬЕЗНО ВЛИЯНИЕ НА RKF45
C       СЛИШКОМ БОЛЬШОГО ЗАТРЕБОВАННОГО ЧИСЛА ВЫХОД-
C       НЫХ ТОЧЕК
C
        IF(ABS(H) .GE. 2.0*ABS(DT)) KOP=KOP+1
        IF(KOP .NE. 100) GO TO 85
C
C       ЧРЕЗМЕРНАЯ ЧАСТОТА ВЫХОДОВ
        KOP=0
        IFLAG=7
        RETURN
C
   85   IF(ABS(DT) .GT. U26*ABS(T)) GO TO 95
C
C       ЕСЛИ ОЧЕНЬ БЛИЗКО К ТОЧКЕ ВЫХОДА, ПРОЭКСТРАПО-
C       ЛИРОВАТЬ И ВЕРНУТЬСЯ ПО МЕСТУ ВЫЗОВА
C
        DO 90 K=1, NEQN
   90   Y(K)=Y(K)+DT*YP(K)
        A=TOUT
        CALL F(A, Y, YP)
        NFE=NFE+1
        GO TO 300
C
C       ПРИСВОИТЬ НАЧАЛЬНОЕ ЗНАЧЕНИЕ ИНДИКАТОРУ ТОЧКИ
C       ВЫХОДА
C
C
   95   OUTPUT=.FALSE.
C
C       ЧТОБЫ ИЗБЕЖАТЬ НЕОПРАВДАННОГО МАШИННОГО НУЛЯ
C       ПРИ ВЫЧИСЛЕНИИ ФУНКЦИИ ОТ ГРАНИЦ ПОГРЕШНО-
C       СТЕЙ, ПРОМАСШТАБИРОВАТЬ ЭТИ ГРАНИЦЫ
C
        SCALE=2.0/RELERR
        AE=SCALE*ABSERR
C
C
C       ПОШАГОВОЕ ИНТЕГРИРОВАНИЕ
C
  100   HFAILD=.FALSE.
C
C       УСТАНОВИТЬ НАИМЕНЬШУЮ ДОПУСТИМУЮ ВЕЛИЧИНУ
C       ШАГА
C
        HMIN=U(26)*ABS(T)
C
C       ИСПРАВИТЬ ПРИ НЕОБХОДИМОСТИ ВЕЛИЧИНУ ШАГА,
C       ЧТОБЫ ДОСТИГНУТЬ ТОЧКИ ВЫХОДА. РАССЧИТАТЬ НА
C       ДВА ШАГА ВПЕРЕД, ЧТОБЫ ИЗБЕЖАТЬ СЛИШКОМ РЕЗКИХ
C       ИЗМЕНЕНИЙ В ВЕЛИЧИНЕ ШАГА И ТЕМ САМЫМ УМЕНЬ-
C       ШИТЬ ВЛИЯНИЕ ВЫХОДНЫХ ТОЧЕК НА ПРОГРАММУ.
C
        DT=TOUT-T
        IF(ABS(DT) .GE. 2.0*ABS(H)) GO TO 200
        IF(ABS(DT) .GT. ABS(H)) GO TO 150
```

```
C
C     СЛЕДУЮЩИЙ УСПЕШНЫЙ ШАГ ЗАВЕРШИТ ИНТЕГРИРО-
C     ВАНИЕ ДО УКАЗАННОЙ ТОЧКИ ВЫХОДА
C
      OUTPUT=.TRUE.
      H=DT
      GO TO 200
C
  150 H=0.5*DT
C
C
C
C     ВНУТРЕННИЙ ОДНОШАГОВЫЙ ИНТЕГРАТОР
C
C     ГРАНИЦЫ ПОГРЕШНОСТЕЙ БЫЛИ ПРОМАСШТАБИРОВАНЫ,
C     ЧТОБЫ ИЗБЕЖАТЬ НЕОПРАВДАННОГО МАШИННОГО НУЛЯ
C     ПРИ ВЫЧИСЛЕНИИ ФУНКЦИЙ ЕТ ОТ НИХ.
C     ЧТОБЫ ИЗБЕЖАТЬ ОБРАЩЕНИЯ В НУЛЬ ЗНАМЕНАТЕЛЯ
C     В ТЕСТЕ, ОТНОСИТЕЛЬНАЯ ОШИБКА ИЗМЕРЯЕТСЯ ПО
C     ОТНОШЕНИЮ К СРЕДНЕМУ ИЗ ВЕЛИЧИН РЕШЕНИЯ
C     В НАЧАЛЕ И КОНЦЕ ШАГА.
C     В ФОРМУЛЕ, ОЦЕНИВАЮЩЕЙ ОШИБКУ, ПРОИЗВЕДЕНА
C     ГРУППИРОВКА СЛАГАЕМЫХ, УМЕНЬШАЮЩАЯ ПОТЕРЮ
C     ВЕРНЫХ ЗНАКОВ.
C     ЧТОБЫ РАЗЛИЧАТЬ МЕЖДУ СОБОЙ РАЗНЫЕ АРГУМЕНТЫ,
C     ДЛЯ H НЕ ДОПУСКАЮТСЯ ЗНАЧЕНИЯ, МЕНЬШИЕ УМНО-
C     ЖЕННОЙ НА 26 ОШИБКИ ОКРУГЛЕНИЯ В T.
C     ВВЕДЕНЫ ПРАКТИЧЕСКИЕ ОГРАНИЧЕНИЯ НА СКОРОСТЬ
C     ИЗМЕНЕНИЯ ВЕЛИЧИНЫ ШАГА, ЧТОБЫ СГЛАДИТЬ ПРО-
C     ЦЕСС ВЫБОРА ЭТОЙ ВЕЛИЧИНЫ И ИЗБЕЖАТЬ ЧРЕЗМЕР-
C     НОГО ЕЕ РАЗБРОСА В ЗАДАЧАХ С НАРУШЕНИЕМ НЕПРЕ-
C     РЫВНОСТИ.
C     ИЗ ПРЕДОСТОРОЖНОСТИ ПРОГРАММА БЕРЕТ 9/10 ОТ ТОЙ
C     ВЕЛИЧИНЫ ШАГА, КОТОРАЯ НУЖНА ПО ЕЕ ОЦЕНКЕ.
C     ЕСЛИ НА ДАННОМ ШАГЕ БЫЛА НЕУДАЧНАЯ ПОПЫТКА,
C     ТО ПРИ ПЛАНИРОВАНИИ СЛЕДУЮЩЕГО УВЕЛИЧЕНИЕ
C     ДЛИНЫ ШАГА НЕ ДОПУСКАЕТСЯ. ЭТО ПОВЫШАЕТ ЭФФЕК-
C     ТИВНОСТЬ ПРОГРАММЫ ДЛЯ ЗАДАЧ С РАЗРЫВАМИ И
C     В ОБЩЕМ СЛУЧАЕ, ПОСКОЛЬКУ ИСПОЛЬЗУЕТСЯ ЛОКАЛЬ-
C     НАЯ ЭКСТРАПОЛЯЦИЯ И ДОПОЛНИТЕЛЬНАЯ ПРЕДОСТО-
C     РОЖНОСТЬ КАЖЕТСЯ ОПРАВДАННОЙ.
C
C
C     ПРОВЕРИТЬ ЧИСЛО ВЫЧИСЛЕНИЙ ПРОИЗВОДНЫХ. ЕСЛИ
C     ОНО НЕ ПРЕВЫШАЕТ УСТАНОВЛЕННОГО ПРЕДЕЛА, ПО-
C     ПРОБОВАТЬ ПРОДОЛЖИТЬ ИНТЕГРИРОВАНИЕ С T ДО T+H
C
  200 IF(NFE. LE. MAXNFE) GO TO 220
C
C     СЛИШКОМ БОЛЬШАЯ РАБОТА
      IFLAG=4
      KFLAG=4
      RETURN
C
C     ПРОДОЛЖИТЬ ПРИБЛИЖЕННОЕ РЕШЕНИЕ НА ОДИН ШАГ
C     ДЛИНЫ H
```

```
C
  220 CALL FEHL(F, NEQN, Y, T, H, YP, F1, F2, F3, F4, F5, F1)
      NFE=NFE+5
C
C     ВЫЧИСЛИТЬ И СРАВНИТЬ ДОПУСТИМЫЕ ГРАНИЦЫ И
C     ОЦЕНКИ ЛОКАЛЬНОЙ ОШИБКИ, А ЗАТЕМ СНЯТЬ МАСШТА-
C     БИРОВАНИЕ ГРАНИЦ. ЗАМЕТЬТЕ, ЧТО ОТНОСИТЕЛЬНАЯ
C     ОШИБКА ИЗМЕРЯЕТСЯ ПО ОТНОШЕНИЮ К СРЕДНЕМУ ИЗ
C     ВЕЛИЧИН РЕШЕНИЯ В НАЧАЛЕ И КОНЦЕ ШАГА.
C
      EEOET=0.0
      DO 250 K=1, NEQN
         ET=ABS(Y(K))+ABS(F1(K))+AE
         IF(ET. GT. 0.0)GO TO 240
C
C        НЕПРАВИЛЬНАЯ ГРАНИЦА ПОГРЕШНОСТИ
         IFLAG=5
         RETURN
C
  240 EE=ABS((-2090.0*YP(K)+(21970.0*F3(K)-15048.0*F4(K)))
     1          +(22528.0*F2(K)-27360.0*F5(K)))
  250 EEOET=AMAX1(EEOET, EE/ET)
C
      ESTTOL=ABS(H)*EEOET*SCALE/752400.0
C
      IF(ESTTOL .LE. 1.0) GO TO 260
C
C
C     НЕУДАЧНЫЙ ШАГ
C              УМЕНЬШИТЬ ВЕЛИЧИНУ ШАГА И СНОВА ПО-
C              ПРОБОВАТЬ
C              УМЕНЬШЕНИЕ ОГРАНИЧИВАЕТСЯ СНИЗУ МНО-
C              ЖИТЕЛЕМ 1/10
      HFAILD=.TRUE.
      OUTPUT=.FALSE.
      S=0.1
      IF(ESTTOL .LT. 59049.0)S=0.9/ESTTOL**0.2
      H=S*H
      IF(ABS(H) .GT. HMIN) GO TO 200
C
C     ЗАДАННАЯ ГРАНИЦА ОШИБКИ НЕДОСТИЖИМА ДАЖЕ ПРИ
C     НАИМЕНЬШЕЙ ДОПУСТИМОЙ ВЕЛИЧИНЕ ШАГА
      IFLAG=6
      KFLAG=6
      RETURN
C
C
C     УСПЕШНЫЙ ШАГ
C              ПОМЕСТИТЬ В МАССИВ Y РЕШЕНИЕ В ТОЧКЕ
C              T+H И ВЫЧИСЛИТЬ ПРОИЗВОДНЫЕ В ЭТОЙ
C              ТОЧКЕ
C
  260 T=T+H
      DO 270 K=1, NEQN
  270    Y(K)=F1(K)
      A=T
      CALL F(A, Y, YP)
```

```
      NFE=NFE+1
C
C
C                 ВЫБРАТЬ ВЕЛИЧИНУ СЛЕДУЮЩЕГО ШАГА
C                 УВЕЛИЧЕНИЕ ОГРАНИЧЕНО МНОЖИТЕЛЕМ 5
C                 ЕСЛИ НА ДАННОМ ШАГЕ БЫЛА НЕУДАЧНАЯ
C                 ПОПЫТКА, ТО ДЛЯ СЛЕДУЮЩЕГО НЕ ДОПУ-
C                 СКАЕТСЯ ВЫБОР БОЛЬШЕЙ ВЕЛИЧИНЫ ШАГА
C
      S=5.0
      IF(ESTTOL .GT. 1.889568E-4)S=0.9/ESTTOL**0.2
      IF(HFAILD)S=AMIN1(S, 1.0)
      H=SIGN(AMAX1(S*ABS(H), HMIN), H)
C
C     КОНЕЦ ОДНОШАГОВОГО ИНТЕГРАТОРА
C
C
C     НУЖНО ЛИ ДЕЛАТЬ ОЧЕРЕДНОЙ ШАГ
C
      IF(OUTPUT)GO TO 300
      IF(IFLAG .GT. 0) GO TO 100
C
C
C     ИНТЕГРИРОВАНИЕ УСПЕШНО ЗАВЕРШЕНО
C
C     РЕЖИМ ОДНОШАГОВОГО ИНТЕГРИРОВАНИЯ
      IFLAG=-2
      RETURN
C
C     РЕЖИМ ИНТЕГРИРОВАНИЯ НА ИНТЕРВАЛЕ
  300 T=TOUT
      IFLAG=2
      RETURN
C
      END

      SUBROUTINE FEHL(F,NEQN, Y, T, H, YP, F1, F2, F3, F4, F5, S)
C
C     МЕТОД РУНГЕ—КУТТА—ФЕЛЬБЕРГА ЧЕТВЕРТОГО-ПЯТОГО
C     ПОРЯДКА
C
C     ПОДПРОГРАММА FEHL ИНТЕГРИРУЕТ СИСТЕМУ ИЗ NEQN
C     ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ
C     ПЕРВОГО ПОРЯДКА СЛЕДУЮЩЕГО ВИДА
C                  DY(I)/DT=F(T, Y(1), ...Y(NEQN)),
C     ГДЕ НАЧАЛЬНЫЕ ЗНАЧЕНИЯ Y(I) И НАЧАЛЬНЫЕ ПРОИЗ-
C     ВОДНЫЕ YP(I) ЗАДАНЫ В НАЧАЛЬНОЙ ТОЧКЕ T. FEHL
C     ПРОДОЛЖАЕТ РЕШЕНИЕ НА ФИКСИРОВАННЫЙ ШАГ H
C     И ПОМЕЩАЕТ В МАССИВ S(I) ПРИБЛИЖЕНИЕ К РЕШЕНИЮ
C     В ТОЧКЕ T+H, ИМЕЮЩЕЕ ПЯТЫЙ ПОРЯДОК ТОЧНОСТИ
C     (ЛОКАЛЬНЫЙ ПОРЯДОК РАВЕН ШЕСТИ). F1, ..., F5—МАС-
C     СИВЫ РАЗМЕРНОСТИ NEQN, НЕОБХОДИМЫЕ ВНУТРИ
C     ПРОГРАММЫ.
C     В ФОРМУЛАХ ПРОИЗВЕДЕНА ГРУППИРОВКА С ЦЕЛЬЮ
C     УМЕНЬШИТЬ ПОТЕРЮ ВЕРНЫХ ЗНАКОВ.
C     ЧТОБЫ МОЖНО БЫЛО РАЗЛИЧАТЬ РАЗНЫЕ НЕЗАВИ-
C     СИМЫЕ АРГУМЕНТЫ, ПРИ ОБРАЩЕНИИ К FEHL НЕ СЛЕ-
```

```
C     ДУЕТ ЗАДАВАТЬ ДЛЯ H ЗНАЧЕНИЕ, МЕНЬШЕЕ УМНОЖЕН-
C     НОЙ НА 13 ОШИБКИ ОКРУГЛЕНИЯ В T.
C
C
      INTEGER NEQN
      REAL Y(NEQN), T, H, YP(NEQN), F1(NEQN), F2(NEQN),
     1   F3(NEQN), F4(NEQN), F5(NEQN), S(NEQN)
C
      REAL CH
      INTEGER K
C
      CH=H/4.0
      DO 221 K=1, NEQN
  221    F5(K)=Y(K)+CH*YP(K)
      CALL F(T+CH, F5, F1)
C
      CH=3.0*H/32.0
      DO 222 K=1, NEQN
  222    F5(K)=Y(K)+CH*(YP(K)+3.0*F1(K))
      CALL F(T+3.0*H/8.0, F5, F2)
C
      CH=H/2197.0
      DO 223 K=1, NEQN
  223    F5(K)=Y(K)+CH*(1932.0*YP(K)+(7296.0*F2(K)-
     1   7200.0*F1(K)))
      CALL F(T+12.0*H/13.0, F5, F3)
C
      CH=H/4104.0
      DO 224 K=1, NEQN
  224    F5(K)=Y(K)+CH*((8341.0*YP(K)-845.0*F3(K))+
     1   (29440.0*F2(K)-32832.0*F1(K)))
      CALL F(T+H, F5, F4)
C
      CH=H/20520.0
      DO 225 K=1, NEQN
  225    F1(K)=Y(K)+CH*((-6080.0*YP(K)+(9295.0*F3(K)-
     1            5643.0*F4(K)))+(41040.0*F1(K)-28352.0*F2(K)))
      CALL F(T+H/2.0, F1, F5)
C
C     ВЫЧИСЛИТЬ ПРИБЛИЖЕННОЕ РЕШЕНИЕ В ТОЧКЕ T+H
C
      CH=H/7618050.0
      DO 230 K=1, NEQN
  230    S(K)=Y(K)+CH*((902880.0*YP(K)+(3855735.0*F3(K)-
     1            1371249.0*F4(K)))+(3953664.0*F2(K)+
     2            277020.0*F5(K)))
C
      RETURN
      END
```

УПРАЖНЕНИЯ

6.1. Функция ошибок определяется обычно посредством интеграла

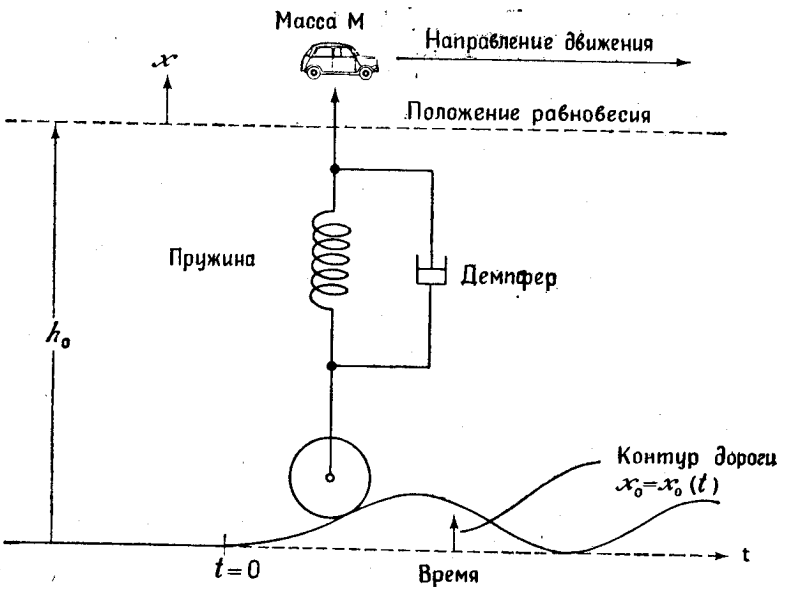
$$\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt,$$

но она может быть определена и как решение дифференциального уравнения

$$y'(x) = \frac{2}{\sqrt{\pi}} e^{-x^2},$$
$$y(0) = 0.$$

(Почему?) Напишите программу, которая, используя RKF45, печатает таблицу функции erf (x) для x=0.0, 0.1, 0.2, . . ., 1.9, 2.0. Сравните эту таблицу с той, что была получена в упр. 5.1. Если вы можете установить точное время выполнения программы на вашей машине, сравните время, необходимое этим двум методам для генерирования одной и той же таблицы.

6.2. Автомобиль массы M, поддерживаемый пружиной с демпфером, показанными на рисунке, перемещается с постоянной горизонтальной скоростью. В момент времени t=0 центр тяжести автомобиля находится на расстоянии h_0 от земли, и при этом вертикальная скорость отсутствует. В дальнейшем вертикальное смещение дороги от основного уровня описывается функцией $x_0(t)$.



Предположим, что пружина — линейная с коэффициентом упругости k, а коэффициент демпфирования r является нелинейной функцией относительной скорости двух концов демпфера:

$$r = r_0\left(1 + c\left|\frac{dx}{dt} - \frac{dx_0}{dt}\right|\right).$$

Легко показать, что смещение $x(t)$ центра тяжести автомобиля есть решение обыкновенного дифференциального уравнения второго порядка

$$M\frac{d^2x}{dt^2} = -k(x - x_0) - r\left(\frac{dx}{dt} - \frac{dx_0}{dt}\right)$$

с начальными условиями

$$x(0)=0,$$
$$\left.\frac{dx}{dt}\right|_{t=0}=0.$$

Напишите фортран-программу, вычисляющую $x(t)$, $0\leqslant t\leqslant t_{\max}$, для контура дороги, описываемого функцией

$$x_0(t)=A(1-\cos\omega t),$$

где $2A$ — максимальное смещение дороги относительно основного уровня.

Заметим, что для линейного случая ($c=0$) докритическому, критическому и закритическому демпфированию отвечают значения коэффициента

$$\xi=\frac{r}{2\sqrt{kM}},$$

соответственно меньшие, равные и большие единицы.

Ваша программа должна вводить значения параметров M, RO, C, K, A, W, TMAX и FREQ; параметр W соответствует ω. Параметр FREQ управляет частотой печатания, именно значения t, x_0, x, dx/dt, d^2x/dt^2 и $\xi(t)$ печатаются каждые FREQ секунд.

Предлагаемые тестовые данные:

M = 10 фунт·сек²/дюйм, A = 2 дюйм;
W = 7 рад/сек, TMAX = 5 сек, K = 640 фунт/дюйм.

Исследовать значения

RO (фунт·сек/дюйм)	C [(сек/дюйм)$^{1/2}$]
80	0
160	1
240	10

6.3. Основное уравнение теории балок имеет вид

$$\frac{d^2y}{dx^2}=-\frac{M}{EI},$$

где x — горизонтальное расстояние вдоль балки, y — вертикальный прогиб, M — изгибающий момент, E — модуль Юнга, наконец, I — момент площади (иногда называемый моментом инерции) поперечного сечения относительно нейтральной оси. Момент площади I не обязан быть постоянным, поскольку форма поперечного сечения балки может изменяться вдоль ее измерения x.

Легко показать, что

$$\frac{dM}{dx}=V,$$

где V — усилие сдвига, а

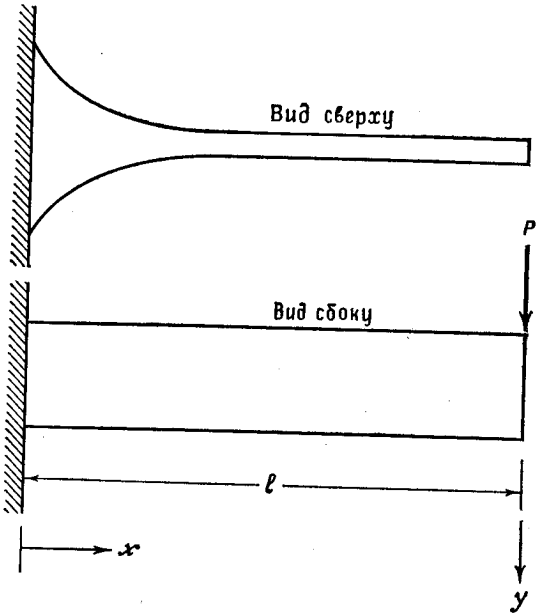
$$\frac{dV}{dx}=-w,$$

где $w(x)$ — нагрузка на балку.

Дифференцируя дважды основное уравнение и делая подстановку, получаем

$$\frac{d^4y}{dx^4}=\frac{-2}{l}\frac{dl}{dx}\frac{d^3y}{dx^3}-\frac{1}{l}\frac{d^2l}{dx^2}\frac{d^2y}{dx^2}+\frac{w}{EI}.$$

Большинство приложений этого уравнения приводят к краевым задачам. Однако, практический интерес представляет и следующая задача Коши.



Рассмотрим консольную балку с переменным поперечным сечением, показанную на рисунке. Длина балки равна l; в ее конце приложена нагрузка P. Для этой балки

$$V(x)=P$$

и

$$M(x)=-P(l-x).$$

Пусть

$$I(x)=5\left(1+4e^{-6x/l}\right) \text{ дюйм}^4,$$
$$E=30\times10^6 \text{ фунт/дюйм}^2,$$
$$l=100 \text{ дюйм},$$
$$P=500 \text{ фунт}.$$

Вычислите $y(l)$, предполагая, что балка не подвергается необратимым деформациям и не разрушается.

Указание: «фиксированный» конец консоля при $x=0$ означает, что $y(0)=0$ и $y'(0)=0$. Другие два начальных значения при $x=0$ можно найти из уравнений для $V(0)$ и $M(0)$.

6.4. Рассмотрим простую экосистему, состоящую из кроликов, для которых имеется неограниченный запас пищи, и лис, которые для пропитания охотятся за кроликами. Классическая математическая модель, принадлежащая

Вольтерра, описывает эту систему двумя нелинейными дифференциальными уравнениями первого порядка:

$$\frac{dr}{dt}=2r-\alpha rf, \quad r(0)=r_0,$$

$$\frac{df}{dt}=-f+\alpha rf, \quad f(0)=f_0,$$

где t — время, $r=r(t)$ — число кроликов, $f=f(t)$ — число лис и α — положительная константа. При $\alpha=0$ две популяции не взаимодействуют и кролики делают то, что у кроликов получается лучше всего, а лисы вымирают от голода. При $\alpha>0$ лисы встречают кроликов с вероятностью, пропорциональной произведению числа тех и других. В результате таких встреч число кроликов убывает, а число лис (по менее очевидным причинам) возрастает.

Исследуйте поведение этой системы для $\alpha=0.01$ и различных значений r_0 и f_0, простирающихся от 2 или 3 до нескольких тысяч. Начертите графики интересных решений или от руки, или используя любой доступный вам графопостроитель. Начертите также график с осями r и f. (Поскольку мы умалчиваем о единицах измерения, нет причин ограничивать r и f целыми значениями.)

(а) Вычислите решение для $r_0=300$ и $f_0=150$. Вы должны обнаружить из результата, что поведение системы периодично с периодом, очень близким к пяти единицам времени. Другими словами, $r(5)$ близко к $r(0)$, а $f(5)$ близко к $f(0)$.

(б) Вычислите решение для $r_0=15$ и $f_0=22$. Вы должны получить, что число кроликов в конечном счете становится меньше 1. Это можно интерпретировать так, что кролики вымирают. Найдите начальные условия, которые обрекают на вымирание лис. Найдите начальные условия с $r_0=f_0$, при которых вымирают оба вида.

(в) Может ли какая-либо компонента точного решения стать отрицательной? Может ли стать отрицательным численное решение? Что произойдет в этом случае? (На практике ответы на последние два вопроса могут зависеть от заданных вами границ погрешностей.)

(г) Было предложено много модификаций этой простой модели, чтобы более точно отразить то, что происходит в природе. Например, можно воспрепятствовать неограниченному возрастанию числа кроликов, изменив первое уравнение на

$$\frac{dr}{dt}=2\left(1-\frac{r}{R}\right)r-\alpha rf.$$

Теперь даже при $\alpha=0$ число кроликов никогда не может превысить R. Выберите какое-либо разумное значение для R и вновь рассмотрите некоторые из поставленных выше вопросов. В частности, что произойдет с периодичностью решений?

Данную модель подробно изучали и математики, и биологи. Основные математические свойства описаны в книге Дэвис (1962). Сравнение с реальными наблюдениями популяций рысей и зайцев в области, прилегающей к Гудзонову заливу, приведено в книге Лотка (1956), стр. 88—94. Обсуждение модификаций этой модели и дальнейшие ссылки можно найти в статьях Мэй (1972) и Гилпин (1972).

6.5. Знаменитой задачей нелинейной механики является задача о *перевернутом маятнике*. Маятник — это жесткий стержень длины L, поддерживаемый на одном конце шарниром без трения. Посредством электромотора вынуждают ся быстрые колебания поддерживающего шарнира в вертикальном направлении:

$$s=A\sin\omega t.$$

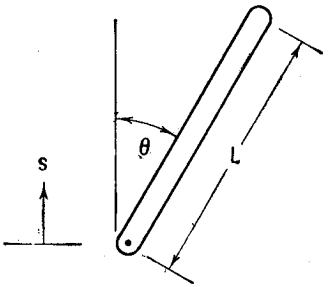
Простое применение второго закона Ньютона дает уравнение движения

$$\ddot{\theta} = \frac{3}{2L}(g - A\omega^2 \sin \omega t) \sin \theta,$$

где g — ускорение силы тяжести. Для малых значений θ справедливо $\sin \theta \approx \theta$, и это уравнение превращается в хорошо известное уравнение Матье; для некоторых значений A и ω и начальных условий уравнение Матье, как известно, устойчиво. При $A=0$ получаем знакомое уравнение маятника

$$\ddot{\theta} = \frac{3g}{2L} \sin \theta,$$

которое можно линеаризовать для значений θ, близких к π.



Самый интересный аспект этой задачи состоит в том, что существуют области, в которых уравнение движения устойчиво для начальных значений, соответствующих перевернутой конфигурации, и такие случаи были физически реализованы. Напишите фортран-программу, которая, используя RKF45, рассчитывает движение $\theta(t)$ для различных значений параметров L, A, ω и начальных значений $\theta(0)$ и $\dot{\theta}(0)$. Отлаживайте вашу программу для случая $L=10$ дюйм, $A=0$ дюйм, $\omega=0$ рад/сек, $\theta(0)=3.1$ рад и $\dot{\theta}(0)=0$ рад/сек; для g возьмите значение 386.09 дюйм/сек. Сравните вычисленное вами решение с аналитическим решением линеаризованного уравнения. Вам потребуется аналитическое решение и для того, чтобы определить разумные значения различных параметров подпрограммы RKF45. Напечатайте θ и $\dot{\theta}$ по крайней мере для двух колебаний маятника.

Когда ваша программа будет работать удовлетворительно, попробуйте более интересные случаи:

L	A	ω	$\theta(0)$	$\dot{\theta}(0)$
10	0.5	5.3	3.1	0
10	10	100	3.1	0
10	10	100	0.1	0
10	2	100	0.1	0
10	0.5	200	0.05	0

Интерпретируйте физически ваши решения.

6.6. Приводимые ниже дифференциальные уравнения описывают движение тела по орбите около двух много более тяжелых тел. Примером может быть капсула «Аполлона» на орбите около Земли и Луны. Координатная система

здесь несколько необычная. Три тела определяют в пространстве плоскость и двумерную декартову систему координат в этой плоскости. Начало находится в центре масс двух тяжелых тел, за ось x берется прямая, проходящая через эти два тела, а расстояние между ними принимается за единицу. Таким образом, если μ — отношение массы Луны к массе Земли, то Луна и Земля локализуются в точках с координатами $(1-\mu, 0)$ и $(\mu, 0)$ соответственно; координатная система перемещается при вращении Луны вокруг Земли. По предположению масса третьего тела, «Аполлона», пренебрежимо мала в сравнении с двумя другими, положение его как функция времени есть $(x(t), y(t))$. Уравнения выводятся из ньютонова закона движения и гравитационного закона обратных квадратов. Производные первого порядка в уравнениях возникают вследствие вращения системы координат:

$$x'' = 2y' + x - \frac{\mu^*(x+\mu)}{r_1^3} - \frac{\mu(x-\mu^*)}{r_2^3},$$

$$y'' = -2x' + y - \frac{\mu^* y}{r_1^3} - \frac{\mu y}{r_2^3},$$

$$\mu = \frac{1}{82.45}, \quad \mu^* = 1-\mu,$$

$$r_1 = ((x+\mu)^2 + y^2)^{1/2}, \quad r_2 = ((x-\mu^*)^2 + y^2)^{1/2}.$$

Хотя об этих уравнениях известно очень многое, не удается найти их решения в замкнутом виде. Один интересный круг вопросов связан с изучением периодических решений. Известно, что начальные условия

$$x(0) = 1.2, \quad x'(0) = 0,$$
$$y(0) = 0, \quad y'(0) = -1.04935751$$

приводят к периодическому решению с периодом $T = 6.19216933$. Это означает, что «Аполлон» стартует с указанной начальной скоростью, находясь за дальней стороной Луны на высоте, примерно равной 0.2 от расстояния Земля — Луна. Получающаяся орбита приводит «Аполлон» очень близко к Земле, затем далеко за противоположную от Луны сторону Земли, потом снова близко к Земле и, наконец, обратно за дальнюю сторону Луны в исходное положение и с исходной скоростью.

(а) Используя RKF45, вычислите решение с указанными начальными условиями. Проверьте, что решение — периодическое с приведенным выше периодом.

(б) Насколько близко подходит «Аполлон» на этой орбите к поверхности Земли? В уравнениях расстояния измеряются от центров Земли и Луны. Считайте, что Луна находится на расстоянии 238 000 миль от Земли, а Земля представляет собой шар с радиусом в 4 000 миль. Заметьте, что начало координатной системы находится внутри этого шара, но не совпадает с его центром.

(в) Модифицируйте подпрограмму FEHL, используемую в RKF45, так, чтобы она печатала положение «Аполлона» в начале каждого шага. Рассчитайте полностью орбиту и обратите внимание на изменения длины шага. Вы должны обнаружить, что делаются очень малые шаги, когда «Аполлон» находится вблизи Земли и его орбита быстро изменяется, и, напротив, делаются довольно большие шаги, когда «Аполлон» удаляется в глубокий космос. Испытайте несколько различных значений для границ погрешностей. Оцените, насколько больше работы потребовалось бы, чтобы получить ту же точность методом с фиксированной длиной шага. Если вы имеете доступ к графопостроителю, постройте что-нибудь вроде рис. 5.1, который показывает изменения длины шага.

6.7. Напишите программу, вычисляющую полные производные функции $f(y, t)$. Эти производные определяются формулами

$$f^{(0)} = f,$$
$$f^{(k+1)} = \frac{\partial f^{(k)}}{\partial y} \cdot f + \frac{\partial f^{(k)}}{\partial t}.$$

(а) Используйте арифметический язык типа ФОРТРАНа или АЛГОЛа. Считайте, что $f(y, t)$ — полином от двух переменных, задаваемый двумерным массивом своих коэффициентов.

(б) Используйте язык обработки символьной информации типа СНОБОЛа. Какой класс функций $f(y, t)$ можно обрабатывать, и как они могут быть представлены?

(в) Если имеется возможность, воспользуйтесь какой-либо системой символической алгебры, вроде Formac, Altran, Reduce, Scratchpad или Macsyma.

7. РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Пусть f — полином или трансцендентная функция одного переменного, действительного или комплексного. Задача состоит в том, чтобы найти один или более *нулей* f, т. е. решений уравнения $f(x)=0$. Нахождение формул для нулей полиномов было одним из важнейших разделов итальянской математики эпохи Ренессанса. Для полиномов 2-й, 3-й и 4-й степеней еще несколько столетий назад были найдены алгоритмы, выражающие корни посредством конечного числа квадратичных или кубичных радикалов [1]) и рациональных операций. Но только в тридцатых годах прошлого века Галуа доказал невозможность подобных алгоритмов для полиномов 5-й или более высокой степени, даже если допустить в формулах радикалы с показателем n.

Конечно, располагая вычислительной машиной, мы едва ли станем заботиться, разрешима ли задача в смысле Ренессанса. Используемые алгоритмы являются итерационными, и важны здесь такие вопросы: много ли требуется вычислений функции $f(x)$, нужны ли вычисления производных $f'(x)$ и $f''(x)$ и т. д.

В § 7.1 и 7.2 мы обсудим методы вычисления действительных нулей функций f общего вида. В § 7.3 речь пойдет о вычислении комплексных нулей. Специальные методы для нахождения нулей полиномов вкратце описываются в § 7.4.

Обобщение этой задачи приводит к вычислению некоторых или всех решений системы из n нелинейных алгебраических или трансцендентных уравнений с n неизвестными. Этому вопросу посвящен небольшой § 7.5.

7.1. Трансцендентные уравнения — действительные корни

Пусть $f(x)$ — функция одного действительного переменного x. Чтобы начать поиск нуля $f(x)$, предположим, что можно найти

[1]) Имеется в виду — от коэффициентов полинома.— *Прим. перев.*

интервал $[a, b]$, на котором $f(x)$ меняет знак. Однако если мы ничего не знаем относительно f, то мы не можем быть уверены, что она имеет нуль на этом интервале. Даже если *математическая* функция непрерывна и изменяет знак в $[a, b]$, *вычисляемая* функция переменного с плавающей точкой, значения которой также являются числами с плавающей точкой, принимает лишь дискретное множество значений, среди которых, возможно, нет нуля. Поэтому в конечном счете более практично искать не нуль f, а *малый* интервал $[\alpha, \beta]$, в котором f меняет знак. Такой интервал всегда можно найти, и можно сузить его настолько, насколько позволяет система чисел с плавающей точкой; т. е. так, чтобы концевыми точками были два соседних числа этой системы. Если о функции f ничего не известно, то наиболее надежным алгоритмом является *метод бисекции* [1]). При заданной точности ε метод состоит из таких шагов:

1. Положить $\alpha=a$ и $\beta=b$. Вычислить $f(\alpha)$ и $f(\beta)$.
2. Положить $\gamma=(\alpha+\beta)/2$. Вычислить $f(\gamma)$.
3. Если $\operatorname{sign}(f(\gamma))=\operatorname{sign}(f(\alpha))$, то заменить α на γ; в противном случае заменить β на γ.
4. Если $\beta-\alpha>\varepsilon$, то перейти к шагу 2; в противном случае — останов.

Ясно, что в каждой итерации алгоритма бисекции приобретается один бит точности. Таким образом, при критерии $\beta-\alpha \leqslant 2^{-t}(b-a)$ нужно t итераций. На машинах серий IBM 360/370 в арифметике с удвоенной точностью требуется приблизительно 56 итераций, чтобы сократить исходный интервал $[1, 16]$ до двух последовательных чисел с плавающей точкой.

Алгоритм бисекции довольно медлителен, но зато абсолютно застрахован от неудачи. Если каждое вычисление $f(x)$ несложно, то обычно нет серьезных причин, чтобы отвергнуть этот метод. Единственная причина, почему мы не используем его,— это то, что имеется другой алгоритм, ZEROIN, который гарантированно не может быть много медленней бисекции и быстрей ее, когда f — гладкая функция. Добавочная скорость очень полезна, если вычисление $f(x)$ требует много времени.

Если математическая функция f достаточно гладкая (имеет одну или две непрерывные производные), то часто есть возможность значительно сократить число вычислений функции по сравнению с методом бисекции. Было изучено большое число различных итерационных методов, из которых мы обсудим здесь только метод Ньютона и метод секущих. Дальнейшие сведения об этих и прочих методах можно найти в статье Трауба: «Решение трансцендентных уравнений» (Рэлстон, Уилф (1967)).

[1]) Или половинного деления.— *Прим. перев.*

В методе Ньютона (иногда называемом также методом Ньютона — Рафсона) нуль r функции f находится как предел последовательности действительных чисел $\{x_k\}$. (Мы предполагаем сейчас точную арифметику действительных чисел.) Каждое новое приближение x_{k+1} вычисляется как единственный нуль касательной прямой к функции $y=f(x)$ в точке x_k, т. е. путем *локальной линеаризации* f около x_k. Как показывается в анализе,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

при условии, что $f'(x_k) \neq 0$.

Теорема

Если $f(r)=0$, а $f'(r)\neq 0$ и f'' непрерывна, то существует открытый интервал $N(r)$, содержащий r и такой, что если x_1 принадлежит $N(r)$, то для метода Ньютона $x_k \to r$ при $k\to\infty$. Более того, если обозначить через e_k ошибку приближения x_k : $e_k = x_k - r$, то

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{f''(r)}{2f'(r)}.$$

Вместо того чтобы доказывать теорему, мы обсудим ее. Условие означает, что r — простой нуль f. Утверждение устанавливает, что ошибка e_{k+1} примерно равна Ce_k^2, где $C=f''(r)/2f'(r)$. Итак, говоря приблизительно, каждая итерация возводит ошибку e_k в квадрат. При $e_k \to 0$ число правильных десятичных (или двоичных) цифр примерно удваивается на каждой итерации. Если C порядка 1, то при x_k столь близком к r, что $|e_k| < \frac{1}{2}$, имеем

$$|e_{k+1}| < \frac{1}{2^2}, \quad |e_{k+2}| < \frac{1}{2^4}, \quad \ldots, \quad |e_{k+6}| < \frac{1}{2^{64}}.$$

Таким образом, грубо говоря, нужно примерно шесть итераций, чтобы сократить ошибку от $\frac{1}{2}$ до наименьшего значения, возможного для плавающей арифметики IBM 360 с удвоенной точностью. Это нужно сопоставить с приблизительно 56 итерациями, необходимыми для достижения той же точности методом бисекции.

Об итерационном процессе, для которого ошибка e_k удовлетворяет соотношению

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^d} = C \neq 0,$$

говорят, что он имеет *сходимость порядка d*. В условиях теоремы, если $f''(r)\neq 0$, метод Ньютона имеет сходимость порядка 2, иногда называемую также *квадратичной сходимостью*.

По мере того как ошибка в методе Ньютона уменьшается до значений, сравнимых с расстоянием между соседними числами с плавающей точкой, зернистая структура числовой системы делает продолжение алгоритма невозможным.

Подлинная трудность в методе Ньютона заключается в выборе начального приближения x_1, которое бы находилось внутри интервала $N(r)$ искомого нуля r. Если график f имеет вид, показанный на рис. 7.1, то $N(r)$ приблизительно совпадает с интервалом $(r-\varepsilon,\ r+\varepsilon)$. Если x_1 взято вне $N(r)$, то последовательные ите-

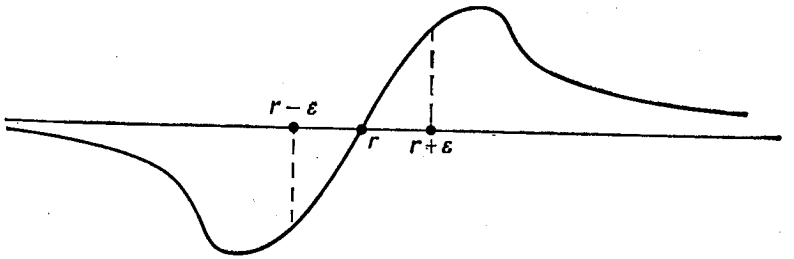


Рис. 7.1.

рации метода Ньютона все больше удаляются от r и нуль не будет найден.

Вследствие этого методу Ньютона часто предшествует какой-нибудь *глобально сходящийся* алгоритм типа бисекции, прежде чем можно будет переключиться на быстро сходящиеся ньютоновы итерации. Таким образом, метод Ньютона зачастую является лишь завершающей процедурой более медленного, но зато гарантированного начального алгоритма. При таком комбинировании, к примеру, последние 25 или около того итераций бисекции могут быть заменены 6 ньютоновыми шагами.

Если r — не простой нуль, так что $f'(r)=0$, то условия теоремы нарушаются. Сходимость метода Ньютона к двойному корню имеет порядок 1 (называется также *линейной сходимостью*), а не 2. Например, для $f(x)=x^2$ в ньютоновом процессе $x_{k+1}=x_k/2$, следовательно, $e_{k+1}/e_k=\frac{1}{2}$ при всех k.

Заметим, что каждая итерация метода Ньютона требует вычисления не только $f(x)$, но и $f'(x)$. Есть функции, для которых вычисление $f'(x)$ после того, как найдено $f(x)$, очень дешево. Для других функций стоимость вычисления $f'(x)$ эквивалентна второму вычислению $f(x)$. Наконец, для третьих функций вычисление $f'(x)$ почти невозможно.

Главное достоинство метода Ньютона состоит в том, что с его помощью можно находить комплексные нули аналитических функций f, а также в том, что его можно распространить на решение систем нелинейных уравнений с многими переменными.

При нахождении нулей функции f, для которой вычисление $f'(x)$ затруднено, *метод секущих* является часто лучшим выбором, чем метод Ньютона. В этом алгоритме начинают с двумя исходными числами x_1 и x_2. На каждом шаге x_{k+1} получают из x_k и x_{k-1} как единственный нуль линейной функции, принимающей значения $f(x_k)$ в x_k и $f(x_{k-1})$ в x_{k-1}. Эта линейная функция представляет секущую к кривой $y=f(x)$, проходящую через ее точки

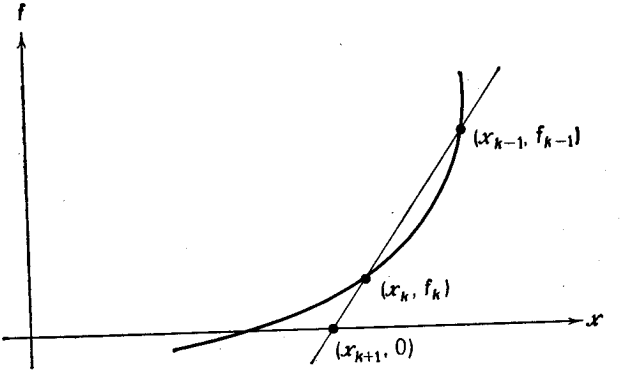


Рис. 7.2.

с абсциссами x_k и x_{k-1} — отсюда название *метод секущих* (рис. 7.2).

Легко показать, что

$$x_{k+1} = x_k - \frac{f_k}{(f_{k-1} - f_k)/(x_{k-1} - x_k)},$$

где $f_k = f(x_k)$. (Правую часть лучше не приводить к общему знаменателю. Почему?) Теорема сходимости для метода секущих формулируется так:

Теорема

Если $f(r)=0$, *но* $f'(r)\neq 0$ *и* $f''(r)\neq 0$, *а* f'' *непрерывна, то существует открытый интервал* $N(r)$, *содержащий* r *и такой, что если* x_1 *и* x_2 *— различные точки из* $N(r)$, *то последовательность* x_k *сходится к* r *при* $k\to\infty$. *Более того,*

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^p} = C \neq 0,$$

где $p = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$.

Доказательство значительно более сложно, чем простое доказательство, возможное для метода Ньютона. Утверждение теоремы означает, что для достаточно хороших начальных приближений метод секущих сходится к простому нулю функции,

имеющей непрерывную вторую производную, и при этом сходимость имеет порядок 1.618.

Если учесть, что на каждом шаге метода Ньютона требуется два вычисления функции (одно вычисление f и одно f'), то можно считать, что порядок сходимости метода в пересчете на одно вычисление функции равен $\sqrt{2}\approx 1.414$. Поскольку шаг метода секущих требует лишь одного вычисления функции, этот метод может расцениваться как более быстрый по сравнению с методом Ньютона.

Как и метод Ньютона, метод секущих очень хорошо работает для аналитических функций комплексного переменного. Однако обобщение метода на системы уравнений по-видимому довольно трудно, хотя и возможно.

Подобно методу Ньютона, наибольшая трудность в методе секущих заключается в нахождении x_1 и x_2, достаточно близких к r для того, чтобы могла начаться сходимость.

Если $f(x)$ была вычислена более чем в двух точках, то, видимо, разумно использовать эту информацию для улучшения последующих оценок нуля. Одним из подобных методов является *обратная квадратичная интерполяция*, где берутся три точки: x_{k-2}, x_{k-1} и x_k. Пусть $g(y)$ — квадратичный многочлен от переменного y, для которого $x_i=g(f_i)$, $i=k-2$, $k-1$, k. Тогда в качестве следующего приближения к нулю берется $x_{k+1}=g(0)$. Это можно записать непосредственно в терминах трех значений x и трех значений f, однако точная формула не важна здесь. Необходимо, чтобы три значения f были различны. Если это не так, в формуле произойдет деление на нуль.

Скорость сходимости обратной квадратичной интерполяции равна 1.839, что несколько быстрее, чем в методе секущих. Однако нужны три начальных значения, и если они выбраны недостаточно близко к нулю, то поведение алгоритма может быть весьма странным.

7.2. Подпрограмма ZEROIN

Один из лучших имеющихся машинных алгоритмов для нахождения действительного нуля функции сочетает безотказность бисекции с асимптотической скоростью метода секущих в случае гладких функций. Он называется ZEROIN и был изобретен в 1960-х годах в Математическом центре Амстердама (Ван Вейнгарден, Деккер и др.). Описание и анализ даны в публикации Уилкинсон (1967), именно с. 8—12. Впервые алгоритм был опубликован Деккером (1969) и затем улучшен Брентом (1973).

Мы используем фортранную реализацию алгоритма Деккера в версии Брента. В конце этого параграфа помещен текст подпро-

граммы-функции ZEROIN. Типичное обращение к ZEROIN имеет вид

$$ZZ = ZEROIN(A, B, F, TOL).$$

Здесь A, B — концевые точки интервала, на котором ищется нуль. Параметр F обозначает вещественную подпрограмму-функцию, имеющую аргументом одну вещественную переменную. TOL — это граница погрешности, допустимой в результате. Программа предполагает без проверки, что F(A) и F(B) имеют разные знаки.

ZEROIN выполняет итерационный процесс, в котором на каждом шаге присутствуют три абсциссы A, B и C. Обычно

1. B — последнее и наилучшее приближение к нулю.
2. A — предыдущее приближение.
3. C — предыдущее или еще более раннее приближение, такое, что F(B) и F(C) имеют противоположные знаки.

Во всех случаях B и C ограничивают нуль. Кроме того, $|F(B)| \leqslant |F(C)|$. Если длина интервала |B—C| уменьшилась настолько, что выполняется условие

$$|B-C| \leqslant TOL + 4.*EPS*ABS(B),$$

то значение B выдается как значение функции ZEROIN. Кроме TOL, в проверке сходимости участвует параметр машинной точности EPS, чтобы подстраховать возможный случай, когда заданное значение TOL слишком мало. В частности, чтобы заставить ZEROIN найти наименьший возможный интервал, нужно задать TOL равным нулю.

На каждом шаге ZEROIN выбирает очередное приближение из двух кандидатов — один получен алгоритмом бисекции, а другой — алгоритмом интерполяции. Если A, B и C различны, используется обратная квадратичная интерполяция; в противном случае — линейная интерполяция (метод секущих). Если точка, полученная интерполяцией, «разумна», то выбирается она; иначе выбирается точка бисекции. Определение «разумности» довольно техническое, но по существу оно означает, что точка находится внутри текущего интервала и не слишком близка к его концам. Следовательно, длина интервала гарантированно убывает на каждом шаге и убывает быстро, если функция хорошо ведет себя. Более подробно см. в книге Брент (1973).

Нужно упомянуть некоторые программистские детали ZEROIN, поскольку они могут быть важны и в других ситуациях. Точка бисекции вычисляется как

$$XM = B + 0.5*(C-B),$$

а не по обычной формуле

$$XM = 0.5 * (C + B).$$

Чтобы это понять, возьмем C=0.982 и B=0.987. Предположим, что вычисления выполняются на машине с трехзначной десятичной плавающей арифметикой с усечением. Тогда для C+B будет получено значение 1.96 и обычная формула даст в качестве средней точки 0.980, что находится вне интервала. Общий принцип таков: лучше преобразовать формулы, чтобы они выражали желаемую величину как малую поправку к хорошему приближению.

Большое внимание уделено проблеме машинных нулей и переполнений. Например, проверка, что F(B) и F(C) имеют разные знаки, отличается от обычного условия

$$F(B) * F(C) < 0.0.$$

Если $F(B)=10^{-40}$ и $F(C)=-10^{-40}$, то они имеют разные знаки; однако на многих машинах произведение будет машинным нулем и тест не будет выполнен. Точка, получаемая интерполяционными алгоритмами, записывается в виде

$$B+P/Q,$$

но деление не производится, пока это не будет необходимо и безопасно. Если берется точка бисекции, то эта величина не нужна вовсе.

Дадим теперь краткое резюме того, что утверждает Брент относительно своей версии алгоритма ZEROIN. Во-первых, она всегда сходится, даже для плавающей арифметики. Во-вторых, количество вычислений функции не может превысить числа, равного примерно

$$\left[\log_2\left(\frac{B-A}{TOL1}\right)\right]^2,$$

где TOL1=0.5*TOL+2.0*EPS*ABS(B). В-третьих, нуль R, получаемый алгоритмом, таков, что F гарантированно меняет знак в определенном интервале, приблизительно совпадающем с [R—2*TOL1, R+2*TOL1]. В-четвертых, Брент очень широко проверял свой алгоритм на весьма разнообразных функциях и установил, что в типическом случае для гладкой функции требуется порядка 10 вычислений значений. В-пятых, он ни разу не обнаружил функции, для которой потребовалось бы более трехкратного числа вычислений по сравнению с методом бисекции, т. е. больше 170 на IBM 360 для корня, не находящегося вблизи нуля. В-шестых, если функция F достаточно гладкая, например имеет непрерывную вторую производную вблизи простого нуля R, то алгоритм ZEROIN (начатый достаточно близко к R) постепенно перестанет делать бисекции и будет сходиться к R посред-

ством процесса, очень схожего с методом секущих и имеющего скорость сходимости не меньше 1.618. Доказательства этих утверждений см. в книге Брент (1973).

Приводимая ниже программа иллюстрирует использование ZEROIN для простого примера $f(x)=x^3-2x-5$. Из приблизительного графика $f(x)$ легко видеть, что имеется только один действительный нуль и что он лежит между 2 и 3.

Результат: Z=2.0945514815.

В этом примере мы имеем дело с полиномом, к которому можно было бы применить методы § 7.4, однако здесь он использован по причинам исторического интереса. В книге Уиттекера и Робинсона «Анализ наблюдений» (Whittaker, Robinson, The Calculus of Observations, 1924) приводится следующий отрывок из письма, написанного де Морганом Вивеллу (Whewell) в 1861 г.: «Я потому называю $x^3-2x-5=0$ прославленным уравнением, что именно на нем Валлис рискнул применить метод Ньютона, когда тот впервые опубликовал его, вследствие чего *каждый* численный метод должен проявить себя в том числе и на этом примере. Изобрести численный метод и пренебречь показать, как он работает для этого уравнения, означает оказаться в положении паломника, не желающего войти в маленькую дверцу. (см. J. Bunyan)» [1].

```
C   ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ ZEROIN
C
      REAL FUNCTION F(X)
      REAL X
      F=X*(X*X-2.)-5.
      RETURN
      END
C
      EXTERNAL F
      REAL A, B, Z, TOL, ZEROIN
        A=2.
        B=3.
      TOL=1.0E-10
        Z=ZEROIN(A, B, F, TOL)
      WRITE(6, 1)Z
    1 FORMAT(3H Z=, F15.10)
      STOP
      END

      REAL FUNCTION ZEROIN(AX, BX, F, TOL)
      REAL AX, BX, F, TOL
C
C       НУЛЬ ФУНКЦИИ F(X) ВЫЧИСЛЯЕТСЯ В ИНТЕРВАЛЕ
C       AX, BX
C
C     ВХОДНАЯ ИНФОРМАЦИЯ..
```

[1] Де Морган имеет в виду книгу Pilgrim's Progress английского религиозного писателя Джона Баниана (1628—1688). — *Прим. перев.*

```
C
C       AX          ЛЕВЫЙ КОНЕЦ ИСХОДНОГО ИНТЕРВАЛА
C       BX          ПРАВЫЙ КОНЕЦ ИСХОДНОГО ИНТЕРВАЛА
C       F           ПОДПРОГРАММА-ФУНКЦИЯ, КОТОРАЯ ВЫЧИ-
C                   СЛЯЕТ F(X)ДЛЯ ЛЮБОГО X В ИНТЕРВАЛЕ AX, BX
C       TOL         ЖЕЛАЕМАЯ ДЛИНА ИНТЕРВАЛА НЕОПРЕДЕ-
C                   ЛЕННОСТИ КОНЕЧНОГО РЕЗУЛЬТАТА
C
C
C       ВЫХОДНАЯ ИНФОРМАЦИЯ...
C
C       ZEROIN АБСЦИССА, АППРОКСИМИРУЮЩАЯ НУЛЬ ФУНК-
C              ЦИИ F В ИНТЕРВАЛЕ AX, BX
C
C
C          БЕЗ ПРОВЕРКИ ПРЕДПОЛАГАЕТСЯ, ЧТО F(AX) И F(BX)
C       ИМЕЮТ ПРОТИВОПОЛОЖНЫЕ ЗНАКИ. ZEROIN ВЫЧИСЛЯЕТ
C       НУЛЬ X В ЗАДАННОМ ИНТЕРВАЛЕ AX, BX В ПРЕДЕЛАХ
C       ДОПУСКА НА ОШИБКУ 4*MACHEPS*ABS(X)+TOL, ГДЕ
C       MACHEPS—ОТНОСИТЕЛЬНАЯ МАШИННАЯ ТОЧНОСТЬ.
C          ЭТА ПОДПРОГРАММА-ФУНКЦИЯ ПРЕДСТАВЛЯЕТ СОБОЙ
C       СЛЕГКА МОДИФИЦИРОВАННУЮ ТРАНСЛЯЦИЮ АЛГОЛ
C       60—ПРОЦЕДУРЫ ZERO, ПРИВЕДЕННОЙ В КНИГЕ RICHARD
C       BRENT, ALGORITHMS FOR MINIMIZATION WITHOUT
C       DERIVATIVES, PRENTICE-HALL, INC. (1973).
C
      REAL A,B,C,D,E,EPS,FA,FB,FC,TOL1,XM,P,Q,R,S
C
C  ВЫЧИСЛИТЬ EPS, ОТНОСИТЕЛЬНУЮ МАШИННУЮ ТОЧНОСТЬ
C
      EPS=1.0
   10 EPS=EPS/2.0
      TOL1=1.0+EPS
      IF(TOL1 .GT. 1.0) GO TO 10
C
C  ПРИСВОЕНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ
C
      A=AX
      B=BX
      FA=F(A)
      FB=F(B)
C
C  НАЧАТЬ ШАГ
C
   20 C=A
      FC=FA
      D=B-A
      E=D
   30 IF(ABS(FC) .GE. ABS(FB)) GO TO 40
      A=B
      B=C
      C=A
      FA=FB
```

```
      FB=FC
      FC=FA
C
C   ПРОВЕРКА СХОДИМОСТИ
C
   40 TOL1=2.0*EPS*ABS(B)+0.5*TOL
      XM= .5*(C-B)
      IF(ABS(XM) .LE. TOL1)GO TO 90
      IF(FB.EQ.0.0)GO TO 90
C
C   НЕОБХОДИМА ЛИ БИСЕКЦИЯ
C
      IF(ABS(E) .LT. TOL1)GO TO 70
      IF(ABS(FA) .LE. ABS(FB))GO TO 70
C
C     ВОЗМОЖНА ЛИ КВАДРАТИЧНАЯ ИНТЕРПОЛЯЦИЯ
C
      IF(A .NE. C)GO TO 50
C
C   ЛИНЕЙНАЯ ИНТЕРПОЛЯЦИЯ
C
      S=FB/FA
      P=2.0*XM*S
      Q=1.0-S
      GO TO 60
C
C   ОБРАТНАЯ КВАДРАТИЧНАЯ ИНТЕРПОЛЯЦИЯ
C
   50 Q=FA/FC
      R=FB/FC
      S=FB/FA
      P=S*(2.0*XM*Q*(Q-R)-(B-A)*(R-1.0))
      Q=(Q-1.0)*(R-1.0)*(S-1.0)
C
C     ВЫБРАТЬ ЗНАКИ
C
   60 IF(P .GT. 0.0)Q=-Q
      P=ABS(P)
C
C   ПРИЕМЛЕМА ЛИ ИНТЕРПОЛЯЦИЯ
C
      IF((2.0*P) .GE. (3.0*XM*Q-ABS(TOL1*Q)))GO TO 70
      IF(P .GE. ABS(0.5*E*Q))GO TO 70
      E=D
      D=P/Q
      GO TO 80
C
C   БИСЕКЦИЯ
C
   70 D=XM
      E=D
C
C   ЗАВЕРШИТЬ ШАГ
C
   80 A=B
      FA=FB
```

```
      IF(ABS(D) .GT. TOL1)B=B+D
      IF(ABS(D) .LE. TOL1)B=B+SIGN(TOL1,XM)
      FB=F(B)
      IF((FB*(FC/ABS(FC))) .GT. 0.0)GO TO 20
      GO TO 30
C
C  КОНЧЕНО
C
   90 ZEROIN=B
      RETURN
      END
```

7.3. Трансцендентные уравнения— комплексные корни

Обычно аналитические функции имеют не только действительные, но и комплексные корни. Могут ли методы, обсуждавшиеся в § 7.1 и 7.2, использоваться для нахождения комплексных нулей?

Метод бисекции опирается на то, что непрерывная функция, меняющая знак в интервале, имеет нуль внутри этого интервала. Эту идею нелегко обобщить для локализации комплексных нулей аналитических функций. Одна родственная идея — это *принцип аргумента*: предположим, что f аналитична в области R комплексной плоскости, и пусть C — простая замкнутая кривая в R. Предположим, что когда z описывает контур C, $f(z)$ ровно один раз обходит начало координат. Тогда f имеет ровно один нуль внутри C.

Было сделано множество попыток непосредственно применить этот принцип к локализации нулей f внутри области, но ни одна не оказалась особенно успешной. Основная трудность состоит в необходимости большого количества вычислений $f(z)$, чтобы увидеть, заключает ли кривая $\{f(z) : z \in C\}$ внутри себя начало координат. Лемер (D. H. Lehmer) использовал усложненный вариант этой идеи для локализации нулей полиномов. Но для функций f общего вида, по-видимому, нет какого-либо хотя и медленного, но гарантированного аналога метода бисекции. Поэтому на практике пользуются другими методами.

Методы Ньютона и секущих можно использовать в комплексной плоскости без каких-либо изменений в теоремах. Как и для действительных корней, необходимо близко подойти к нулю, прежде чем может начаться быстрая сходимость. Зачастую само происхождение задачи подсказывает приближенное расположение нулей, и этой информацией можно воспользоваться для начала итераций. При отсутствии таких указаний метод Ньютона часто применяют, исходя из некоего случайного комплексного начального приближения z_0. В принципе для последовательности $\{z_k\}$ возможны следующие три варианта поведения:

1. Она расходится к ∞.
2. Она сходится к нулю r.
3. Она ведет себя хаотически.

Для некоторых функций в случае 3 последовательность $\{z_k\}$ в конечном счете переходит в состояние, в котором точки почти зацикливаются с периодом 2,3 или любым другим конечным периодом.

Если вам нужен нуль r функции f, то вы можете попробовать применить метод Ньютона, снабженный каким-либо приспособлением, препятствующим слишком большому удалению z_k от z_{k-1} в попытке устранить расходимость к ∞. Затем остается надеяться, что метод сойдется к нулю. Если после достаточно большого числа итераций сходимости нет, то можно начать снова с каким-нибудь другим z_0. Практика показывает, что часто таким образом нуль находится.

Пусть был найден один нуль r_1 и нужен другой, тогда необходим способ воспрепятствовать возвращению последующих итераций к r_1. Часто бывает полезно продолжать итерационный процесс с новой функцией

$$f_1(z)=\frac{f(z)}{z-r_1},$$

где деление выполняется лишь для *числовых значений* $f(z)$ и $z-r_1$. Основной член в методе Ньютона — отношение $f(z)/f'(z)$; обратная к нему величина есть $f'(z)/f(z)=(d/dz)\ln f(z)$. Поэтому

$$\frac{f_1'(z)}{f_1(z)}=\frac{d}{dz}\ln f(z)-\frac{d}{dz}\ln(z-r_1)=\frac{f'(z)}{f(z)}-\frac{1}{z-r_1}.$$

После того как были найдены s корней, алгоритм Ньютона применяют к функции $f(z)/\Pi_{k=1}^{s}(z-r_k)$ и по-прежнему логарифмическое дифференцирование в принципе производится легко.

Если прямое применение алгоритмов Ньютона или секущих не приводит к успеху, то, видимо, следует предпослать им какой-нибудь метод приближения к нулю. Одна из возможностей — использовать алгоритм для минимизации действительной функции двух действительных переменных x и y в применении к функции

$$\varphi(x,\ y)=|f(x+iy)|^2.$$

Легко доказать, что для аналитической функции f все локальные минимумы функции $\varphi(x,\ y)$ соответствуют случаю $\varphi=0$. Методы минимизации обсуждаются в гл. 8.

Широко используемый метод для нахождения нулей аналитической функции комплексного переменного принадлежит Мюллеру. Это обобщение метода секущих с двух интерполяционных точек на три Мюллер начинает с произвольных чисел z_1, z_2 и

z_3. В общем случае имеются три точки z_{n-2}, z_{n-1} и z_n и соответствующие значения функции, которые обозначим через f_{n-2}, f_{n-1} и f_n. Мюллер составляет (единственную) квадратичную функцию от z, интерполирующую три точки (z_i, f_i), где $i=n-2$, $n-1$, n. В качестве z_{n+1} затем берется тот из двух нулей квадратичной функции, который ближе к z_n. Далее это же повторяется с z_{n-1}, z_n и z_{n+1}.

Исходной мотивировкой Мюллера для этого метода было то обстоятельство, что использование квадратичного многочлена позволяет процессу вычисления нуля переходить от первоначальных действительных итераций к последующим комплексным в отличие от методов Ньютона и секущих. Впоследствие он обнаружил, что метод хорошо работает и при нахождении действительных корней.

7.4. Нули полиномов

Вследствие того что полиномы являются весьма специальными функциями, можно понять наличие множества алгоритмов для вычисления их нулей. Некоторые из них принадлежат к числу старейших алгоритмов численного анализа. С минувших столетий ведут свое происхождение методы Горнера, Греффе и Бернулли; в вычислительную эпоху созданы методы Рутисхаузера, Лемера, Лина, Бэрстоу (Bairstow), Бэрайсса (Bareiss) и другие.

С математической точки зрения большой теоретический интерес представляет создание алгоритмов, для которых можно было бы доказать гарантированную сходимость, хотя бы в обманчивых рамках действительной арифметики. С точки зрения практической имеется потребность в алгоритмах, которые бы почти всегда работали и вычисляли каждый нуль за малые доли секунды, если полином умеренного порядка, а машина достаточно мощная.

По сравнению с трансцендентными функциями полиномы имеют то преимущество, что наперед известно точное число их корней и, следовательно, известно, когда нужно остановить алгоритм.

Читатель должен припомнить крайнюю неустойчивость корней некоторых полиномов как функций от их коэффициентов (§ 2.5). Это значит, что многие задачи, включающие в себя нахождение нулей полиномов, требуют либо предельно точного вычисления коэффициентов, либо совершенно иного подхода. Например, 20 или 30 лет назад считалось, что наиболее естественный путь вычисления собственных значений матрицы состоит в построении, а затем решении характеристического уравнения.

В настоящее время известны гораздо лучшие способы получения собственных значений — способы, которые не требуют проведения вычислений в арифметике очень высокой точности (гл. 9). Вероятно, и многие другие задачи, где используются полиномы, должны решаться иначе. Таким образом, возможно, что нахождение нулей полиномов уже не является столь важной частью научных вычислений, как это было до сих пор.

Авторы имеют лишь небольшой опыт работы с фортран-программами для этой задачи; однако полагают, что метод Мюллера вполне хорош во многих приложениях. Быстрая и надежная фортран-программа для нахождения нулей полиномов с комплексными коэффициентами приведена в статье Дженкинс, Трауб (1972).

7.5. Нелинейные системы уравнений

Очень распространенной вычислительной задачей является нахождение некоторых или всех решений системы из n нелинейных алгебраических или трансцендентных уравнений с n неизвестными. Обозначая через $\mathbf{x}$ вектор-столбец $(x_1, \ldots, x_n)^T$, можно записать уравнения в виде: $f_1(\mathbf{x})=0$, $f_2(\mathbf{x})=0, \ldots, f_n(\mathbf{x})=0$. Или, вводя для вектора-столбца функций $(f_1, \ldots, f_n)^T$ обозначение $\mathbf{f}$, можем записать всю систему в компактной форме $\mathbf{f}(\mathbf{x})=\mathbf{0}$. Подобные системы уравнений могут возникать непосредственно, например при конструировании нелинейных физических систем, а могут получаться опосредованно. К примеру, пытаясь минимизировать функцию $G(\mathbf{x})$, можно попробовать найти те точки, где градиент этой функции равен нулю. Полагая $\mathbf{f}=\operatorname{grad} G$, получаем нелинейную систему.

Один из подходов к решению системы $\mathbf{f}(\mathbf{x})=\mathbf{0}$ состоит в обобщении на размерность n итерационных процессов, полезных при решении единственного уравнения (случай $n=1$). Если могут быть вычислены все частные производные функций f_i по переменным x_j, то можно применить процесс Ньютона. Пусть $J(\mathbf{x})$ обозначает матрицу Якоби; ее элемент (i, j) есть значение производной $\partial f_i/\partial x_j$ в точке $\mathbf{x}$. Как и в одномерном случае, метод Ньютона начинает с произвольного $\mathbf{x}$, скажем $\mathbf{x}^0$. Далее, функцию $\mathbf{f}$ линеаризуют в точке $\mathbf{x}^0$, разлагая ее в ряд Тейлора и удерживая лишь члены нулевой и первой степени:

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^0) + J(\mathbf{x}^0)(\mathbf{x}-\mathbf{x}^0) + \ldots .$$

Линейное приближение к $\mathbf{f}$ около $\mathbf{x}^0$ задается, следовательно, формулой

$$\mathbf{L}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^0) + J^0(\mathbf{x}-\mathbf{x}^0),$$

где J^0 — сокращенное обозначение для $J(\mathbf{x}^0)$.

Чтобы найти следующее приближение $\mathbf{x}$ к решению системы $\mathbf{f}(\mathbf{x})=\mathbf{0}$, решают уравнение

$$\mathbf{f}(\mathbf{x}^0)+J^0(\mathbf{x}^1-\mathbf{x}^0)=\mathbf{0}.$$

Разумеется, решение можно записать в форме

$$\mathbf{x}^1=\mathbf{x}^0-(J^0)^{-1}\mathbf{f}(\mathbf{x}^0);$$

в этом виде формула сильно напоминает одномерную форму метода Ньютона. Однако для большинства систем из n линейных уравнений с n неизвестными вычисление обратной матрицы $(J^0)^{-1}$ не является ни необходимым, ни даже желательным; предпочтительней как раз решать линейную систему относительно поправки $\mathbf{x}^1-\mathbf{x}^0$.

В общем случае, имея $\mathbf{x}^k$, находят $\mathbf{x}^{k+1}$ прибавлением к $\mathbf{x}^k$ поправки $\mathbf{x}^{k+1}-\mathbf{x}^k$, полученной решением линейной системы

$$J^k(\mathbf{x}^{k+1}-\mathbf{x}^k)=-\mathbf{f}(\mathbf{x}^k).$$

Для такого итерационного процесса можно доказать теорему, аналогичную первой теореме § 7.1. Сформулируем ее неформально. Пусть r — решение системы $\mathbf{f}(\mathbf{x})=\mathbf{0}$, такое, что $J(r)$ не вырождена и вторые частные производные функции $\mathbf{f}$ непрерывны вблизи r. Тогда, если $\mathbf{x}^0$ достаточно близко к r, то ньютоновы итерации будут сходиться. Более того, для $\mathbf{e}^k=\mathbf{x}^k-\mathbf{r}$ при $k\to\infty$ отношение $\|\mathbf{e}^{k+1}\|/\|\mathbf{e}^k\|^2$ ограничено. Таким образом, сходимость будет порядка 2.

Как и в одномерном случае, основная проблема в том, чтобы подойти достаточно близко к желаемому корню r, чтобы могла начаться быстрая сходимость. На практике можно подчас, итерируя с мужеством и оптимизмом, найти корень без особых трудностей.

Еще одно важное соображение — трудность и трудоемкость вычисления матрицы Якоби. В одномерном случае можно не без оснований предполагать, что стоимость вычисления $f'(x)$ примерно та же, что и для $f(x)$, хотя есть, конечно, задачи, для которых это неверно. При n измерениях $\mathbf{f}(\mathbf{x})$ становится вектором, а $\mathbf{f}'(\mathbf{x})$ превращается в матрицу Якоби. Поэтому вычисление $\mathbf{f}'(\mathbf{x})$ зачастую во много раз более трудоемко, чем вычисление $\mathbf{f}(\mathbf{x})$.

Попытки избежать явного использования якобианов претворились в методы многих различных типов. Прямое обобщение метода секущих неудовлетворительно, поскольку приближения к $J(\mathbf{x})$, получаемые из n-мерных аналогов секущей, часто оказываются вырожденными. См. статью Грэг, Стьюарт (1976). Более удачны методы, называемые *квазиньютоновыми*. Они генерируют приближения к $J(x)$, очень грубые в начале итераций; однако их точность по мере продолжения итераций все время возрастает.

Обычно эти методы описывают в применении к задачам многомерной оптимизации, которую мы рассмотрим в § 8.3.

Обсуждение процесса Ньютона содержится в цитированной выше статье Трауба (Рэлстон, Уилф (1967)). Тщательная формулировка и доказательство теоремы сходимости даны в книге Островский (1966). Книгу Ортега, Рейнболдт (1970) можно считать полным справочником по большинству теоретических результатов.

УПРАЖНЕНИЯ

7.1. С помощью подпрограммы ZEROIN найдите значение x, такое, что

$$\mathrm{erf}\,(x)-0.5=0.0.$$

Для вычисления $\mathrm{erf}\,(x)$ используйте любую доступную подпрограмму или же см. упр. 5.1.

7.2. С помощью подпрограммы ZEROIN найдите десять наименьших положительных значений x, для которых прямая $y=x$ пересекает график кривой $y=\mathrm{tg}\,x$.

7.3. Пусть $f(x)=x(x-1)^5$, и пусть A=—0.50 и B=0.98. Примените ZEROIN и выведите на печать значения A, B и C для каждой итерации. Выведите также для каждого шага указание, выполнялся ли этот шаг бисекцией, методом секущих или обратной квадратичной интерполяцией.

7.4. Используйте метод Ньютона для вычисления комплексного корня нашего исторического примера

$$x^3-2x-5=0.$$

7.5. Существует классический метод Кардано для решения кубических уравнений. Кубическое уравнение

$$x^3+ax^2+bx+c=0$$

преобразуют к приведенной форме

$$y^3+py+q=0$$

подстановкой $x=y-a/3$. Коэффициенты приведенной формы:

$$p=b-\frac{a^2}{3},$$

$$q=c-\frac{ab}{3}+2\left(\frac{a}{3}\right)^3.$$

Один действительный корень приведенной формы можно найти по формулам:

$$s=\left[\left(\frac{p}{3}\right)^3+\left(\frac{q}{2}\right)^2\right]^{1/2},$$

$$y_1=\left[-\frac{q}{2}+s\right]^{1/3}+\left[-\frac{q}{2}-s\right]^{1/3},$$

после чего действительный корень исходного уравнения вычисляется как

$$x_1 = y_1 - \frac{a}{3}.$$

Можно найти аналогичные формулы для других двух корней или же разделить исходное уравнение на $x - x_1$ и решить полученное квадратное уравнение.

(а) Примените метод Кардано к нахождению действительных корней уравнения

$$x^3 + 3x^2 + \alpha^2 x + 3\alpha^2 = 0$$

для различных значений α. Исследуйте потерю точности вследствие ошибок округлений при больших α, например α, имеющих порядок величины, обратной к машинному эпсилон.

(б) Примените к тому же уравнению для тех же значений α метод Ньютона. Исследуйте влияние ошибок округлений, а также выбора начального приближения.

7.6. Какой результат будет получен следующей программой? Объясните, почему.

```
      FUNCTION F(X)
      IF (X .EQ. 0.) Y = -1.
      IF (X .GT. 0.) Y = 9.
      WRITE(6,1) X,Y
1     FORMAT(F15.10, F5.0)
      F = Y
      RETURN
      END

      EXTERNAL F
      A = 0.
      B = 1.
      TOL = 1.E-6
      Z = ZEROIN(A, B, F, TOL)
      STOP
      END
```

7.7. В этой задаче речь идет о распространении волн в среде с переменной скоростью распространения. Конкретная постановка связана с подводным распространением звука, но используемые методы применимы и в других ситуациях. Здесь комбинируются выравнивание данных, решение обыкновенного дифференциального уравнения и вычисление нулей. Используются подпрограммы SPLINE, RKF45 и ZEROIN. Задача извлечена из статьи Моулер, Соломон (1970).

Скорость звука в океанской воде зависит от давления, температуры и солености; все эти параметры меняются довольно сложно при изменении глубины. Пусть z — глубина в футах под поверхностью океана (так что ось z направлена вниз), и пусть $c(z)$ — скорость звука на глубине z. Будем пренебрегать изменениями скорости звука, наблюдаемыми в горизонтальных направлениях. Можно измерить $c(z)$ для дискретных значений z; вот типичная таблица, полученная этим путем:

z (фут)	$c(z)$ (фут/сек)
0	5 042
500	4 995
1 000	4 948
1 500	4 887
2 000	4 868
2 500	4 863
3 000	4 865
3 500	4 869
4 000	4 875
5 000	4 875
6 000	4 887
7 000	4 905
8 000	4 918
9 000	4 933
10 000	4 949
11 000	4 973
12 000	4 991

Чтобы получить значения $c(z)$ между заданными точками, можно применить кубическую сплайн-интерполяцию. Нужно также вычислять $c'(z)=dc(z)/dz$, поэтому первый этап этой задачи будет такой:

(а) Используя подпрограмму SPLINE, найдите коэффициенты кубического сплайна, интерполирующего указанные в таблице точки. Модифицируйте подпрограмму SEVAL таким образом, чтобы она выдавала в заданной точке значения как сплайна, так и его производной.

Поскольку скорость звука изменяется с глубиной, звуковые лучи [1] будут распространяться по криволинейным путям. Этот эффект есть непрерывный аналог хорошо известного явления — преломления световых лучей на границе раздела воздух — вода. Основное уравнение — это непрерывный вариант закона Снелла. Пусть x обозначает горизонтальное (радиальное) расстояние от источника звука, измеряемое в футах, а $z(x)$ — глубину данного луча на расстоянии x. Через $\theta=\theta(x)$ обозначим угол между горизонтальной прямой и касательной к лучу в точке x; тогда

$$\operatorname{tg}\theta=\frac{dz}{dx}.$$

Закон Снелла можно записать соотношением

$$\frac{\cos\theta}{c(z)}=\text{постоянная}.$$

Эти два уравнения совместно дают обыкновенное дифференциальное уравнение, которое, по-видимому, определяет z как функцию от x. Однако оказывается, что у этого уравнения нарушается единственность решений в тех точках, где луч становится горизонтальным. Чтобы исключить ненужные решения, продифференцируем оба уравнения по x; комбинируя результаты, получим

[1] Распространение звука рассматривается здесь с позиций геометрической акустики.— *Прим. перев.*

уравнение второго порядка

$$\frac{d^2z}{dx^2} = -\frac{c'(z)}{A^2c(z)^3},$$

где A — константа, входящая в закон Снелла. Эту константу и начальные условия удобно выразить через глубину z_0 источника и угол θ_0 выхода луча из источника. Получаем

$$z(0) = z_0,$$

$$\frac{dz}{dx}(0) = \operatorname{tg}\theta_0,$$

$$A^2 = \left(\frac{\cos\theta_0}{c(z_0)}\right)^2.$$

Это приводит нас ко второй части задачи.

(б) Используя подпрограмму RKF45, найдите луч с начальными условиями z_0=2000 фут и θ_0=5.4°. Проследите траекторию луча на протяжении 24 морских миль, выводя на печать значения глубины с интервалом в 1 милю. Считайте, что в одной морской миле содержится 6076 футов; не забудьте, что тригонометрические функции ФОРТРАНа предполагают, что аргумент задан в радианах. Вы должны обнаружить, что глубина на расстоянии 24 миль близка к 3000 футов.

Предположим теперь, что источник звука с глубины 2000 футов «вещает» на приемник, находящийся на расстоянии 24 миль [1] и на глубине 3000 футов. Проведенные выше вычисления показывают, что один из лучей от источника к приемнику выходит из источника под углом, близким к 5.4°. Сколько имеется других лучей с теми же концами? Пусть x_f=24 морские мили. При изменении θ_0 меняется и $z(x_f)$. Нас интересуют значения θ_0, для которых $z(x_f)$=3000.

(в) Напишите подпрограмму-функцию F(THETA), которая прослеживает траекторию луча с начальным углом THETA и выдает значение $z(x_f)$—3000. Выведите на печать таблицу значений этой функции для THETA, меняющегося в интервале от —10° до 10°. Так как вычисление этой функции очень дорогостоящее, то шаг, принятый в таблице, будет зависеть от количества машинного времени, которым вы располагаете, и эффективности вашей программы.

(г) Используя ZEROIN с начальными значениями, полученными из задания в), найдите лучи, которые проходят через приемник (или на минимальном возможном расстоянии от него).

(д) Предположим, что дно океана находится на глубине 12 000 футов, а поверхности соответствует уровень 0 футов. Что произойдет, если θ_0 больше 10°? А что если θ_0 меньше —10°? Существуют ли лучи с начальными углами в этих областях?

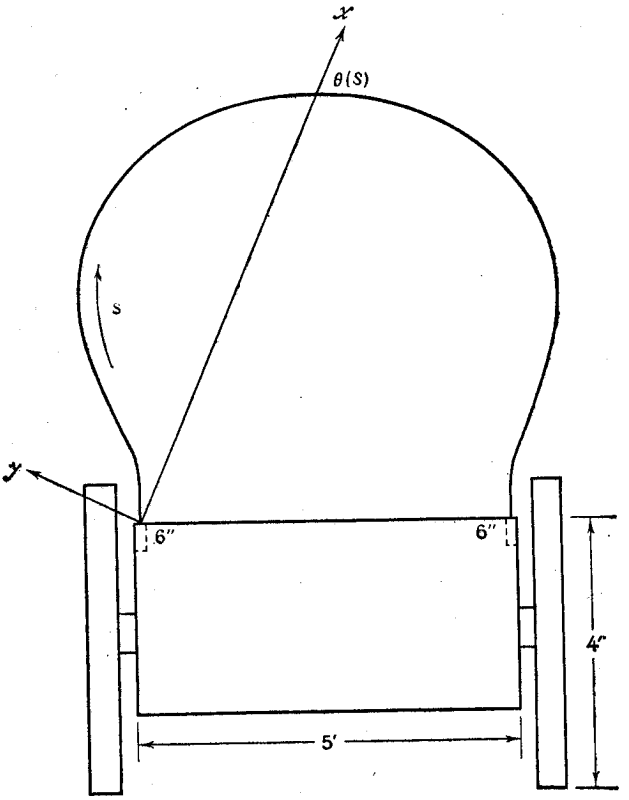
(е) Обсудите влияние выбора границ для погрешностей в RKF45 и ZEROIN на точность и цену полученных решений. Рассмотрите возможность использования других границ в заданиях в) и г).

7.8. Предположим, что имеется «непокрытый» вагон, показанный на сопровождающем рисунке, и вы должны соорудить покрытие для него. Задача заключается в том, чтобы выбрать длину тонких деревянных реек (ребер), на которые натягивается брезент. На местном складе имеются в наличии рейки следующих длин: 14, 16, 18, 20, 22 и 24 фута. Вы хотели бы максимально увеличить вагон; однако начальник товарного состава предупредил вас, что вагоны, превышающие по высоте 11 футов, случалось, опрокидывались. Какую из указанных выше длин вы выбрали бы, чтобы добиться максимальной высоты

[1] Имеется в виду — по горизонтали.— *Прим. перев.*

вагона, не превышающей 11 футов? Заметьте, что каждый конец рейки закрепляется в 6-дюймовом жестком вертикальном пазе.

Указания: тонкие деревянные рейки при столь сильных прогибах подчиняются закону Гука линейной упругости и хорошо моделируются *эластиками*, или *упругими линиями*. Известно, что для такой линии кривизна $\varkappa$ удов-



летворяет нелинейному дифференциальному уравнению

$$\frac{d^2\varkappa}{ds^2} = -\frac{1}{2}\varkappa^3,$$

где s — длина кривой, измеряемая вдоль эластики. Вспомним, что, согласно анализу,

$$\varkappa = \frac{d\theta}{ds},$$

где θ — угол, образуемый кривой с какой-нибудь фиксированной прямой. Если принять эту прямую за ось x декартовой системы координат, то

$$\frac{dx}{ds} = \cos\theta$$

и

$$\frac{dy}{ds}=\sin\theta.$$

Вследствие симметрии относительно вертикальной оси, данная задача превращается в краевую задачу, которая может быть решена методом стрельбы.

Удобно выбрать координатную систему так, как показано на рисунке. Тот факт, что концы рейки зафиксированы в жестких пазах, дает краевое условие $\varkappa(0)=0$. Симметрия требует, чтобы верх (центр) рейки был горизонтальным, так что $\theta(S)+\theta(0)=\frac{\pi}{2}$. Для стрельбы имеются два параметра: $\theta(0)$ и $(d\varkappa/ds)(0)$. Поэтому нужны две копии подпрограммы ZEROIN, и одну из них придется переименовать.

Можно избежать повторных вызовов библиотечных функций SIN и COS, заметив, что

$$\frac{d}{ds}(\sin\theta)=\cos\theta\frac{d\theta}{ds}=\varkappa\cos\theta,$$

$$\frac{d}{ds}(\cos\theta)=-\sin\theta\frac{d\theta}{ds}=-\varkappa\sin\theta.$$

Таким образом, для вычисления синусов и косинусов можно интегрировать вместе с другими и эти два дополнительных дифференциальных уравнения.

Цель этого задания — заставить вас удивиться тому, что люди смогли заселить запад, не имея компьютеров.

7.9. Решите следующую нелинейную краевую задачу относительно $y(x)$ на интервале $0\leqslant x\leqslant 1$:

$$y''=y^2-1,\quad y(0)=0,\quad y(1)=1.$$

а) Примените метод стрельбы, описанный в § 6.6, используя ZEROIN и RKF45.

б) Изучив § 7.5, попробуйте решить задачу иначе. Разбейте заданный интервал на n равных подынтервалов:

$$0=x_0<x_1<x_2<\ldots<x_{n-1}<x_n=1.$$

Замените дифференциальное уравнение разностными уравнениями с $n-1$ неизвестными $y_1, y_2, \ldots, y_{n-1}$, где y_i аппроксимирует $y(x_i)$:

$$y_{i+1}-2y_i+y_{i-1}=h^2(y_i^2-1),\quad i=1,\ 2,\ \ldots,\ n-1,$$
$$y_0=0,\ y_n=1.$$

Решите эту *нелинейную* трехдиагональную систему, скажем, для $n=50$.

в) В качестве третьего возможного подхода, попробуйте следующее. Заметим, что $y''=y^2-1$ можно переписать в виде

$$\frac{d}{dx}\left(\frac{(y')^2}{2}-\frac{y^3}{3}+y\right)=0.$$

Отсюда

$$\frac{(y')^2}{2}-\frac{y^3}{3}+y=c$$

для некоторой константы c. Поскольку $y(0)=0$, то $y'(0)=\sqrt{2c}$. Следовательно, если бы мы могли вычислить c, то краевая задача превратилась бы в задачу

Коши. Интегрирование полученного уравнения дает

$$x = \int_0^y \frac{dy}{\sqrt{2}\left(c + \frac{y^3}{3} - y\right)^{1/2}}.$$

Используйте подпрограммы ZEROIN и QUANC8, чтобы найти c из уравнения

$$1 = \int_0^1 \frac{dy}{\sqrt{2}\left(c + \frac{y^3}{3} - y\right)^{1/2}},$$

а затем с помощью RKF45 постройте искомое решение.

8. ОПТИМИЗАЦИЯ

Часто встречающаяся в научных вычислениях задача состоит в определении максимума или минимума (и соответствующих аргументов) действительной функции $f(x_1, \ldots, x_n)$ от n действительных переменных на множестве S n-мерного пространства. Слово *оптимизация* означает либо минимизацию, либо максимизацию функции. Иногда S совпадает со всем n-мерным пространством; в этом случае задача оптимизации называется *безусловной*. В противном случае задача имеет *ограничения*, именно условия, определяющие множество S. Обычно S определяется совокупностью нелинейных функций, удовлетворяющих уравнениям или неравенствам. Другими словами, точка $\mathbf{x}$ принадлежит S тогда и только тогда, когда $\mathbf{x}$ удовлетворяет неравенствам

$$g_i(\mathbf{x}) \geqslant 0, \quad i = 1, \ldots, n,$$

где g_i — заданные функции от $\mathbf{x}$.

Задачи с ограничениями, где g_i — нелинейные функции, обычно гораздо труднее решать, чем соответствующие безусловные задачи. Если функция f и все ограничения g_i являются линейными функциями, то говорят о задаче *линейного программирования*. Можно показать, что решение лежит в вершине выпуклого многогранника, определяемого ограничениями в n-мерном пространстве. Обычный метод решения состоит в поиске нужной вершины, осуществляемом перемещением от очередной вершины к смежной. Трудные проблемы линейного программирования по существу связаны с решением задач очень высокого порядка n, которое приводит к разреженным матричным задачам. В основе трудности таких задач лежит комбинаторная сложность общего n-мерного многогранника. Если функция f либо какое-нибудь из ограничений нелинейно, то говорят о задаче *нелинейного программирования*. Задачи линейного и нелинейного программирования выходят за рамки этой книги. Интересующийся читатель должен обратиться, например, к книгам Орчард-Хейс (1968) или Данциг

(1963). Мы ограничимся обсуждением задач либо безусловных, либо относящихся к интервалу.

Оптимизационные задачи часто возникают непосредственно в контексте поисков наилучшей конструкции какого-либо объекта. Например, можно искать значения некоторых n параметров системы, которые минимизируют ее стоимость, выраженную как функция этих параметров.

Иногда оптимизационные задачи возникают опосредованно как средство решения каких-то других задач. Стандартный пример — сведение задачи решения системы n нелинейных уравнений

$$f_1(x_1, \ldots, x_n) = 0,\ f_2(x_1, \ldots, x_n) = 0,\ \ldots,\ f_n(x_1, \ldots, x_n) = 0$$

к минимизации функции

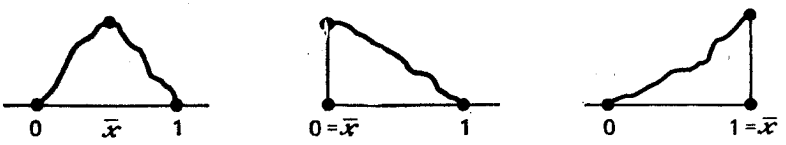
$$g(x_1, \ldots, x_n) = \sum_{i=1}^{n} |f_i(x_1, \ldots, x_n)|^2.$$

Очевидно, что точки минимума g с нулевыми значениями есть в точности решения системы. (Могут быть также посторонние ненулевые локальные минимумы.) Однако эту задачу обычно не решают общими алгоритмами минимизации, поскольку в данном случае можно извлечь особую выгоду из типа минимизируемой функции.

Как и в гладком одномерном анализе, легче отыскать относительные или локальные минимумы функции, чем найти ее абсолютный или глобальный минимум во всей области. И в самом деле, в настоящее время не существует практичных алгоритмов для вычисления глобального минимума для $n>2$ или 3. Даже чтобы найти приближенное значение глобального минимума в n-мерном единичном кубе, потребовалось бы вычислять функцию в точках густой решетки, помещенной в куб, при условии априорного знания о том, что функция достаточно гладкая, чтобы не иметь выбросов между точками решетки. Но густая решетка в n-мерном кубе содержит слишком много точек для того, чтобы ее можно было обрабатывать. Если, к примеру, $n=10$, то беря 10 абсцисс на каждом измерении, получим всего 10^{10} точек. Таким образом, на практике единственный способ найти глобальный минимум состоит в том, чтобы получить из самой задачи информацию о его местоположении, а затем искать локальный минимум. Поэтому мы сосредоточим свое внимание на нахождении локальных минимумов и максимумов. Методы вычисления минимумов тривиально переносятся на задачу максимизации (поскольку минимумы f есть максимумы для $-f$), и мы будем говорить попеременно о той и другой задаче.

8.1. Одномерная оптимизация

Предположим, что f — действительная функция, определенная на $[0, 1]$. Предположим, далее, что имеется единственное значение $\bar{x}$, такое, что $f(\bar{x})$ — максимум $f(x)$ на $[0, 1]$, и что $f(x)$ строго возрастает для $x \leqslant \bar{x}$ и строго убывает для $\bar{x} \leqslant x$. Такая функция называется *унимодальной:* для ее графика имеются три возможные формы, показанные на рис. 8.1. Заметим, что унимодальная функция не обязана быть гладкой или даже непрерывной.



Из предположений немедленно следует, что для любых точек интервала x_1, x_2, таких, что $x_1 < x_2 \leqslant \bar{x}$, справедливо $f(x_1) < f(x_2)$. Аналогично, если $\bar{x} \leqslant x_1 < x_2$, то $f(x_1) > f(x_2)$. Обратно, если $x_1 < x_2$ и $f(x_1) < f(x_2)$, то $x_1 \leqslant \bar{x} \leqslant 1$, а если $f(x_1) > f(x_2)$, то $0 \leqslant \bar{x} \leqslant x_2$. [Конечно, если $f(x_1) = f(x_2)$, то мы получаем дополнительную информацию: $x_1 \leqslant \bar{x} \leqslant x_2$, но нам не придется этим пользоваться.] Задача состоит в том, чтобы найти множество абсцисс $x_1, x_2, \ldots x_k$, в которых вычисляется функция, такое, что оптимальное значение f лежит при некотором i в интервале $x_{i-1} \leqslant \bar{x} \leqslant x_{i+1}$. Такой интервал называется *интервалом неопределенности.*

Алгоритм выбора абсцисс $x_i (i = 1, \ldots, k)$ называется *планом поиска.* Если известно только то, что f унимодальна, то какова оптимальная стратегия для нахождения $\bar{x}$? При заданном количестве вычислений функции *оптимальным* планом поиска будет тот, который приводит к наименьшему интервалу неопределенности. В некоторых ситуациях, таких, как физический эксперимент или счет на параллельных процессорах, приходится одновременно вычислять функцию f во всех точках x_i. Можно показать, что оптимальный план поиска выражается формулами

$$x_i = \frac{(1+\varepsilon)\lfloor (i+1)/2 \rfloor}{(k/2)+1} - \left(\left\lfloor \frac{i+1}{2} \right\rfloor - \left\lfloor \frac{i}{2} \right\rfloor \right) \varepsilon,$$

где $\lfloor y \rfloor$ — целая часть числа y. При k вычислениях функции получается интервал неопределенности длины

$$\frac{1+\varepsilon}{(k/2)+1}.$$

В этих формулах ε выбирается из условия, чтобы $f(x) \neq f(x+\varepsilon)$ для любой точки $x \in [0, 1]$, находящейся от $\bar{x}$ на расстоянии, не меньшем ε. Более подробное обсуждение методов одновременного поиска можно найти в книге Вильде (1964). В остальной части этой главы мы будем считать, что значения функции вычисляются последовательно.

Предположим, что разрешено последовательно вычислять функцию k раз, где $k>1$ — заданное число. Как использовать эти вычисления так, чтобы заключить x в наименьший возможный интервал неопределенности?

Соответствующая теория была начата работами Кифера (J. Kiefer) в первой половине пятидесятых годов. Алгоритм оптимальной стратегии последовательно строит k тестовых точек, которые мы предпочитаем занумеровать в обратном порядке x_k, $x_{k-1}, \ldots, x_2$, x_1. На первом этапе одновременно выбираются две точки, x_k и x_{k-1}, причем $x_{k-1}<x_k$. Если $f(x_{k-1}) \leqslant f(x_k)$, то интервал неопределенности сужается до $[x_{k-1}, 1]$, и *у нас уже есть опорное значение $f(x_k)$ для последующего уточнения интервала неопределенности*. С другой стороны, если $f(x_{k-1}) \geqslant f(x_k)$, то интервал неопределенности сужается до $[0, x_k]$, и мы уже знаем $f(x_{k-1})$. Вот ключ к оптимальному выбору x_{k-1} и x_k: обеспечить, чтобы тестовая точка, унаследованная суженным интервалом неопределенности, находилась в оптимальном для последующего поиска положении.

Полезно определить *координату r точки x относительно интервала* $[a, b]$ как число, равное $(x-a)/(b-a)$. Таким образом, a имеет по отношению к интервалу $[a, b]$ координату 0, b — координату 1, а средняя точка — координату $\frac{1}{2}$.

Пусть $F_0=F_1=1$, $F_2=2$, $F_3=3$, $F_4=5$, $F_5=8, \ldots$, $F_k=F_{k-1}+F_{k-2}$. Числа F_i — это знаменитые числа Фибоначчи. Оптимальная стратегия последовательного поиска максимума называется *поиском Фибоначчи*, поскольку она тесно связана с этими числами. При оптимальной стратегии выбирают $x_{k-1}=F_{k-2}/F_k$ и $x_k=F_{k-1}/F_k$. Какой бы из интервалов $[0, x_k]$ или $[x_{k-1}, 1]$ не стал суженным интервалом неопределенности, унаследованная точка будет иметь по отношению к новому интервалу одну из двух следующих координат: F_{k-3}/F_{k-1} или F_{k-2}/F_{k-1}. Тогда в качестве x_{k-2} выбирается точка, имеющая по отношению к новому интервалу другую из этих координат. Используя $f(x_{k-2})$ и значение функции, унаследованное от прежнего интервала, можно еще сократить интервал неопределенности и передать в наследство одно значение функции.

На последнем шаге мы придем к некоторому интервалу неопределенности $[a, b]$, причем средняя точка будет унаследованной от предыдущего шага. Тогда в качестве x_1 выбирается точка

с относительной координатой $\frac{1}{2}+\varepsilon$, и окончательным интервалом неопределенности будет либо $[0, \frac{1}{2}+\varepsilon]$, либо $[\frac{1}{2}, 1]$ относительно $[a, b]$.

На первом шаге длина интервала неопределенности уменьшилась с 1 до F_{k-1}/F_k. На последующих шагах уменьшение длин интервалов выражается числами

$$\frac{F_{k-2}}{F_{k-1}}, \frac{F_{k-3}}{F_{k-2}}, \ldots, \frac{F_2}{F_3}, \frac{F_1}{F_2}(1+2\varepsilon).$$

Таким образом, длина окончательного интервала неопределенности равна $(1+2\varepsilon)/F_k$. Пренебрегая ε, заметим, что асимптотически $1/F_k$ равно r^k при $k\to\infty$, где

$$r=\frac{\sqrt{5}-1}{2}\approx 0.6180.$$

Следовательно, асимптотически, для больших k каждый шаг поиска Фибоначчи сужает интервал неопределенности с коэффициентом 0.6180. Этот результат нужно сравнить с 0.5, коэффициентом сужения интервала неопределенности в методе бисекции для нахождения нуля функции.

Для больших значений k координаты точек x_{k-1} и x_k близки соответственно к $1-r\approx 0.3820$ и $r\approx 0.6180$, и старт с этих значений близок к оптимальной стратегии. Чтобы понять, как продолжать дальше, предположим (для определенности), что $f(0.3820)>f(0.6180)$. Тогда, как мы знаем, $\bar{x}$ находится в интервале $[0, 0.6180]$. Следовательно, нужно вычислять f в точках $0.3820*0.6180$ и $0.6180*0.6180$. Но, поскольку $0.6180*0.6180\approx 0.3820\approx x_{k-1}$, то в этой точке f уже известна. Таким образом, на каждом шаге, начиная со второго, требуется лишь одно вычисление функции, и каждый шаг уменьшает длину интервала неопределенности с коэффициентом 0.6180. В противоположность поиску Фибоначчи, здесь не нужно фиксировать число k до начала поиска.

В связи с исторической известностью числа $r\approx 0.6180$ этот метод нахождения $\bar{x}$ называется *поиском золотого сечения*. Число $1/r=\varphi=(1+\sqrt{5})/2=1.6180\ldots$ называется *золотым отношением*. В гл. 7 было отмечено, что порядок сходимости метода секущих равен числу φ. Золотое отношение φ имеет важное значение и возникает во многих различных задачах. Интересное обсуждение, связанное с этим числом, есть в книге Гарднер (1961).

Поиск золотого сечения аналогичен методу бисекции для нахождения действительного нуля функции (гл. 7) в том отношении, что он с гарантией работает даже в самом худшем случае и что платой за эту гарантию является медленная, всего лишь

линейная сходимость. Метод золотого сечения не извлекает никаких выгод из возможной гладкости функции f.

Если известно, что f имеет непрерывные производные и можно начать достаточно близко к $\bar{x}$, то можно сконструировать следующий итерационный процесс: начинаем с трех произвольных действительных чисел v_1, v_2, v_3. В общем случае имеем числа v_{k-2}, v_{k-1} и v_k. Пусть v_{k+1} — абсцисса вершины параболы (с вертикальной осью), проходящей через точки $(v_i, f(v_i))$, $i=k-2$, $k-1$, k. Продолжаем итерации с числами v_{k-1}, v_k и v_{k+1}. Этот процесс называется *последовательной параболической интерполяцией*. Можно доказать, что для достаточно хороших начальных приближений итерации сходятся со скоростью порядка $\approx 1.324\ldots$, при условии, что $f''(x)>0$ в точке минимума (т. е. $f'(x)$ имеет простой нуль).

8.2. Подпрограмма FMIN

В книге Брент (1973) предложен алгоритм нахождения минимума, в основе которого — комбинация метода золотого сечения и последовательной параболической интерполяции. Алгоритм этот совершенно аналогичен методу нахождения нуля ZEROIN из гл. 7, который использует комбинацию бисекции и обратной квадратичной интерполяции. В конце этого параграфа помещена трансляция на ФОРТРАН алгоритма Брента, записанного автором на языке АЛГОЛ 60. Это — подпрограмма-функция

FUNCTION FMIN (A, B, F, TOL).

Здесь [A, B] — исходный интервал, на котором определена функция F. TOL — это входной параметр, который, грубо говоря, задает желаемую длину интервала неопределенности на выходе. Следовательно, если положить TOL равным 10^{-10}, а результат, полученный FMIN, имеет величину порядка 1, то этот результат имеет около десяти верных цифр. Если же задать для TOL значение 10^{-6}, а ответ равен (например) 0.00000385746, то, вполне возможно, в этом ответе верных знаков нет вовсе; однако, если вы интерпретируете результат такой величины как нуль, тогда вы не нуждаетесь ни в каких верных цифрах. FMIN использует поиск золотого сечения, переключаясь по возможности на последовательную параболическую интерполяцию.

Подпрограммы ZEROIN и FMIN имеют одни и те же параметры — интервал поиска, вычисляемую функцию и границу погрешности. Обе подпрограммы пытаются уменьшить длину интервала, пока она не станет меньше заданной границы. Есть, однако, важное различие между двумя программами, которое

влияет на выбор границ. Если $f(x)=0$, а $f'(x)\neq 0$, то для малых ε

$$f(x+\varepsilon)=f(x)+\varepsilon f'(x)+\varepsilon^2\frac{f''(x)}{2}+\ldots\approx c\varepsilon,$$

где $c=f'(x)$.

Малые изменения в x вызывают пропорционально малые изменения в $f(x)$. Разумно, поэтому, выбирать границу погрешности для ZEROIN примерно той же величины, что и ошибки в значениях функции. Зачастую эти ошибки имеют порядок ошибки округления в машине.

Однако если мы разыскиваем точку минимума, где $f'(x)=0$, а $f''(x)\neq 0$, то для малых ε

$$f(x+\varepsilon)=f(x)+\varepsilon^2\frac{f''(x)}{2}+\ldots\approx f(x)+c\varepsilon^2,$$

где $c=f''(x)/2$.

Изменение порядка ε в x теперь вызывает изменение в $f(x)$ порядка ε^2. Следовательно, было бы неразумно задавать в качестве границы погрешности для FMIN число, меньшее, чем квадратный корень из ошибки в значениях функции. Другими словами, простые нули функции часто можно найти с почти *полной* машинной точностью, а точки, где функция достигает минимума — лишь с примерно *половинной* точностью.

Если TOL — заданная граница для ZEROIN или FMIN, а MACHEPS — машинная точность (определяемая самими подпрограммами), то ZEROIN никогда не вычисляет значения функции в точках, отстоящих одна от другой меньше, чем на

$$2*\mathrm{MACHEPS}*\mathrm{ABS}(X)+\mathrm{TOL}/2,$$

в то время как FMIN никогда не вычисляет функцию в точках, отстоящих одна от другой меньше, чем на

$$\mathrm{SQRT}(\mathrm{MACHEPS})*\mathrm{ABS}(X)+\mathrm{TOL}/3.$$

Таким образом, FMIN с большей вероятностью, чем ZEROIN, игнорирует малые значения параметра TOL.

Унимодальные функции встречаются в практике относительно редко. Часто известно лишь, что $f(x)$ убывает при возрастании x от 0 до некоторой неизвестной точки минимума $\bar{x}$, после чего $f(x)$ возрастает [1]). Желательно как можно быстрее найти приближение к $\bar{x}$, причем никакого априорного представления о том, где $\bar{x}$ находится, нет. Нам не известна какая-либо теория для этой задачи. Программы часто выбирают некоторое $a>0$ и затем вычисляют $f(0)$, $f(a)$, $f(2a)$, $f(2^2a)$, $f(2^3a)$ и т. д., пока зна-

[1]) А затем, быть может, снова убывает. В этом смысле унимодальность отсутствует.— *Прим. перев.*

чения f продолжают убывать; в качестве приближения к $\bar{x}$ принимается кратное $2^k a$ с наименьшим $f(x)$. Или же, если $f(a) > f(0)$, то программа вычисляет значения $f(2^{-1}a)$, $f(2^{-2}a)$,..., пока не будет найдено $f(2^{-r}a)$, такое, что $f(2^{-r}a) < f(0)$, и тогда $2^{-r}a$ принимается за приближение к $\bar{x}$. Читатель может придумать различные способы улучшения приближений к $\bar{x}$, получаемых этими грубыми методами. Во многих приложениях, однако, целесообразно принять подобную грубую оценку для $\bar{x}$ и затем вернуться к основному алгоритму, для которого одномерный поиск является подпрограммой.

Для программы, иллюстрирующей использование FMIN, мы взяли ту же функцию $f(x)=x^3-2x-5$, что и для ZEROIN. Поскольку $f'(x)$ — квадратный многочлен, то экстремумы можно найти аналитически. Заметьте, что значение параметра TOL, используемое здесь, есть квадратный корень из величины, заданной для ZEROIN.

Результат: Z=0.81650.

```
C  ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ FMIN
C
      REAL FUNCTION F(X)
      REAL X
      F=X*(X*X-2.)-5.
      RETURN
      END
C
      EXTERNAL F
      REAL A,B,Z,TOL,FMIN
      A=0.
      B=1.
      TOL=1.0E-5
      Z=FMIN(A,B,F,TOL)
      WRITE(6,1)Z
    1 FORMAT(3H Z=,F12.5)
      STOP
      END

      REAL FUNCTION FMIN(AX,BX,F,TOL)
      REAL AX,BX,F,TOL
C
C  ВЫЧИСЛЯЕТСЯ ПРИБЛИЖЕНИЕ X К ТОЧКЕ, ГДЕ F ДОСТИГАЕТ
C  МИНИМУМА НА ИНТЕРВАЛЕ (AX,BX)
C
C
C  ВХОДНАЯ ИНФОРМАЦИЯ..
C
C       AX          ЛЕВЫЙ КОНЕЦ ИСХОДНОГО ИНТЕРВАЛА
C       BX          ПРАВЫЙ КОНЕЦ ИСХОДНОГО ИНТЕРВАЛА
C       F           ПОДПРОГРАММА-ФУНКЦИЯ, КОТОРАЯ ВЫЧИС-
C                   ЛЯЕТ F(X) ДЛЯ ЛЮБОГО X В ИНТЕРВАЛЕ (AX, BX)
C       TOL         ЖЕЛАЕМАЯ ДЛИНА ИНТЕРВАЛА НЕОПРЕДЕЛЕН-
C                   НОСТИ КОНЕЧНОГО РЕЗУЛЬТАТА ( .GE. 0.0)
C
```

```
C
C  ВЫХОДНАЯ ИНФОРМАЦИЯ..
C
C  FMIN          АБСЦИССА, АППРОКСИМИРУЮЩАЯ ТОЧКУ, ГДЕ
C                F ДОСТИГАЕТ МИНИМУМА
C
C
C     МЕТОД ИСПОЛЬЗУЕТ КОМБИНАЦИЮ ПОИСКА ЗОЛОТОГО
C  СЕЧЕНИЯ И ПОСЛЕДОВАТЕЛЬНОЙ ПАРАБОЛИЧЕСКОЙ ИНТЕР-
C  ПОЛЯЦИИ. СХОДИМОСТЬ НИКОГДА НЕ БЫВАЕТ ЗНАЧИТЕЛЬ-
C  НО ХУЖЕ, ЧЕМ ДЛЯ ПОИСКА ФИБОНАЧЧИ. ЕСЛИ F ИМЕЕТ
C  НЕПРЕРЫВНУЮ ВТОРУЮ ПРОИЗВОДНУЮ, ПОЛОЖИТЕЛЬНУЮ
C  В ТОЧКЕ МИНИМУМА (НЕ СОВПАДАЮЩЕЙ НИ С АХ, НИ С ВХ),
C  ТО СХОДИМОСТЬ СВЕРХЛИНЕЙНАЯ И ОБЫЧНО ИМЕЕТ ПОРЯ-
C  ДОК ПРИМЕРНО 1.324...
C     ФУНКЦИЯ F НИКОГДА НЕ ВЫЧИСЛЯЕТСЯ В ДВУХ ТОЧ-
C  КАХ, ОТСТОЯЩИХ ДРУГ ОТ ДРУГА МЕНЕЕ ЧЕМ НА
C  EPS*ABS(X)+(TOL/3), ГДЕ EPS ПРИБЛИЗИТЕЛЬНО РАВНО КВАД-
C  РАТНОМУ КОРНЮ ИЗ ОТНОСИТЕЛЬНОЙ МАШИННОЙ ТОЧНОС-
C  ТИ. ЕСЛИ F—УНИМОДАЛЬНАЯ ФУНКЦИЯ И ВЫЧИСЛЕННЫЕ
C  ЗНАЧЕНИЯ F СОХРАНЯЮТ УНИМОДАЛЬНОСТЬ ПРИ СОБЛЮ-
C  ДЕНИИ УКАЗАННОГО УСЛОВИЯ РАЗДЕЛЕННОСТИ, ТО FMIN
C  АППРОКСИМИРУЕТ АБСЦИССУ ГЛОБАЛЬНОГО МИНИМУМА F
C  НА ИНТЕРВАЛЕ (АХ,ВХ) С ОШИБКОЙ, МЕНЬШЕЙ 3*EPS*ABS(X)+
C  +TOL. ЕСЛИ F НЕ ЯВЛЯЕТСЯ УНИМОДАЛЬНОЙ, ТО FMIN
C  МОЖЕТ С ТОЙ ЖЕ ТОЧНОСТЬЮ АППРОКСИМИРОВАТЬ ЛОКАЛЬ-
C  НЫЙ МИНИМУМ, ВОЗМОЖНО, НЕ СОВПАДАЮЩИЙ С ГЛОБАЛЬ-
C  НЫМ.
C     ЭТА ПОДПРОГРАММА-ФУНКЦИЯ ЯВЛЯЕТСЯ СЛЕГКА МОДИ-
C  ФИЦИРОВАННОЙ ВЕРСИЕЙ АЛГОЛ 60—ПРОЦЕДУРЫ LOCALMIN,
C  ПРИВЕДЕННОЙ В КНИГЕ RICHARD BRENT, ALGORITHMS FOR
C  MINIMIZATION WITHOUT DERIVATIVES, PRENTICE—HALL,
C  INC.(1973).
C
C
      REAL A,B,C,D,E,EPS,XM,P,Q,R,TOL1,TOL2,U,V,W
      REAL FU,FV,FW,FX,X
C
C  C ЕСТЬ ВОЗВЕДЕННАЯ В КВАДРАТ ВЕЛИЧИНА, ОБРАТНАЯ
C  К ЗОЛОТОМУ СЕЧЕНИЮ
C
      C=0.5*(3.-SQRT(5.0))
C
C  EPS ПРИБЛИЗИТЕЛЬНО РАВНО КВАДРАТНОМУ КОРНЮ ИЗ ОТ-
C  НОСИТЕЛЬНОЙ МАШИННОЙ ТОЧНОСТИ
C
      EPS=1.00
   10 EPS=EPS/2.00
      TOL1=1.0+EPS
      IF(TOL1 .GT. 1.00)GO TO 10
      EPS=SQRT(EPS)
C
C  ПРИСВОЕНИЕ НАЧАЛЬНЫХ ЗНАЧЕНИЙ
C
      A=AX
      B=BX
      V=A+C*(B-A)
```

```
      W=V
      X=V
      E=0.0
      FX=F(X)
      FV=FX
      FW=FX
C
C   ЗДЕСЬ НАЧИНАЕТСЯ ОСНОВНОЙ ЦИКЛ
C
   20 XM=0.5*(A+B)
      TOL1=EPS*ABS(X)+TOL/3.0
      TOL2=2.0*TOL1
C
C   ПРОВЕРИТЬ КРИТЕРИЙ ОКОНЧАНИЯ
C
      IF(ABS(X-XM) .LE. (TOL2-0.5*(B-A)))GO TO 90
C
C   НЕОБХОДИМО ЛИ ЗОЛОТОЕ СЕЧЕНИЕ
C
      IF(ABS(E) .LE. TOL1)GO TO 40
C
C   ПОСТРОИТЬ ПАРАБОЛУ
C
      R=(X-W)*(FX-FV)
      Q=(X-V)*(FX-FW)
      P=(X-V)*Q-(X-W)*R
      Q=2.00*(Q-R)
      IF(Q .GT. 0.0)P=-P
      Q=ABS(Q)
      R=E
      E=D
C
C   ПРИЕМЛЕМА ЛИ ПАРАБОЛА
C
   30 IF(ABS(P) .GE. ABS(0.5*Q*R))GO TO 40
      IF(P .LE. Q*(A-X))GO TO 40
      IF(P .GE. Q*(B-X))GO TO 40
C
C   ШАГ ПАРАБОЛИЧЕСКОЙ ИНТЕРПОЛЯЦИИ
C
      D=P/Q
      U=X+D
C
C   F НЕ СЛЕДУЕТ ВЫЧИСЛЯТЬ СЛИШКОМ БЛИЗКО К АХ ИЛИ ВХ
C
      IF((U-A) .LT. TOL2)D=SIGN(TOL1,XM-X)
      IF((B-U) .LT. TOL2)D=SIGN(TOL1,XM-X)
      GO TO 50
C
C   ШАГ ЗОЛОТОГО СЕЧЕНИЯ
C
   40 IF(X .GE. XM)E=A-X
      IF(X .LT. XM)E=B-X
      D=C*E
C
C   F НЕ СЛЕДУЕТ ВЫЧИСЛЯТЬ СЛИШКОМ БЛИЗКО К X
C
```

```
50      IF(ABS(D) .GE. TOL1) U=X+D
        IF(ABS(D) .LT. TOL1) U=X+SIGN(TOL1, D)
        FU=F(U)
C
C   ПРИСВОИТЬ НОВЫЕ ЗНАЧЕНИЯ ПАРАМЕТРАМ A, B, V, W И X
C
        IF(FU .GT. FX) GO TO 60
        IF(U .GE. X) A=X
        IF(U .LT. X) B=X
        V=W
        FV=FW
        W=X
        FW=FX
        X=U
        FX=FU
        GO TO 20
    60  IF(U .LT. X) A=U
        IF(U. GE. X) B=U
        IF(FU .LE. FW) GO TO 70
        IF(W .EQ. X) GO TO 70
        IF(FU .LE. FV) GO TO 80
        IF(V .EQ. X) GO TO 80
        IF(V .EQ. W) GO TO 80
        GO TO 20
    70  V=W
        FV=FW
        W=U
        FW=FU
        GO TO 20
    80  V=U
        FV=FU
        GO TO 20
C
C   КОНЕЦ ОСНОВНОГО ЦИКЛА
C
    90  FMIN=X
        RETURN
        END
```

8.3. Многомерная оптимизация

Локальная минимизация функции n переменных — настолько важная задача, что алгоритмы для ее решения были изобретены еще более 130 лет назад. В 1845 г. Коши предложил метод, называемый теперь *методом скорейшего спуска*. Обозначим вектор $(x_1, \ldots, x_n)^T$ через $\mathbf{x}$; предположим, что функция $\mathbf{f}(\mathbf{x})$ достаточно гладкая. Пусть $g(\mathbf{x})$ обозначает градиент функции f в точке $\mathbf{x}$, т. е. вектор, i-я компонента которого равна $\partial f/\partial x_i$. Тогда — $\mathbf{g}(\mathbf{x})$ определяет направление наискорейшего убывания f в точке $\mathbf{x}$. Коши предложил искать минимальное значение f на полупрямой $\mathbf{x}-\mathbf{tg}\,(0<t<\infty)$. Это одномерный поиск того типа, который мы обсуждали в конце § 8.2. Найдя этот минимум, начинают новый

поиск вдоль полупрямой скорейшего спуска, исходящей из нового $\mathbf{x}$. При некоторых слабых предположениях метод сойдется к тому локальному минимуму функции f, в чьей *чаше* находится первое приближение.

Теоретический анализ предсказал, что метод Коши в некоторых ситуациях должен сходиться крайне медленно, и опыт подтвердил это во многих практических случаях даже для скромных значений n, равных 2, 3 или 4. Причину можно понять, рассматривая при $n=2$ функцию f, для которой линиями уровня являются эллипсы большого эксцентриситета, вроде показанного на

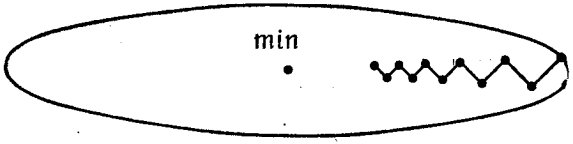


рис. 8.2. (В качестве такой функции можно взять, например, $x_1^2+100\,000\,x_2^2$.) Продвижение к минимуму, находящемуся в центре, происходит очень медленно; при этом маршрут состоит из осцилляций в направлениях локальных градиентов. Имеется явная необходимость в ускорении сходимости.

Основная трудность здесь в том, что локальный градиент даже приблизительно не указывает направление к центру эллипсов. Чтобы найти направление быстрого изменения произвольной строго выпуклой квадратичной функции f от n переменных, разложим $f(\mathbf{x})$ около точки минимума, которую обозначим через $\mathbf{a}$. Пусть B — постоянная матрица, составленная из вторых частных производных функции f в точке $\mathbf{a}$,— так называемая *матрица Гессе:*

$$b_{ij}=\frac{\partial^2 f}{\partial x_i\,\partial x_j}.$$

Тогда, применяя теорему Тейлора и учитывая, что $\mathbf{g}(\mathbf{a})=0$ в точке минимума, можем написать

$$f(\mathbf{x})=f(\mathbf{a})+\frac{1}{2}(\mathbf{x}-\mathbf{a})^T B(\mathbf{x}-\mathbf{a}).$$

Следовательно, градиент $\mathbf{g}$ в точке $\mathbf{x}$ равен

$$\mathbf{g}(\mathbf{x})=B(\mathbf{x}-\mathbf{a}).$$

Если мы находимся в произвольной точке $\mathbf{x}$, то можем вычислить $\mathbf{a}$ по формуле

$$\mathbf{a}=\mathbf{x}-B^{-1}\mathbf{g}(\mathbf{x}).$$

Таким образом, минимум функции f можно найти, зная B^{-1} и градиент $\mathbf{g}$ в точке $\mathbf{x}$. (B — положительно определенная мат-

рица.) Обозначим матрицу B^{-1} через H. Тогда матрица H есть оператор, который переводит локальное направление скорейшего спуска — $\mathbf{g}$ в точке $\mathbf{x}$ в правильное направление от $\mathbf{x}$ к центру эллипсоидов, т. е. к точке минимума f. Через H точку минимума $\mathbf{a}$ можно выразить как

$$\mathbf{a} = \mathbf{x} - H\mathbf{g}.$$

Можно рассматривать эту формулу как применение метода Ньютона к решению системы уравнений $\mathbf{g}(\mathbf{x})=0$.

Если гладкая функция f не является квадратичной, она тем не менее будет приблизительно квадратичной в окрестности минимума, в котором матрица Гессе не вырождается. Поэтому полученные уравнения подсказывают возможный подход к задаче нахождения минимума произвольной функции f. Применим итерационный процесс вида

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k \mathbf{g}_k,$$

где $\mathbf{g}_k = \mathbf{g}(\mathbf{x}_k)$, а H_k есть k-е приближение к матрице H, обратной для матрицы Гессе B функции f в точке минимума $\mathbf{a}$. Положительное число α_k выбирается на каждом шаге итераций так, чтобы достигался локальный минимум f в направлении $-H_k\mathbf{g}_k$ от точки $\mathbf{x}_k$. В этом итерационном процессе пытаются одновременно найти и точку минимума $\mathbf{a}$, и обратную матрицу H.

Основная идея — строить H_k таким образом, чтобы после n шагов H_n совпала с H, будь f квадратичной функцией. Таким образом, в случае квадратичной функции f n-е приближение $\mathbf{x}_n$ совпало бы при использовании точной арифметики с $\mathbf{a}$. Можно полагать поэтому, что и для неквадратичных функций n-е приближение $\mathbf{x}_n$ должно быть гораздо лучшим приближением к минимуму $\mathbf{a}$, чем $\mathbf{x}_0$.

Методы этого типа называются алгоритмами с *переменной метрикой*, поскольку матрицу B можно интерпретировать как метрику в пространстве векторов $\mathbf{g}$. Другое распространенное наименование — *квазиньютоновы* методы. Первый подобный алгоритм построил в 1959 г. Давидон. Со времени опубликования статьи Давидона был предложен ряд усовершенствований квазиньютоновых методов. Мы отсылаем читателя к обзору Бройдена, содержащемуся в сборнике Мюррей (1972).

Пауэлл доказал сходимость алгоритма для любой функции $f(\mathbf{x})$, матрица Гессе которой положительно определена при всех $\mathbf{x}$; однако функции, к которым применяется алгоритм, редко удовлетворяют этому условию. Известно, что сходимость метода быстрее линейной.

Алгоритмы этого класса называют еще *квадратично завершающимися* [1]), поскольку для квадратичных функций они сходятся

[1]) В оригинале — qudratically terminating.— *Прим. перев.*

в конечное число шагов. Это не означает, что для функций общего вида порядок сходимости равен 2, хотя так и может случиться.

Подпрограммы для нелинейной оптимизации можно классифицировать соответственно тому, решают ли они безусловные задачи или задачи с ограничениями, является ли целевая функция суммой квадратов или произвольной нелинейной функцией, требуется или нет вычисление производных. Библиотеки таких программ обычно включают и программы для решения нелинейных систем уравнений.

Большой вклад в ведущуюся сейчас работу по созданию надежных, эффективных и универсальных подпрограмм внесли британские ученые, в особенности сотрудники Научно-исследовательского центра атомной энергии в Харуэлле и Национальных физических лабораторий в Теддингтоне. Примерами таких программ могут быть АЛГОЛ 60 — процедуры, включенные в сборник Гилл и др. (1972). Ряд других программ вместе с соответствующим теоретическим обоснованием описан в двух родственных обзорах: один из них (Мюррей (1972)) посвящен безусловной оптимизации, другой (Гилл, Мюррей (1972)) — оптимизации при наличии ограничений.

Ко времени написания этой главы (т. е. в 1976 г.) наиболее современными были программы, имеющиеся в Харуэлле, Теддингтоне и Оксфорде, где работает Британская группа по численным алгоритмам. За пределами Великобритании, в США, группа из Национальных лабораторий в Аргонне ведет работу по созданию пакета подпрограмм под названием MINPACK. Эти программы будут опубликованы после тщательного тестирования.

УПРАЖНЕНИЯ

8.1. При расширении области определения функции $\operatorname{erf}(x)$ на комплексные значения аргумента возникает действительная функция действительного переменного x, называемая интегралом Доусона:

$$D(x) = e^{-x^2} \int_0^x e^{t^2}\, dt.$$

Поскольку $D(x)$ стремится к нулю при стремлении x к 0 или ∞, а для промежуточных значений $D(x)$ положительна, то должен иметься конечный максимум. Используя FMIN, найдите максимум функции $D(x)$.

8.2. С помощью FMIN найдите максимум функции

$$f(x) = (\sin x)^6 \operatorname{tg}(1-x)\, e^{30x}$$

на интервале [0,1].

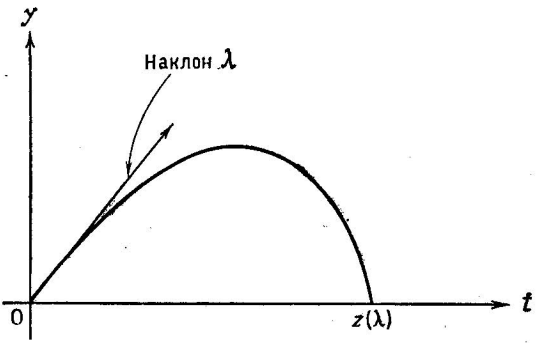
8.3. Для любого $\lambda > 0$ обозначим через $z(\lambda)$ наименьший положительный нуль функции $y(t)$, являющейся решением задачи Коши для обыкновенного

дифференциального уравнения

$$y''(t)+I_0(y)+\frac{t}{10}=0,$$
$$y(0)=0, \quad y'(0)=\lambda,$$

где $I_0(t)$ — модифицированная функция Бесселя нулевого порядка (см. Абрамовиц, Стегун (1964), с. 374—375) [1].

График решения $y(t)$ приблизительно совпадает с показанным на рисунке.



(а) Найдите λ_{max} — единственное значение λ, для которого $z(\lambda)$ максимально.

(б) Найдите также $z(\lambda_{max})$.

(в) Постройте таблицу значений $y(t)$ при $t=0(0.1)z(\lambda_{max})$ для той функции $y(t)$, которая максимизирует $z(\lambda)$, т. е. для $y(t)$, удовлетворяющей условию $y'(0)=\lambda_{max}$.

(г) Обсудите точность полученных результатов.

---

[1] Или: Никифоров А. Ф., Уваров В. Б. Специальные функции математической физики.— М.: Наука, 1978, с. 112.— *Прим. перев.*

9. НАИМЕНЬШИЕ КВАДРАТЫ И СИНГУЛЯРНОЕ РАЗЛОЖЕНИЕ

9.1. Выравнивание данных методом наименьших квадратов

Во многих различных областях науки встречается такая задача. Предположим, что заданы m точек

$$(t_i,\ y_i),\ i=1,\ \ldots,\ m.$$

Будем рассматривать t как независимое переменное, а y — как зависимое, связанное с t некоторой неизвестной функциональной зависимостью

$$y_i=y(t_i).$$

Предположим, далее, что y нужно аппроксимировать линейной комбинацией n заданных базисных функций φ_j:

$$y(t)\approx c_1\varphi_1(t)+c_2\varphi_2(t)+\ldots+c_n\varphi_n(t).$$

Линейная комбинация в правой части называется *линейной математической моделью*. Задача состоит в том, чтобы выбрать n коэффициентов $c_1,\ldots,c_n$ таким образом, чтобы модель в том или ином смысле согласовывалась с данными. Модель линейна, потому что коэффициенты входят в нее линейно; однако базисные функции могут быть нелинейными функциями от t.

Наиболее распространенной линейной моделью является полином

$$y(t)\approx c_1+c_2t+c_3t^2+\ldots+c_nt^{n-1}.$$

Он получается, если взять $\varphi_j(t)=t^{j-1}$, хотя, как мы увидим, более подходящими могут оказаться иные базисные полиномы. Среди других примеров — тригонометрическое приближение

$$y(t)\approx c_1\sin t+c_2\sin 2t+\ldots+c_n\sin nt,$$

где $\varphi_j(t)=\sin jt$, и экспоненциальное приближение с фиксированными экспонентами

$$y(t)\approx c_1e^{\lambda_1 t}+\ldots+c_ne^{\lambda_n t},$$

в котором $\varphi_j(t)=\exp(\lambda_j t)$ для заданных λ_j. (Если нужно определять также и λ_j, то это превращается в нелинейную модель. См. § 8.5 и упр. 9.3.)

Мы будем рассматривать случай, когда число m заданных точек больше или равно числу n неизвестных коэффициентов. В этом случае задача выбора коэффициентов переопределена, и обычно не удается построить модель, точно удовлетворяющую данным, т. е. интерполирующую их.

Из многих различных критериев для определения коэффициентов c_j наиболее часто используется метод *наименьших квадратов*. Для любого выбора коэффициентов c_j невязка в i-й заданной точке равна

$$r_i=\sum_{j=1}^{n} c_j\varphi_j(t_i)-y_i.$$

Критерий наименьших квадратов требует, чтобы c_j выбирались из условия минимального значения для суммы квадратов невязок; т. е.

$$\underset{c_j}{\text{минимизировать}} \sum_{i=1}^{m} r_i^2.$$

Если модель может точно удовлетворить данным, то этот минимум будет нулем, так что интерполяция включается сюда как специальный случай.

Критерий наименьших квадратов необязательно определяет единственный набор коэффициентов. Если базисные функции линейно зависимы в заданных точках, что означает существование коэффициентов γ_j, среди которых есть ненулевые, таких, что

$$\sum_{j=1}^{n} \gamma_j\varphi_j(t_i)=0,\quad i=1,\ \ldots,\ m,$$

то к коэффициентам c_j можно прибавить любое кратное коэффициентов γ_j, не изменяя сумму квадратов невязок. Важной и часто забываемой задачей при выравнивании данных методом наименьших квадратов с произвольными базисными функциями является обнаружение такой зависимости и неединственности и принятие надлежащих мер.

Имеется много различных алгоритмов для вычисления набора коэффициентов, дающего минимальную сумму квадратов. Одна из возможностей — это применение математического анализа. Пусть

$$r=\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{n} c_j\varphi_j(t_i)-y_i\right)^2\right)^{1/2}.$$

Наша цель — минимизировать r или, что все равно, минимизировать r^2; потребуем, поэтому, чтобы для $k=1, \ldots, n$

$$\frac{\partial r^2}{\partial c_k}=0.$$

Беря производные и изменяя порядок суммирования, получаем

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m}\varphi_j(t_i)\varphi_k(t_i)\right)c_j=\sum_{i=1}^{m}y_i\varphi_k(t_i).$$

Это система из n линейных уравнений с n неизвестными c_j. Ее можно записать в матричной форме

$$Pc=q,$$

где

$$p_{kj}=\sum_{i=1}^{m}\varphi_k(t_i)\varphi_j(t_i),$$

$$q_k=\sum_{i=1}^{m}\varphi_k(t_i)y_i.$$

Например, при аппроксимации в смысле наименьших квадратов посредством прямой

$$y(t)\approx c_1+c_2t$$

имеем

$$P=\begin{pmatrix} m & \sum t_i \\ \sum t_i & \sum t_i^2 \end{pmatrix},$$

$$q=\begin{pmatrix} \sum y_i \\ \sum t_i y_i \end{pmatrix}.$$

Уравнения $Pc=q$ называются *нормальными уравнениями*. Заметим, что матрица P зависит лишь от базисных функций; значения y_i в нее не входят. В принципе нормальные уравнения можно решать, используя подпрограммы DECOMP и SOLVE из гл. 3. Однако, поскольку $p_{kj}=p_{jk}$, матрица P симметрична, и время и память, требуемые для DECOMP, могут быть сокращены вдвое. Более того, можно показать, что P *положительно определена*, и потому не требуется выбор ведущих элементов. Следовательно, используя вариант гауссова исключения, рассчитанный на положительно определенные симметричные матрицы, можно решать нормальные уравнения с менее чем половинными затратами по сравнению с DECOMP и SOLVE.

Однако есть серьезные, фундаментальные доводы против использования нормальных уравнений. Оказывается, что матрица P часто имеет очень большое число обусловленности; поэтому каким бы методом ни решались нормальные уравнения,

ошибки во входной информации и ошибки округлений, внесенные в процессе решения, чрезмерно преумножаются в вычисленных коэффициентах.

В крайней ситуации, когда базисные функции $\varphi_j(t)$ линейно зависимы, можно показать, что матрица P вырождена, и число обусловленности может рассматриваться как бесконечное. Следовательно, методы, избегающие высокого числа обусловленности, связанного с нормальными уравнениями, есть в то же время методы, способные обнаруживать линейную зависимость среди базисных функций. Гауссово исключение и его варианты не слишком хорошо приспособлены к выявлению такой зависимости. Наша оценка числа обусловленности в какой-то мере помогает, однако она может лишь сигнализировать об опасности, но не предложить лекарство.

Наиболее надежный метод для вычисления коэффициентов в общей задаче наименьших квадратов основан на матричной факторизации, называемой сингулярным разложением. Есть другие методы, требующие меньше машинного времени и памяти, однако они менее эффективны в том, что касается учета ошибок заданной информации, ошибок округления и линейной зависимости.

Сингулярное разложение, или SVD [1], подробно описывается в последующих параграфах этой главы. Однако, чтобы использовать SVD в практическом выравнивании наименьшими квадратами, совсем не обязательно понимать эти параграфы во всех деталях.

Метод SVD начинается с составления матрицы, известной в статистическом анализе экспериментов как *матрица плана*. Это прямоугольная матрица из m строк и n столбцов, элементы которой суть

$$a_{ij}=\varphi_j(t_i).$$

Если через y обозначить вектор размерности m с элементами y_i, а через c — вектор размерности n с компонентами c_j, то приближенные равенства

$$\sum_{j=1}^{n} c_j\varphi_j(t_i) \approx y_i, \quad i=1, \ldots, m,$$

можно переписать в виде

$$Ac \approx y.$$

Подпрограмма SVD, приведенная в конце главы, исходя из входной матрицы A, получает на выходе три матрицы Σ, U и V. Матрица Σ — диагональная с неотрицательными диагональны-

[1] От singular value decomposition.— *Прим. перев.*

ми элементами, называемыми *сингулярными числами* матрицы A. Матрицы U и V используются для преобразования уравнений

$$Ac \approx y$$

в эквивалентную диагональную систему

$$\Sigma \bar{c} \approx \bar{y}.$$

Пусть σ_j, $j=1, \ldots, n$,— диагональные элементы Σ. В принципе если все σ_j отличны от нуля, то можно разрешить преобразованные уравнения, полагая

$$\bar{c}_j = \frac{\bar{y}_j}{\sigma_j}, \quad j = 1, \ldots, n.$$

Однако на практике это не всегда желательно, если некоторые σ_j малы.

Оказывается, что все σ_j не равны нулю тогда и только тогда, когда базисные функции φ_j линейно независимы в заданных точках. Поэтому ключ к правильному использованию SVD — это введение границы τ, отражающей точность исходных данных и точность используемой плавающей арифметики. Всякое σ_j, большее, чем τ, приемлемо, и соответствующее $\bar{c}_j$ вычисляется как $\bar{y}_j/\sigma_j$. Любое σ_j, меньшее, чем τ, рассматривается как пренебрежимо малое, и соответствующему $\bar{c}_j$ может быть придано произвольное значение. С этой произвольностью значений связана неединственность набора коэффициентов, получаемых методом наименьших квадратов. Изменения входных данных и ошибки округлений, меньшие, чем τ, могут привести к совершенно другому набору коэффициентов, определяемых этим методом. Поскольку обычно желательно, чтобы эти коэффициенты были по возможности малы, то полагаем $\bar{c}_j=0$, если $\sigma_j \leqslant \tau$.

Отношение $\sigma_{\max}/\sigma_{\min}$, где $\sigma_{\max}$ — наибольшее, а $\sigma_{\min}$ — наименьшее ненулевое сингулярное число, можно рассматривать как число обусловленности матрицы A. Это не то число обусловленности, которое было введено в гл. 3, но его свойства во многом те же самые; обычно и порядок величины у него тот же. Как мы увидим в следующем параграфе, для задач на наименьшие квадраты оно является более подходящей мерой обусловленности.

Отбрасывание чисел σ_j, меньших, чем τ, приводит к уменьшению числа обусловленности до $\sigma_{\max}/\tau$. Поскольку число обусловленности является множителем в увеличении ошибки, то следствием будет более надежное определение коэффициентов c_j. Цена за эту возросшую надежность — возможное увеличение невязок.

Следующие далее сегменты фортранной программы иллюстрируют использование SVD для задач выравнивания по методу наименьших квадратов. Мы не привели полную подпрограмму, чтобы эти сегменты не потонули во множестве прочих деталей, таких, как ввод данных, генерирование базисных функций, вычисление модели в других точках, используя найденные коэффициенты, и т. д.

Матрица плана может быть построена следующим образом:

```
      DO 20 I = 1, M
         T(I) = абсцисса I-ой заданной точки
         Y(I) = ордината I-ой заданной точки
         DO 10 J = 1, N
            A(I,J) = J-я базисная функция,
10       CONTINUE                    вычисленная в T(I)
20    CONTINUE.
```

В случае полиномиальной аппроксимации эффективный способ построения A таков:

```
      DO 20 I = 1, M
         T(I) = ...
         Y(I) = ...
         A(I,1) = 1.0
         DO 10 J = 2, N
            A(I,J) = T(I)*A(I,J-1)
10       CONTINUE
20    CONTINUE.
```

Вслед за этим можно немедленно обратиться к SVD. (Относительно деталей использования SVD см. комментарии в ее распечатке.)

```
      CALL SVD(MDIM, M, N, A, SIGMA, .TRUE., U,
     .TRUE., V, IERR, WORK)
      IF (IERR .NE. 0) WRITE(6,13)
13    FORMAT('ОШИБКА. ВОЗВРАТ ИЗ SVD')
```

В нашей практике не было случая, чтобы при правильном использовании SVD произошел выход с диагностикой ошибки.

Следующий сегмент программы находит наибольшее сингулярное число, которое будет участвовать в выявлении пренебрежимо малых сингулярных чисел. Здесь также присваивается нулевое значение вектору коэффициентов, что будет конечным результатом, если все сингулярные числа окажутся пренебрежимо малыми.

```
      SIGMA1 = 0.
      DO 30 J = 1, N
         IF (SIGMA(J) .GT. SIGMA1) SIGMA1 = SIGMA(J)
         C(J) = 0.
   30 CONTINUE
```

Теперь нужно задать границу относительной погрешности RELERR. Например, если данные имеют три верных знака[1], то подходящим будет значение $\mathrm{RELERR}=10^{-3}$. Если же данные рассматриваются как точные, то RELERR должно отражать ошибку округлений, ожидаемую при вычислениях внутри самой SVD; эта ошибка имеет порядок умноженного на n машинного эпсилон. Граница абсолютной ошибки τ получается теперь как

```
TAU = RELERR*SIGMA1.
```

Следующий сегмент применяет к исходным данным преобразование U, находит преобразованные коэффициенты, а затем применяет преобразование V, чтобы получить сами коэффициенты

```
      DO 60 J = 1, N
         IF (SIGMA(J) .LE. TAU) GO TO 60
         S = 0.
         DO 40 I = 1, M
            S = S + U(I,J)*Y(I)
   40    CONTINUE
         S = S/SIGMA(J)
         DO 50 I = 1, N
            C(I) = C(I) + S*V(I,J)
   50    CONTINUE
   60 CONTINUE
```

Заметим, что M — это число элементов в Y и число строк в A и U, в то время как N — число элементов в C и число строк в V. Все матрицы имеют N столбцов.

Коэффициенты теперь готовы для использования. При желании их можно вывести на печать.

```
WRITE(6,...) (C(J), J=1, N)
```

Вычисляется модель в произвольной точке TT таким образом:

```
      YY = 0.
      DO 70 J = 1, N
         YY = YY + C(J)*(J-я базисная функция,
                         вычисленная в TT)
   70 CONTINUE
```

[1]) Имеется в виду десятичных.— *Прим. перев.*

Для полиномиальной модели следует использовать схему Горнера (см. § 4.2).

```
      YY = 0.
      DO 70 JB = 1, N
         J = N + 1 - JB
         YY = TT*YY + C(J)
   70 CONTINUE
```

Без труда можно получить квадратный корень из суммы квадратов невязок, т. е. ту величину, которая минимизируется.

```
      RSQ = 0.
      DO 90 I = 1, M
         RI = 0.
         DO 80 J = 1, N
            RI = RI + C(J)*A(I,J)
   80    CONTINUE
         RSQ = RSQ + (RI - Y(I))**2
   90 CONTINUE
      R = SQRT(RSQ)
```

Наконец, было сказано, что важной особенностью SVD является способность обнаруживать зависимость и отсутствие единственности. Следующий фрагмент находит пренебрежимо малые сингулярные числа. Для каждого найденного печатается список коэффициентов, называемых *нулевыми коэффициентами*. Любое кратное α этих коэффициентов можно добавить к напечатанным прежде, не увеличивая r более чем на $\alpha\tau$. Коэффициенты нормируются так, что сумма их квадратов равна 1.0.

```
      UNIQUE = .TRUE.
      DO 100 J = 1, N
         IF (SIGMA(J) .GT. TAU) GO TO 100
         UNIQUE = .FALSE.
         WRITE(6,101)
         WRITE(6,...) (V(I,J), I = 1, N)
  100 CONTINUE
      IF (UNIQUE) WRITE(6,102)
  101 FORMAT('НУЛЕВЫЕ КОЭФФИЦИЕНТЫ')
  102 FORMAT('КОЭФФИЦИЕНТЫ ОПРЕДЕЛЕНЫ
                          ОДНОЗНАЧНО')
```

Этот тип дополнительной информации для задач на наименьшие квадраты называется *сингулярным анализом*. Он может быть очень полезен при анализе сложных математических моделей.

Детальное обсуждение сингулярного анализа имеется в книге Лоусон, Хэнсон (1974).

Использование сингулярного анализа для задач выравнивания наименьшими квадратами можно проиллюстрировать на данных Бюро переписей США из упр. 4.5. Заданных точек здесь восемь: $m=8$. Значения t_i суть 1900, 1910, . . ., 1970, а соответствующие значения y_i (единицей является миллион людей) примерно 75.99, 91.97, . . ., 203.21. Если аппроксимировать эти данные квадратичным многочленом

$$y(t)\approx c_1+c_2t+c_3t^2,$$

а вычисления проводить на IBM 360 с удвоенной точностью, то будут получены приблизительно такие результаты

$$c_1=0.373\times 10^5,\ c_2=-0.402\times 10^2,\ c_3=0.108\times 10^{-1}.$$

Проделав те же вычисления с обычной точностью, получим приблизительно следующее:

$$c_1=-0.372\times 10^5,\ c_2=0.368\times 10^2,\ c_3=-0.905\times 10^{-2}.$$

У этих двух наборов коэффициентов не совпадают даже знаки. Если такую модель использовать для предсказания численности населения в 1980 г., то для коэффициентов, вычисленных с удвоенной точностью, прогноз будет 227.78 миллиона, в то время как для коэффициентов, полученных с обычной точностью, 145.21 миллиона. Ясно, что второй набор коэффициентов бесполезен. Но что можно сказать о коэффициентах первого набора? Насколько они достоверны?

Матрица плана для этого примера имеет размеры 8×3:

$$A=\begin{pmatrix} 1 & 0.190\times 10^4 & 0.36100\times 10^7 \\ 1 & 0.191\times 10^4 & 0.36481\times 10^7 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & 0.197\times 10^4 & 0.39204\times 10^7 \end{pmatrix}$$

Ее сингулярные числа можно с достаточной точностью найти в обоих случаях. Они таковы:

$$\sigma_1=0.106\times 10^8,\ \sigma_2=0.648\times 10^2,\ \sigma_3=0.346\times 10^{-3}.$$

Число обусловленности равно $\sigma_1/\sigma_3=0.306\times 10^{11}$, и столь большая величина сигнализирует наличие каких-то трудностей. Для

значений t между 1900 и 1970 гг. три базисные функции 1, t и t^2 очень близки к линейной зависимости.

Чтобы исправить ситуацию, можно предпринять одну из следующих двух мер. Во-первых, можно выбрать границу для относительной ошибки, которая бы отражала точность данных и точность арифметики. Если взять границу, например, в интервале от 0.3×10^{-10} до 0.6×10^{-5}, то описанная выше программа сингулярного анализа отбросит σ_3. Тогда будут получены такие наборы коэффициентов:

$$c_1=-0.167\times10^{-2},\ c_2=-0.162\times10^{1},\ c_3=0.871\times10^{-3}$$

для удвоенной точности и

$$c_1=-0.166\times10^{-2},\ c_2=-0.162\times10^{1},\ c_3=0.869\times10^{-3}$$

для обычной. Теперь они находятся в гораздо лучшем согласии друг с другом. Кроме того, коэффициенты стали существенно меньше, а это значит, что не будет столь большого, как прежде, взаимного уничтожения слагаемых при вычислении квадратичного многочлена. Прогноз численности населения на 1980 г. теперь будет соответственно 212.91 и 214.96 миллиона. Эффект обычной точности все еще заметен, однако результаты уже не являются катастрофическими.

Программа сингулярного анализа печатает также набор нулевых коэффициентов, соответствующих пренебрежимо малому сингулярному числу. Вот эти коэффициенты:

$$\gamma_1=-0.9999995,\ \gamma_2=0.103\times10^{-2},\ \gamma_3=-0.266\times10^{-6}.$$

Для значений t от 1900 до 1970 величина функции $|\gamma_1+\gamma_2 t+\gamma_3 t^2|$ не превосходит 0.0017. Поэтому при любом α коэффициенты можно изменить с c_j на $c_j+\alpha\gamma_j$, и при этом значения, выдаваемые моделью, изменятся не более чем на 0.0017α. Любой из четырех вычисленных нами наборов коэффициентов можно получить из любого другого подобным изменением.

Другая мера, которую можно принять для улучшения ситуации, состоит в изменении базиса. Модели вида

$$y(t)\approx c_1+c_2(t-1900)+c_3(t-1900)^2$$

или

$$y(t)\approx c_1+c_2\left(\frac{t-1935}{10}\right)+c_3\left(\frac{t-1935}{10}\right)^2$$

гораздо более удовлетворительны. Важно при этом то, что независимое переменное преобразуется из интервала [1900, 1970] в какой-нибудь более приемлемый интервал вроде [0, 70] или, еще лучше, [—3.5, 3.5]. Сингулярные числа матриц планов, по-

лученных для этих двух моделей, будут

$$\sigma_1=0.684\times10^4,\ \sigma_2=0.293\times10^2,\ \sigma_3=0.119\times10^1$$

и

$$\sigma_1=0.198\times10^2,\ \sigma_2=0.648\times10^1,\ \delta_3=0.185\times10^1$$

соответственно. Числа обусловленности равны 0.575×10^4 и 10.7; последнее значение более чем приемлемо даже при счете с обычной точностью.

Вычисления можно теперь проводить и в той, и в другой разрядности; если заданная граница погрешности не будет слишком высока, то ни одно из сингулярных чисел не будет расценено как пренебрежимо малое. Все приближения, построенные этим способом, имеют достоверные коэффициенты, и все предсказывают, что численность населения в 1980 г. будет равна примерно 227.78 миллиона.

Мы все же не уверены, какова будет численность населения в 1980 г. В попытке выяснить это можно увеличить степень полинома. Выбор базисных полиномов и границы для ошибки становится тогда более критическим. Возможно, и вероятно даже разумно, применить модели, основанные не на полиномах, а на каких-либо других функциях. Исследование этих возможностей предоставляется читателю в качестве упражнения. Главное, что мы хотели показать, это то, что использование сингулярного разложения предоставляет ценную информацию о надежности и чувствительности моделей, полученных методом наименьших квадратов, и помогает составителю таких моделей оценить их полезность.

9.2. Ортогональность и сингулярное разложение

Сингулярное разложение SVD является мощным вычислительным средством для анализа матриц и задач, связанных с матрицами, которое имеет приложения во многих областях. В последующих параграфах этой главы будет определено SVD, описаны некоторые другие его применения и дан алгоритм для его вычисления. Этот алгоритм — типичный представитель используемых в настоящее время алгоритмов для решения различных матричных задач на собственные значения и может служить одновременно введением в численные методы для этих задач.

Малоизвестно, что сингулярное разложение имеет довольно давнюю историю. Основополагающими в большой мере были работы Голуба и его коллег Кахана, Бизингера и Райнша. Наше обсуждение будет построено главным образом на статье Голуб,

Райнш (1971) [1]). Авторы используемых алгоритмов для матричных собственных значений — Фрэнсис, Рутисхаузер и Уилкинсон; эти алгоритмы описаны в книге Уилкинсон (1965). Сравнительно недавно изданные книги Лоусон, Хэнсон (1974) и Стьюарт (1973) содержат SVD и ряд примыкающих сюда вопросов.

В элементарной линейной алгебре множество векторов называется линейно независимым, если ни один из них не может быть представлен в виде линейной комбинации других. В вычислительной линейной алгебре очень полезно иметь количественную оценку «степени» независимости. Мы хотели бы определить величину, которая бы отражала тот факт, что, например, векторы

$$\begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix} \text{ и } \begin{pmatrix}0\\0\\1\end{pmatrix}$$

очень независимы, в то время как векторы

$$\begin{pmatrix}1.01\\1.00\\1.00\end{pmatrix}, \begin{pmatrix}1.00\\1.01\\1.00\end{pmatrix} \text{ и } \begin{pmatrix}1.00\\1.00\\1.01\end{pmatrix}$$

почти зависимы.

Поскольку два вектора зависимы, если они параллельны, то разумно считать их очень независимыми, если они перпендикулярны или ортогональны. Используем индекс T, чтобы обозначать транспонирование вектора или матрицы. Два вектора u и v называются *ортогональными*, если их скалярное произведение равно нулю, т. е. если

$$u^T v = 0.$$

Далее, вектор u имеет длину 1, если

$$u^T u = 1.$$

Квадратная матрица называется *ортогональной*, если ее столбцы суть попарно ортогональные векторы длины 1. Таким образом, матрица U ортогональная, если

$$U^T U = I,$$

где I — единичная матрица. Заметим, что ортогональная матрица автоматически не вырождена, поскольку $U^{-1} = U^T$. Вскоре мы придадим точный смысл представлению о том, что ортогональная матрица *очень не вырождена*, а ее столбцы — *очень независимы*.

[1]) По существу, алгоритм сингулярного разложения содержится в работе В. В. Воеводина 1968 г. (Ошибки округления в алгебраических процессах. Сб. докладов под ред. В. В. Воеводина.— М.: ВЦ МГУ, 1968, с. 39—58).— *Прим. перев.*

Простейшими примерами ортогональных матриц являются плоские вращения вида

$$U=\begin{pmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{pmatrix}.$$

Если x — вектор в 2-пространстве, то Ux — тот же самый вектор, повернутый на угол θ. Полезно ассоциировать ортогональные матрицы с такими вращениями, несмотря на то что для более высоких размерностей ортогональные матрицы могут быть устроены более сложно. Например, матрица

$$U=\frac{1}{49}\begin{pmatrix}24 & 36 & 23\\ 41 & -12 & -24\\ 12 & -31 & 36\end{pmatrix}$$

ортогональная, хотя ее нельзя интерпретировать как простое плоское вращение.

Умножение на ортогональные матрицы не изменяет такие важные геометрические величины, как длина вектора или угол между двумя векторами. Ортогональные матрицы имеют также в высшей степени замечательные вычислительные свойства, поскольку они не увеличивают ошибок.

Для любой матрицы A и любых двух ортогональных матриц U и V рассмотрим матрицу Σ, определяемую соотношением

$$\Sigma=U^TAV.$$

Если u_j и v_j суть столбцы матриц U и V соответственно, то отдельные компоненты матрицы Σ равны

$$\sigma_{ij}=u_i^TAv_j.$$

За сингулярным разложением скрывается та идея, что надлежащим выбором матриц U и V можно обратить большинство элементов σ_{ij} в нули; более того, можно даже сделать Σ *диагональной* с неотрицательными диагональными элементами. Поэтому дадим такое определение.

Сингулярным разложением действительной $m\times n$-матрицы A называется всякая ее факторизация вида

$$A=U\,\Sigma V^T,$$

где U — ортогональная $m\times m$-матрица, V — ортогональная $n\times n$-матрица, а Σ—диагональная $m\times n$-матрица, у которой $\sigma_{ij}=0$ при $i\neq j$ и $\sigma_{ii}=\sigma_i\geqslant 0$. Величины σ_i называются *сингулярными числами* матрицы A, а столбцы матриц U и V — левыми и правыми *сингулярными векторами*.

Читатели, знакомые с теорией матричных собственных значений, отметят, что ненулевые собственные значения у матриц AA^T

и A^TA одни и те же и что сингулярные числа A суть положительные квадратные корни из этих собственных значений. Кроме того, левые и правые сингулярные векторы суть частные выборы собственных векторов для AA^T и A^TA соответственно.

На языке абстрактной линейной алгебры матрица A является представлением некоторого линейного оператора в конкретных координатных системах. Выполняя одно ортогональное преобразование координат в области определения оператора, а другое ортогональное преобразование в области его значений, превращаем представление в диагональное.

Чтобы упростить обозначения, предположим, что все прямоугольные матрицы имеют по крайней мере столько же строк, сколько столбцов, так что $m \geqslant n$. В применениях SVD к анализу экспериментальных данных матричный элемент a_{ij} представляет собой i-е наблюдение j-го переменного в эксперименте, следовательно, m — общее число наблюдений, а n — общее число переменных. Нам приходилось встречать задачи, для которых $m = 10\,000$, а $n=5$; однако наиболее распространены, по-видимому, задачи с $m=n$. Предположение $m \geqslant n$ означает, что имеется n сингулярных чисел σ_i, $i=1, \ldots, n$.

В сингулярном разложении имеется определенный произвол. Приведенное выше определение не фиксирует какой-либо конкретный порядок для сингулярных чисел. Даже если такой порядок указан, то столбцы матриц U и V, отвечающие кратным сингулярным числам, определены неоднозначно. Если проводить реальные вычисления по описываемой ниже подпрограмме SVD, задавая одну и ту же матрицу на различных машинах с разной длиной слова или даже только с различными программами извлечения квадратного корня, то могут быть получены совершенно разные матрицы U и V; однако результаты, вычисленные на любой машине, будут удовлетворять определению в пределах точности этой машины и давать удовлетворительные решения наших задач.

Приведем здесь пример с размерами 5×3; он будет использоваться на протяжении всей главы. (Ради экономии места значения элементов приводятся лишь с тремя десятичными знаками, хотя на практике работают обычно с бо́льшим числом знаков.)

Заметим, что из-за нулевых столбцов в Σ лишь самое большое первые n столбцов матрицы U могут действительно вносить вклад в произведение $U\Sigma V^T$. Более того, если некоторые из сингулярных чисел равны нулю, то нужны менее чем n столбцов U. Если k — количество ненулевых сингулярных чисел, то возможно сократить U до размеров $m\times k$, Σ — до размеров $k\times k$ и V^T — до размеров $k\times n$. Формально такие матрицы U и V не являются ортогональными, поскольку они не квадратные, однако их столбцы составляют ортонормированные системы векторов. Этот

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix},$$

$$U = \begin{pmatrix} 0.355 & -0.689 & 0.541 & 0.193 & 0.265 \\ 0.399 & -0.376 & -0.802 & -0.113 & 0.210 \\ 0.443 & -0.062 & 0.160 & -0.587 & -0.656 \\ 0.487 & 0.251 & -0.079 & 0.742 & -0.378 \\ 0.531 & 0.564 & 0.180 & -0.235 & 0.559 \end{pmatrix},$$

$$\Sigma = \begin{pmatrix} 35.127 & 0 & 0 \\ 0 & 2.465 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$V = \begin{pmatrix} 0.202 & 0.890 & 0.408 \\ 0.517 & 0.257 & -0.816 \\ 0.832 & -0.376 & 0.408 \end{pmatrix}.$$

вариант SVD можно назвать *экономичным*. Экономия машинной памяти может быть весьма значительна, если k или n гораздо меньше, чем m. Экономичный вариант нашего примера таков:

$$U = \begin{pmatrix} 0.355 & -0.689 \\ 0.399 & -0.376 \\ 0.443 & -0.062 \\ 0.487 & 0.251 \\ 0.531 & 0.564 \end{pmatrix},$$

$$\Sigma = \begin{pmatrix} 35.127 & 0 \\ 0 & 2.465 \end{pmatrix},$$

$$V = \begin{pmatrix} 0.202 & 0.890 \\ 0.517 & 0.257 \\ 0.832 & -0.376 \end{pmatrix}.$$

Понятие *ранга* матрицы имеет фундаментальное значение в большинстве разделов линейной алгебры. Согласно обычному

определению, это максимальное число независимых столбцов, или, что эквивалентно, максимальный порядок ненулевого минора матрицы. Основываясь на таком определении, трудно вычислять реально ранг матрицы общего вида. Однако если матрица диагональная, то ясно, что ее ранг равен числу ненулевых диагональных элементов. Если независимая система векторов умножается на ортогональную матрицу, то полученная система по-прежнему независима. Другими словами, ранг произвольной матрицы A равен рангу диагональной матрицы Σ в ее сингулярном разложении. Следовательно, практичным было бы определение ранга матрицы как количества ненулевых сингулярных чисел. Для обозначения ранга будем использовать букву k. (В примере $k=2$.)

Матрица размера $m \times n$, $m \geqslant n$, называется матрицей *полного ранга*, если $k=n$, или матрицей *недостаточного ранга*, если $k<n$. Для квадратных матриц часто вместо полного и недостаточного ранга соответственно используют более привычные термины: *невырожденная* и *вырожденная*.

Поскольку ранг матрицы всегда должен быть целым числом, он обязательно будет разрывной функцией элементов матрицы. Произвольно малые изменения (например, ошибки округлений) в матрице недостаточного ранга могут сделать все ее сингулярные числа ненулевыми и, следовательно, породить матрицу, формально имеющую полный ранг. На практике мы имеем дело с *эффективным рангом* — количеством сингулярных чисел, бóльших некой предписанной границы, которая отражает точность данных. Это также разрывная функция, но ее разрывы менее многочисленны и неприятны, чем у теоретического ранга.

Большим преимуществом при использовании SVD для определения ранга матрицы является то, что решения относительно возможного отбрасывания нужно выносить лишь в отношении отдельных чисел — малых сингулярных чисел,— но не векторов или систем векторов.

Мы можем теперь определить точно упомянутую раньше меру независимости. *Число обусловленности* матрицы A полного ранга есть отношение

$$\operatorname{cond}(A) = \frac{\sigma_{\max}}{\sigma_{\min}},$$

где $\sigma_{\max}$ и $\sigma_{\min}$ — наибольшее и наименьшее сингулярные числа A. Если A имеет недостаточный ранг, то $\sigma_{\min}=0$ и говорят, что число $\operatorname{cond}(A)$ бесконечно.

Число обусловленности, определенное в гл. 3 и оцениваемое подпрограммой DECOMP, основано на другой векторной норме, и потому его значения могут отличаться от значений определяемого здесь числа обусловленности. Число обусловленности из

гл. 3 легче вычислять, но зато оно не имеет стольких удобных свойств. Обычно, однако, оба числа обусловленности имеют сравнимую величину.

Ясно, что $\operatorname{cond}(A) \geqslant 1$. Если $\operatorname{cond}(A)$ близко к 1, то столбцы A *очень независимы*. Если $\operatorname{cond}(A)$ велико, то столбцы A *почти зависимы*. Если матрица A — квадратная, то таким терминам, как *почти вырожденная* или *далеко не вырожденная*, можно придать совершенно точное значение. Матрица A считается *более вырожденной*, чем матрица B, если $\operatorname{cond}(A) > \operatorname{cond}(B)$.

Если матрица A — ортогональная, то $\operatorname{cond}(A) = 1$, так что столбцы ортогональной матрицы настолько независимы, насколько это вообще возможно. Обратно, если $\operatorname{cond}(A) = 1$, то оказывается, что A лишь числовым множителем отличается от ортогональной матрицы.

Длина или *норма вектора* определяется как

$$\|x\| = (x^T x)^{1/2}.$$

Отношение $\|Ax\|/\|x\|$ измеряет степень, в которой A *растягивает* x. Как изменяется коэффициент растяжения $\|Ax\|/\|x\|$, когда x пробегает все (ненулевые) векторы? Пусть $A = U\Sigma V^T$ и $y = V^T x$. Так как ортогональные матрицы сохраняют длину, то $\|y\| = \|x\|$ и

$$\|Ax\| = \|U\Sigma Y^T x\| = \|\Sigma y\|.$$

Поскольку Σ — диагональная матрица, ясно, что

$$\sigma_{\min}\|y\| \leqslant \|\Sigma y\| \leqslant \sigma_{\max}\|y\|.$$

Следовательно,

$$\sigma_{\min} \leqslant \frac{\|Ax\|}{\|x\|} \leqslant \sigma_{\max}.$$

Сингулярные числа можно интерпретировать геометрически. Матрица A отображает единичную сферу, т. е. множество векто-

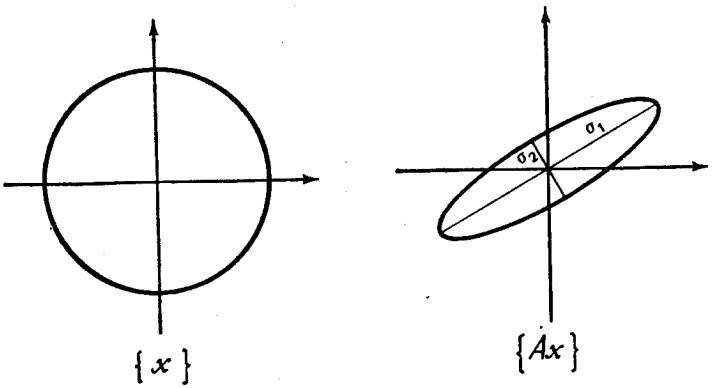


Рис. 9.1.

ров, для которых $\|x\|=1$, в множество векторов $b=Ax$ с различными длинами. Это множество-образ является в действительности k-мерным эллипсоидом, вложенным в m-мерное пространство. Ситуация для $m=n=k=2$ показана на рис. 9.1. Сингулярные числа суть длины разных осей эллипсоида. Крайние сингулярные числа σ_{max} и σ_{min} — это длины наибольшей и наименьшей оси. Число обусловленности связано с эксцентриситетом эллипсоида, причем большие числа обусловленности соответствуют очень вытянутым эллипсоидам.

Норму матрицы можно определить как максимальный коэффициент растяжения, т. е. попросту как

$$\|A\| = \sigma_{max}.$$

Читатель, возможно, вспомнит, что в гл. 3 мы использовали другое определение для $\|A\|$. Определенную там норму гораздо легче вычислять, чем σ_{max}, однако она не столь хорошо приспособлена к задачам того типа, с которым мы имеем дело в этой главе.

9.3. Приложения

В этом параграфе будут описаны еще несколько применений сингулярного разложения.

Решение произвольной системы линейных уравнений

Пусть A — заданная $m\times n$-матрица, причем $m\geqslant n$, а b — заданный вектор размерности m. Нужно найти все векторы x, для которых

$$Ax=b.$$

Заметим, что важный случай, когда матрица A квадратная, но, возможно, вырожденная, включается сюда. Эта задача связана с такими вопросами: Совместны ли уравнения? Существуют ли решения? Единственно ли решение? Имеет ли система $Ax=0$ ненулевые решения? Каков общий вид решения?

Теоретически имеется много различных алгоритмов, которые отвечают на эти вопросы. Но в вычислительной практике с ее неточными данными и неточной арифметикой SVD по существу является единственным известным методом, на надежность которого можно положиться.

Используя сингулярное разложение матрицы A, линейную систему $Ax=b$ можно переписать в виде

$$U\Sigma V^T x=b,$$

откуда

$$\Sigma z = d,$$

где $z = V^T x$ и $d = U^T b$. Система уравнений $\Sigma z = d$ диагональная и, следовательно, легко может быть решена. Она может распадаться на две или три подсистемы в зависимости от значений размерностей m и n и ранга k, т. е. количества ненулевых сингулярных чисел:

$$\begin{aligned} \sigma_j z_j &= d_j, \quad \text{если} \quad j \leqslant n \quad \text{и} \quad \sigma_j \neq 0, \\ 0 \cdot z_j &= d_j, \quad \text{если} \quad j \leqslant n \quad \text{и} \quad \sigma_j = 0, \\ 0 &= d_j, \quad \text{если} \quad j > n. \end{aligned}$$

Вторая подсистема пуста, если $k=n$; третья пуста, если $n=m$.

Уравнения совместны и решение существует в том и только том случае, когда $d_j=0$ всякий раз, когда $\sigma_j=0$ или $j>n$. Если $k<n$, то неизвестному z_j, отвечающему нулевому σ_j, можно придать произвольное значение, и все равно получится решение системы. После возвращения к исходным координатам посредством преобразования $x=Vz$ эти произвольные компоненты z позволяют параметризовать пространство всех возможных решений x.

Обозначим через u_j и v_j столбцы матриц U и V. Тогда разложение $A=U\Sigma V^T$ можно записать так:

$$Av_j = \sigma_j u_j, \quad j = 1, \ldots, n.$$

(Поскольку в основном уравнении, вводящем сингулярное разложение, мы снабдили матрицу V индексом T, то теперь в уравнение входят столбцы обеих матриц U и V, а не строки одной и столбцы другой.) *Ядро* матрицы A есть множество векторов x, для которых $Ax=0$, а *область значений* A — это множество векторов b, для которых система $Ax=b$ имеет решение. Если $\sigma_j=0$, то $Av_j=0$ и v_j принадлежит ядру A; если же $\sigma_j\neq 0$, то u_j принадлежит области значений A.

Исходя из этого, мы можем получить полное описание ядра и области значений следующим образом. Пусть V_0 — система столбцов v_j, для которых $\sigma_j=0$, а V_1 — система остальных столбцов v_j. Пусть U_1 — система столбцов u_j, для которых $\sigma_j\neq 0$, а U_0 — система прочих столбцов u_j, включая те, для которых $j>n$. В системе V_0 имеется $n-k$ столбцов, в V_1 — k столбцов и столько же в U_1, наконец, в U_0 имеется $m-k$ столбцов. Кроме того,

1. V_0 — ортонормированный базис для ядра A.
2. V_1 — ортонормированный базис для ортогонального дополнения ядра A.
3. U_1 — ортонормированный базис для области значений A.
4. U_0 — ортонормированный базис для ортогонального дополнения области значений A.

Указанная ранее матрица

$$A_1 = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}.$$

имеет недостаточный ранг. Ее средний столбец есть полусумма двух других. Это отражается в том факте, что в последнем столбце матрицы V, равном $(0.408, -0.816, 0.408)^T$, компоненты находятся в отношении $1 : -2 : 1$. Система $Ax=b$ имеет решение тогда и только тогда, когда b есть линейная комбинация первых двух столбцов U. Если решение существует, то к нему можно добавить произвольное кратное последнего столбца V.

Например, возьмем

$$b = \begin{pmatrix} 5 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}.$$

Тогда

$$d = U^T b = \begin{pmatrix} 11.071 \\ -1.561 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Последние три компоненты вектора d — нули; следовательно, столбец b был линейной комбинацией первых двух столбцов U, система $Ax=b$, равно как и система $\Sigma z=d$, совместна. Эта последняя имеет вид

$$\begin{aligned} 35.127 z_1 &= 11.071, \\ 2.465 z_2 &= -1.561, \\ 0 z_3 &= 0. \end{aligned}$$

Ее решение —

$$\begin{aligned} z_1 &= 0.315, \\ z_2 &= -0.633, \\ z_3 &\text{ произвольно.} \end{aligned}$$

Беря $z_3=0$ и вычисляя $x=Vz$, получаем

$$x=\begin{pmatrix}-0.500\\0\\0.500\end{pmatrix}.$$

Легко видеть, что $x=(-{}^1/_2,\ 0,\ {}^1/_2)^T$ в самом деле является решением системы $Ax=b$. С другой стороны, взяв $z_3=1.225$, находим

$$x=\begin{pmatrix}-1.091\\1.002\\-0.001\end{pmatrix},$$

и $x=(-1,\ 1,\ 0)^T$ — также решение. В действительности мы можем, используя последний столбец V, получить общее решение.

$$x=\begin{pmatrix}-0.500\\0\\0.500\end{pmatrix}+z_3\begin{pmatrix}0.408\\-0.816\\0.408\end{pmatrix}$$

$$=\frac{1}{2}\begin{pmatrix}-1\\0\\1\end{pmatrix}+\alpha\begin{pmatrix}1\\-2\\1\end{pmatrix}.$$

Выбор $z_3=0$ имеет особый смысл. Он приводит к решению $(-{}^1/_2,\ 0,\ {}^1/_2)^T$, имеющему среди всех возможных решений наименьшую длину.

В качестве другого примера возьмем

$$b=\begin{pmatrix}4\\5\\5\\5\\5\end{pmatrix}.$$

Тогда

$$d=U^Tb=\begin{pmatrix}10.716\\-0.872\\-0.541\\-0.193\\-0.265\end{pmatrix}.$$

Так как три последние компоненты d не равны нулю, то система $Ax=b$ не имеет решений. Другими словами, b не принадлежит области значений A.

Линейная задача наименьших квадратов

Это обобщение предыдущей задачи, но теперь ищутся n-мерные векторы x, для которых Ax лишь приближенно равно b, именно, в том смысле, что длина невязки минимальна. Под невязкой здесь понимается m-мерный вектор

$$r=Ax-b.$$

Итак, задача состоит в том, чтобы выбрать x, который минимизирует $\|r\|^2=\sum_{i=1}^{m} r_i^2$. (Минимизация квадрата длины равносильна минимизации самой длины.)

Эта задача на языке статистиков называется задачей *линейной регрессии*. Матрица A обычно обозначается через X и именуется *матрицей плана*. Правую часть b обычно обозначают через y и называют *вектором данных*. Разумеется, это та же задача, которая рассматривалась в § 9.1. Теперь используется лишь больше матричной терминологии.

Если A имеет полный ранг, то решение x единственно и может быть устойчиво вычислено несколькими различными алгоритмами, из которых некоторые быстрее, чем SVD. Но SVD справляется и со случаем недостаточного ранга и, за исключением очень больших задач, ненамного более трудоемко, чем прочие устойчивые методы. (Во всяком случае оно предпочтительней быстрого алгоритма, дающего, возможно, неверный результат.)

Поскольку ортогональные матрицы сохраняют норму, то

$$\|r\|=\|U^T(AVV^Tx-b)\|=\|\Sigma z-d\|.$$

Следовательно, SVD сводит общую задачу наименьших квадратов к задаче с диагональной матрицей. Легко видеть, что вектор z, дающий минимум для $\|r\|$, выражается соотношениями

$$z_j=\frac{d_j}{\sigma_j}, \text{ если } \sigma_j\neq 0.$$

$$z_j \text{ произвольно, если } \sigma_j=0.$$

Таким образом, k из уравнений диагональной формы решаются точно. Остальные уравнения приводят к тому, что вектор-невязка, возможно, отличен от нуля, причем его норма равна $\|r\|^2=\sum d_j^2$, где сумма берется по всем j, для которых $\sigma_j=0$ либо $j>n$. Обратное преобразование $x=Vz$ решает тогда исходную задачу.

Если задача имеет недостаточный ранг, то решение, минимизирующее $\|r\|$, неединственно. В такой ситуации часто бывает желательно добиться единственности, выбирая решение наименьшей длины. Это достигается, если положить

$$z_j=0, \quad \text{когда} \quad \sigma_j=0.$$

В случае полного ранга решение единственно.

Иногда рассматривают модифицированные задачи наименьших квадратов, в которых минимизируется некоторая комбинация $\|r\|$ и $\|x\|$. Решения таких задач легко получаются из SVD посредством методов, описанных в книге Лоусон, Хэнсон (1974).

В качестве примера снова рассмотрим задачу с

$$b = \begin{pmatrix} 4 \\ 5 \\ 5 \\ 5 \\ 5 \end{pmatrix}.$$

Мы уже видели, что система $Ax=b$ несовместна и, следовательно, не может быть решена точно. Диагонализованные уравнения таковы:

$$\begin{aligned} 35.127z_1 &\approx 10.716, \\ 2.465z_2 &\approx -0.872, \\ 0z_3 &\approx -0.541, \\ 0 &\approx -0.193, \\ 0 &\approx -0.265. \end{aligned}$$

Символ $\approx$ означает, что эти уравнения нужно решить в смысле наименьших квадратов. Первые два можно решить точно, полагая

$$\begin{aligned} z_1 &= 0.305, \\ z_2 &= -0.354. \end{aligned}$$

Третье уравнение не определяет z_3; возьмем

$$z_3 = 0,$$

чтобы получить решение наименьшей длины. Последние три уравнения показывают, что $\|r\|^2$ будет равна $0.541^2+0.193^2+0.265^2 = 0.400$. Заметим, что это можно вычислить без фактического нахождения x; однако несколько более точный способ, особенно в случае малых невязок, состоит в вычислении x, а затем $\|b-Ax\|$.

Решение $x=Vz$ есть

$$x = \begin{pmatrix} -0.253 \\ 0.067 \\ 0.387 \end{pmatrix}.$$

Тогда

$$Ax = \begin{pmatrix} 4.406 \\ 4.607 \\ 4.808 \\ 5.009 \\ 5.210 \end{pmatrix},$$

что довольно близко к заданному вектору b. Опять-таки к вектору x можно прибавлять любые кратные последнего столбца матрицы V, получая другие решения задачи наименьших квадратов, имеющие, однако, бóльшую длину.

Псевдообратная матрица

Матрица X с размерами $n \times m$ называется псевдообратной Мура — Пенроуза для матрицы A, если она удовлетворяет следующим четырем условиям:

1. $AXA = A$.
2. $XAX = X$.
3. AX симметрична.
4. XA симметрична.

Можно показать, что такая матрица X всегда существует и единственна. Если A — квадратная невырожденная матрица, то $X = A^{-1}$, очевидно, удовлетворяет этим условиям. Если A — прямоугольная и имеет полный ранг, то $X = (A^T A)^{-1} A^T$. Если A не имеет полного ранга, то такого простого представления нет. Для псевдообратной матрицы часто используется обозначение $A^\dagger$.

Чтобы применить SVD для вычисления псевдообратной матрицы, начнем со случая 1×1, т. е. с псевдообращения числа:

$$\sigma^\dagger = \begin{cases} \frac{1}{\sigma}, & \text{если } \sigma \neq 0, \\ 0, & \text{если } \sigma = 0. \end{cases}$$

Заметим, что $\sigma\sigma^\dagger = 0$ либо 1 и что четыре условия Мура — Пенроуза выполняются очевидным образом.

Теперь можно проверить, что псевдообратной к диагональной $m \times n$-матрице

$$\Sigma = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & 0 \\ & & \ddots & \\ & 0 & & \\ & & & \sigma_n \\ & & 0 & \end{pmatrix}$$

будет диагональная $n\times m$-матрица

$$\Sigma^{\dagger}=\begin{pmatrix}\sigma_1^{\dagger} & & & \\ & \sigma_2^{\dagger} & & 0 \\ & & \ddots & \\ & 0 & & \ddots \\ & & & & \sigma_n^{\dagger} & \cdot \end{pmatrix}$$

Отметим, что $\Sigma\Sigma^{\dagger}=I_k$, т. е. это матрица с k единицами на диагонали и нулями в прочих позициях.

Наконец, используя трехчленный аналог известного тождества $(AB)^{-1}=B^{-1}A^{-1}$, можно получить псевдообратную произвольной матрицы из ее сингулярного разложения по формуле

$$A^{\dagger}=V\Sigma^{\dagger}U^{T}.$$

Легко проверить, что при таком определении четыре уравнения Мура — Пенроуза выполняются.

Если матрица A — квадратная и невырожденная, то все ее сингулярные числа отличны от нуля и, следовательно, $\Sigma^{\dagger}=\Sigma^{-1}$ и $A^{\dagger}=A^{-1}$.

Псевдообратная матрица связана с задачей наименьших квадратов тем обстоятельством, что вектор x наименьшей длины, минимизирующий $\|Ax-b\|$, можно выразить формулой $x=A^{\dagger}b$.

Если основное соотношение

$$A=U\Sigma V^{T}$$

расписать поэлементно, то для $i=1, \ldots, m$ и $j=1, \ldots, n$ справедливо

$$a_{ij}=\sum_{k=1}^{n}\sigma_k u_{ik}v_{jk}.$$

Конечно, суммирование можно ограничить теми значениями k, для которых $\sigma_k\neq 0$. Если определение

$$A^{\dagger}=V\Sigma^{\dagger}U^{T}$$

также расписать, то для $i=1, \ldots, n$ и $j=1, \ldots, m$

$$a_{ij}^{\dagger}=\sum_{\sigma_k\neq 0}\frac{v_{ik}u_{jk}}{\sigma_k}.$$

Здесь ограничение суммы на ненулевые σ_k становится необходимым.

Нужно подчеркнуть, что элементы псевдообратной матрицы не являются непрерывными функциями элементов данной матрицы. В псевдообратной для меняющейся матрицы будут *скачки* всякий

раз, когда матрица меняет ранг. Поэтому в вычислительной практике важно сравнивать малые сингулярные числа с некоторой границей, которая отражает ошибки в исходных данных и вычислениях. *Эффективную псевдообратную* матрицу можно определить посредством такой границы τ следующим образом:

$$\sigma_\tau^\dagger = \begin{cases} \dfrac{1}{\sigma}, & \text{если } \sigma > \tau, \\ 0 & \text{в противном случае.} \end{cases}$$

Это приводит к матрице $A_\tau^\dagger$, имеющей элементы

$$a_{ij}^\dagger = \sum_{\sigma_k > \tau} \frac{v_{ik} u_{jk}}{\sigma_k}.$$

Матрица $X = A_\tau^\dagger$ удовлетворяет лишь трем последним условиям Мура — Пенроуза, т. е.

$$XAX = X;$$
$$AX \text{ симметрична;}$$
$$XA \text{ симметрична.}$$

Первое условие заменяется неравенством

$$\|AXA - A\| \leqslant \tau,$$

где используется определенная выше норма. Если $\tau \neq 0$, то матрица X, удовлетворяющая этим четырем соотношениям, может быть неединственной; однако $X = A_\tau^\dagger$ имеет то дополнительное свойство, что она минимизирует $\|X\|$.

Именно эта способность учитывать неточность входных данных и приближенность вычислений делает характеризацию псевдообратной матрицы посредством сингулярного разложения предпочтительней других подходов.

Возвращаясь к нашему примеру, находим

$$A^\dagger = \begin{pmatrix} -0.247 & -0.133 & -0.020 & 0.093 & 0.207 \\ -0.067 & -0.033 & 0.000 & 0.033 & 0.067 \\ 0.113 & 0.067 & 0.020 & -0.027 & -0.073 \end{pmatrix}.$$

Заметим, что в этом примере $A^\dagger A$ отнюдь не похожа на единичную.

Аппроксимирующие матрицы

Если A — умеренно большая матрица, например если m и n имеют порядок нескольких сотен, то хранение всех элементов A в машине может потребовать значительной доли всей доступной памяти. Может оказаться желательным аппроксимировать матри-

цу A более «простой» матрицей, требующей меньше памяти. Разумно понимать под *простой* матрицей матрицу сравнительно невысокого ранга, т. е. имеющую лишь несколько независимых столбцов. Кроме экономии памяти, такие матрицы могут давать важную информацию относительно конкретных задач.

Уравнение $A=U\Sigma V^T$ можно переписать в виде

$$A=\sigma_1 E_1+\sigma_2 E_2+\ldots+\sigma_n E_n,$$

где $E_i=u_i v_i^T$ есть *внешнее произведение* столбца матрицы U и соответствующего столбца матрицы V. Каждая E_i — это простая матрица ранга 1, для хранения которой требуется лишь $m+n$ вместо mn ячеек памяти. Кроме того, произведение $E_i x$ можно сформировать для любого вектора x всего лишь за $m+n$ умножений, а не за mn. Чтобы оценить степень экономии, заметим, что для $m=n=300$ будет $m+n=600$, в то время как $mn=90\,000$.

Сумма квадратов всех элементов матрицы E_i равна 1, так что ни одна из этих матриц не является численно более или менее важной, чем любая другая. Зато некоторые из коэффициентов σ_i могут быть значительно меньше других, и потому написанный ряд для A можно оборвать после нескольких членов. Сокращенная сумма определяет приближение к A, для которого, однако, хранение и оперирование обходятся дешевле. Разумеется, полезность этой процедуры зависит от конкретной матрицы A и распределения ее сингулярных чисел.

Недавно подобные приближения были с успехом применены в таких различных областях, как цифровая обработка образов и криптография; см. статьи Эндрюс, Паттерсон (1975) и Моулер, Моррисон (1977).

Вычисление определителей и нахождение вырожденных матриц

Пусть $A(z)$ — квадратная матрица, зависящая, быть может, сложным образом, от некоторого параметра z. Рассмотрим задачу нахождения того значения или тех значений z, для которых $A(z)$ вырождена. Другими словами, найти z, для которых определитель $A(z)$ равен нулю.

Используя символ det для обозначения определителя, можем написать

$$\det(A)=\det(U)\cdot\det(\Sigma)\cdot\det(V^T).$$

Определитель ортогональной матрицы равен ± 1, а определитель диагональной матрицы есть просто произведение ее диагональных элементов. Таким образом,

$$\det(A)=\pm\sigma_1\cdot\sigma_2\cdot\ldots\cdot\sigma_n.$$

Вычисление определителя может оказаться непростым делом, потому что значения его меняются в очень широком интервале, что часто приводит к переполнению либо машинному нулю.

Для многих задач, формулируемых в терминах определителей, не требуется фактическое значение определителя, а лишь некоторое указание на то, когда он равен нулю. Теоретически возможно вычислить $|\det(A)|$, беря произведение сингулярных чисел A. Однако в большинстве ситуаций достаточно использовать лишь наименьшее сингулярное число, избегнув тем самым переполнений, машинных нулей и прочих вычислительных затруднений.

Можно организовать систематический поиск значения z, которое приносит минимум (надо надеяться, нулевой) функции $\sigma_{max}(A(z))$; для этой цели можно использовать универсальные программы минимизации, такие, как FMIN. Мы с большим успехом применяли эту технику для весьма разнообразных задач.

В принципе можно изменить алгоритм SVD таким образом, чтобы произведение $\det(U)\cdot\det(V^T)$ всегда имело фиксированный знак $+$ или $-$. Тогда станет возможно находить знак определителя матрицы $A(z)$ и использовать программы вычисления нулей вроде ZEROIN, а не программы минимизации. У нас не было большой практики в реализации этой идеи.

Проблема собственных значений

В этом пункте будет описано не реальное приложение SVD, а скорее указание на то, как сингулярное разложение связано с более широким классом важных матричных задач.

Пусть A — квадратная матрица. Собственные значения A — это числа λ, для которых

$$Ax=\lambda x$$

имеет ненулевое решение x. Поскольку это эквивалентно требованию, чтобы

$$\det(A-\lambda I)=0,$$

то теоретически собственные значения можно находить как корни такого полинома. Рассмотрим, однако, матрицу

$$A=\begin{pmatrix}1 & e\\ e & 1\end{pmatrix},$$

где e — некоторое малое число, которым все же нельзя пренебречь. Правильные собственные значения суть $\lambda_1=1+e$ и $\lambda_2=1-\varepsilon$. Полином $\det(A-\lambda I)$ равен

$$\lambda^2-2\lambda+(1-e^2).$$

Если e мало, то e^2 много меньше и в коэффициентах полинома необходимо иметь удвоенную точность по сравнению с точностью

матричных элементов или собственных значений. Эта трудность становится еще более явственной для матриц большего порядка. Поэтому современные методы вычисления собственных значений избегают использования полиномов или алгоритмов для нахождения корней полиномов.

Связь между сингулярным разложением и собственными значениями наиболее проста для матриц, являющихся симметричными и положительно полуопределенными. Симметрию легко определить и проверить: A симметрична, если $a_{ij}=a_{ji}$ для всех i и j. Положительная полуопределенность — это гораздо более интимное свойство; она означает, что $x^TAx\geqslant 0$ для всех x. Грубо говоря, это значит, что диагональные элементы A довольно велики по сравнению с внедиагональными элементами. Действительно, достаточным условием будет выполнение для каждого i неравенства

$$a_{ii}\geqslant \sum_{j\neq i} |a_{ij}|,$$

однако эти неравенства не являются необходимыми.

Нетрудно показать, что для действительной симметричной матрицы A все собственные значения — действительные числа, а если A положительно полуопределена, то все ее собственные значения неотрицательны. Если $A=U\Sigma V^T$ и $A=A^T$, то

$$\begin{aligned} A^2 = A^TA &= V\Sigma^TU^TU\Sigma V^T \\ &= V\Sigma^2V^T. \end{aligned}$$

Отсюда

$$A^2V = V\Sigma^2.$$

Обозначим через v_j столбцы матрицы v; рассматривая j-й столбец полученного уравнения, находим

$$A^2v_j = \sigma_j^2v_j.$$

Поскольку собственные значения матрицы A^2 суть квадраты собственных значений A, а эти последние неотрицательны, то мы заключаем, что собственные значения симметричной положительно полуопределенной матрицы равны ее сингулярным числам, а собственными векторами будут столбцы V.

Если A симметричная, но не положительно полуопределенная, то некоторые из ее собственных значений отрицательны. Оказывается, что модули собственных значений равны сингулярным числам, а знаки собственных значений можно найти, сравнивая столбцы U со столбцами V; мы, однако, не будем вдаваться в детали. Если A не является симметричной, то между ее собственными значениями и сингулярными числами нет какой-либо простой связи.

Все это показывает, что алгоритм SVD можно использовать для вычисления собственных значений симметричных матриц.

Однако такой способ не особенно эффективен, и мы не рекомендуем его. Нашей единственной целью было продемонстрировать, что сингулярное разложение тесно связано с матричными задачами на собственные значения. Тщательное изучение алгоритма SVD поможет также и в понимании алгоритмов для собственных значений.

9.4. Вычисление разложения

SVD — это также наименование подпрограммы для вычисления сингулярного разложения произвольной матрицы. Она основана на АЛГОЛ-процедуре Голуба и Райнша (Голуб, Райнш (1971)).

Прежде чем описывать, как работает SVD, полезно рассмотреть более элементарный матричный алгоритм — гауссово исключение — с несколько иной точки зрения.

Пусть

$$A_1 = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}.$$

Если бы A_1 была матрицей коэффициентов системы линейных уравнений, то первым шагом в решении системы было бы вычитание кратных первой строки из других строк с целью получить нули в первом столбце. (В этом простом примере нет необходимости выбирать ведущий элемент.) Мы получили бы

$$A_2 = \begin{pmatrix} 1 & 6 & 11 \\ 0 & -5 & -10 \\ 0 & -10 & -20 \\ 0 & -15 & -30 \\ 0 & -20 & -40 \end{pmatrix}.$$

Как можно описать преобразование A_1 в A_2, используя матричные обозначения? Пусть

$$m_1 = \begin{pmatrix} 0 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}$$

$$e_1^T = (1 \quad 0 \quad 0 \quad 0 \quad 0).$$

С помощью внешнего произведения $m_1 e_1^T$ образуем матрицу $M_1 = I - m_1 e_1^T$:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 \\ -3 & 0 & 1 & 0 & 0 \\ -4 & 0 & 0 & 1 & 0 \\ -5 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Легко проверить, что

$$A_2 = M_1 A_1.$$

Конечно, подпрограмма гауссова исключения в действительности не формирует матрицу M_1 и не выполняет матричного умножения $M_1 A_1$. Она просто оперирует со строками A_1, получая из нее A_2. Однако математическое описание этого процесса компактней всего в терминах матрицы M_1. Кроме того, введение M_1 позволяет рассматривать преобразование с точки зрения операций над столбцами A_1, а не над ее строками. Пусть a_j есть j-й столбец A. Тогда

$$\begin{aligned} M_1 a_j &= (I - m_1 e_1^T) a_j \\ &= a_j - (m_1 e_1^T) a_j \\ &= a_j - m_1 (e_1^T a_j) \\ &= a_j - (a_{1j}) m_1. \end{aligned}$$

В частности,

$$M_1 a_1 = a_1 - 1 \cdot m_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$M_1 a_2 = a_2 - 6 \cdot m_1 = \begin{pmatrix} 6 \\ -5 \\ -10 \\ -15 \\ -20 \end{pmatrix},$$

$$M_1 a_3 = a_3 - 11 \cdot m_1 = \begin{pmatrix} 11 \\ -10 \\ -20 \\ -30 \\ -40 \end{pmatrix}.$$

Это показывает, что столбцы A_2 можно получить путем вычитания некоторых кратных вектора m_1 из столбцов A_1.

Следующий шаг исключения даст матрицу

$$A_3 = \begin{pmatrix} 1 & 6 & 11 \\ 0 & -5 & -10 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Этот шаг можно описать с помощью

$$m_2 = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 3 \\ 4 \end{pmatrix},$$

$$e_2^T = (0\ 1\ 0\ 0\ 0)$$

и

$$M_2 = I - m_2 e_2^T,$$
$$A_3 = M_2 A_2.$$

Алгоритм SVD основан на аналогичных, хотя и несколько более сложных идеях. Реальная подпрограмма выполняет различные операции как над строками, так и над столбцами матрицы. Наиболее прозрачное описание алгоритма использует ортогональные аналоги матриц M_1 и M_2.

Цель алгоритма состоит в том, чтобы найти ортогональные матрицы U и V, такие, что матрица

$$U^T A V = \Sigma$$

диагональная. Обе эти матрицы получаются как произведения ортогональных матриц, называемых хаусхолдеровыми отражениями.

Алгоритм распадается на два этапа. Первый этап заключается в приведении A к *двухдиагональной форме*, т. е. матрице, у которой ненулевыми могут быть лишь элементы диагонали и первой наддиагонали. Второй этап — это итерационный процесс, в котором наддиагональные элементы уменьшаются до пренебрежимо малой величины, что дает желаемую диагональную матрицу.

Снова рассмотрим введение нулей в a_1 — первый столбец матрицы A_1. Гауссово исключение использовать нельзя, поскольку M_1 неортогональная. Возьмем вектор

$$u_1 = \begin{pmatrix} 8.416 \\ 2.000 \\ 3.000 \\ 4.000 \\ 5.000 \end{pmatrix}.$$

Он получается из a_1 прибавлением к первой компоненте числа $\|a_1\|$. Пусть

$$\beta_1 = \frac{\|u_1\|^2}{2} = 62.415,$$
$$U_1 = I - \beta_1^{-1} u_1 u_1^T,$$
$$A_2 = U_1 A_1.$$

Преобразованная матрица A_2 действительно вычисляется столбец за столбцом:

$$\begin{aligned} U_1 a_1 &= (I - \beta_1^{-1} u_1 u_1^T)\, a_1 \\ &= a_1 - \beta_1^{-1} u_1 u_1^T a_1 \\ &= a_1 - u_1 \quad (\text{так как } \beta_1^{-1} u_a^T a_1 = 1) \\ &= \begin{pmatrix} -7.416 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Заметим, что в первый столбец были введены желаемые нули и что его длина не изменилась.

$$
\begin{aligned}
U_1 a_2 &= a_2 - \beta_1^{-1} u_1 u_1^T a_2 \\
&= a_2 - (2.796) u_1 \\
&= \begin{pmatrix} -17.529 \\ 1.409 \\ -0.387 \\ -2.183 \\ -3.949 \end{pmatrix},
\end{aligned}
$$

$$
\begin{aligned}
U_1 a_3 &= a_3 - \beta_1^{-1} u_1 u_1^T a_3 \\
&= a_3 - (4.592) u_1 \\
&= \begin{pmatrix} -27.642 \\ 2.817 \\ -0.774 \\ -4.366 \\ -7.957 \end{pmatrix}.
\end{aligned}
$$

Следовательно,

$$
A_2 = \begin{pmatrix} -7.416 & -17.529 & -27.642 \\ 0 & 1.409 & 2.817 \\ 0 & -0.387 & -0.774 \\ 0 & -2.183 & -4.366 \\ 0 & -3.979 & -7.957 \end{pmatrix}.
$$

Столбцы A_2 получаются вычитанием кратных вектора u_1 из столбцов A_1. Эти кратные порождаются скалярными произведениями, а не отдельными элементами, как в гауссовом исключении.

Преобразующая матрица U_1 и есть хаусхолдерово отражение. В ее ортогональности, критически важной для процесса, можно убедиться, проверив, что $U_1^T U_1 = I$:

$$
\begin{aligned}
U_1^T U_1 &= (I - \beta_1^{-1} u_1 u_1^T)^T (I - \beta_1^{-1} u_1 u_1^T) \\
&= I - 2\beta_1^{-1} u_1 u_1^T + \beta_1^{-2} u_1 u_1^T u_1 u_1^T \\
&= I - 2\beta_1^{-1} u_1 u_1^T + 2\beta_1^{-1} u_1 u_1^T \\
&= I.
\end{aligned}
$$

Эта ортогональность гарантирует, что длина каждого столбца в A_2 равна длине соответствующего столбца в A_1.

Геометрически преобразование любого столбца a_j в $U_1 a_j$ состоит в отражении a_j относительно плоскости, проходящей через

начало координат и перпендикулярной к u_1. Эта плоскость выбрана так, что a_1 отражается в вектор с четырьмя нулевыми компонентами.

Прежде чем вводить дальнейшие нули под диагональю, преобразованием вида

$$A_3 = A_2 V_1$$

получают нули в первой строке. Нули, уже стоящие в первом столбце, не должны быть испорчены, а длину первой строки нужно сохранить, так что в данном примере, где n равно всего лишь 3, можно получить только один нуль. Для матрицы большего порядка на этом шаге было бы получено $n-2$ нуля.

Преобразование порождается первой строкой A_2:

$$\begin{aligned} v_1^T &= (0 \quad -50.261 \quad -27.642), \\ \gamma_1 &= \frac{\|v_1\|^2}{2}, \\ V_1 &= I - \gamma_1^{-1} v_1 v_1^T, \\ A_3 &= A_2 V_1. \end{aligned}$$

Строка матрицы A_3 с номером i получается по формуле

$$\begin{aligned} a_i^T V_1 &= a_i^T (I - \gamma_1^{-1} v_1 v_1^T) \\ &= a_i^T - (\gamma_1^{-1} a_i^T v_1) v_1^T. \end{aligned}$$

Таким образом, из каждой строки A_2 вычитается надлежащее кратное v_1^T. Это дает матрицу

$$A_3 = \begin{pmatrix} -7.416 & 32.732 & 0 \\ 0 & -3.133 & 0.319 \\ 0 & 0.861 & -0.088 \\ 0 & 4.856 & -0.495 \\ 0 & 8.850 & -0.901 \end{pmatrix}$$

Поскольку первая компонента v_1^T нулевая, то нули первого столбца A_2 сохраняются в A_3. Так как V_1 ортогональная, то длина каждой строки в A_3 равна длине соответствующей строки в A_2.

Теперь можно добиться новых нулей под диагональю, не испортив полученных ранее:

$$A_4 = U_2 A_3 = \begin{pmatrix} -7.416 & 32.732 & 0 \\ 0 & 10.605 & -1.080 \\ 0 & 0 & 0.000 \\ 0 & 0 & 0.000 \\ 0 & 0 & 0.000 \end{pmatrix}.$$

Поскольку ранг этой матрицы равен лишь 2, то теперь третий столбец имеет на диагонали и под диагональю элементы порядка ошибки округления. Эти элементы обозначены в матрице через 0.000, чтобы отличить их от элементов, в точности равных нулю. Если бы матрица имела полный ранг, то нужно было бы выполнить еще одно преобразование, чтобы получить нули в третьем столбце:

$$A_5 = U_3 A_4 = \begin{pmatrix} -7.416 & 32.732 & 0 \\ 0 & 10.605 & 1.080 \\ 0 & 0 & 0.000 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

В данном примере третий диагональный элемент был бы точным нулем, если бы не ошибки округлений. Элементы под диагональю во всех столбцах указаны как точные нули, потому что преобразования так и строились, чтобы получить там нули. Последнее преобразование U_3 в этом примере могло бы быть тождественным, однако тогда оно не будет хаусхолдеровым отражением. Фактически использованное U_3 попутно изменяет знак элемента — 1.080 в матрице A_4.

Получилась искомая двухдиагональная матрица, и первый этап закончен. Прямое использование ортогональных преобразований не позволяет получить какие-либо новые нули. Для общего порядка n нужно n преобразований U и $n-2$ преобразований V, чтобы достигнуть этого места. Число преобразований не зависит от строчной размерности m, но от m зависит работа, затрачиваемая на выполнение каждого преобразования.

Второй этап алгоритма SVD представляет собой итерационный процесс. Каждый шаг уменьшает величину изгоняемых наддиагональных элементов. В конечном счете эти элементы становятся сравнимыми с ошибками округлений, и их без ущерба можно отбросить. Фактическое число шагов зависит от конкретной матрицы и точности машины, но скорость сходимости очень высокая, так что обычно на выполнение второго этапа требуется меньше времени, чем на выполнение первого. Это особенно верно, если m много больше, чем n, поскольку на втором этапе сохраняются все нули, полученные ниже первых n строк.

Второй этап есть вариант QR-алгоритма, разработанного Фрэнсисом [1]) около 1960 г. Этот метод стал одним из наиболее надежных и широко используемых численных алгоритмов. Полное

[1]) QR-алгоритм (под названием метода односторонних вращений) был одновременно и независимо предложен В. Н. Кублановской.— *Прим. перев.*

обсуждение его деталей и, в частности, удовлетворительное объяснение того, почему он работает, выходят за рамки этой книги. Мы лишь кратко обрисуем поведение алгоритма для рассматриваемого примера. Интересующемуся читателю советуем за деталями обратиться к указываемой ниже литературе.

Каждый итерационный шаг начинается с двухдиагональной матрицы, которая преобразуется в другую двухдиагональную матрицу с меньшими внедиагональными элементами. Мы подробно рассмотрим один такой шаг в нашем примере. Поскольку две последние строки нулевые и остаются нулевыми в дальнейшем, то их можно опустить. Будем указывать для матричных элементов пять десятичных знаков после запятой вместо трех, как было до сих пор. Начинаем с матрицы

$$\begin{matrix} -7.41620 & 32.73169 & 0 \\ 0 & 10.60517 & 1.08016 \\ 0 & 0 & 0.00000. \end{matrix}$$

Исходя из трех элементов правого нижнего угла, вычисляется некоторое преобразование V; когда A замещается матрицей AV, изменяются два первых столбца:

$$\begin{matrix} 33.56105 & 0.13897 & 0 \\ 10.35262 & -2.30062 & 1.08016 \\ 0 & 0 & 0.00000. \end{matrix}$$

Заметим, что под диагональю появился ненулевой элемент 10.35262. Чтобы избавиться от этого нежелательного элемента, вычисляется преобразование U; при переходе от A к UA изменяются две первые строки:

$$\begin{matrix} 35.12152 & -0.54535 & 0.31839 \\ 0 & -2.23937 & 1.03216 \\ 0 & 0 & 0.00000. \end{matrix}$$

Требуемый нуль восстановлен, однако выше наддиагонали возник новый ненулевой элемент 0.31839. Следующее преобразование V комбинирует два последних столбца, чтобы исключить этот ненужный элемент:

$$\begin{matrix} 35.12152 & 0.63149 & 0 \\ 0 & 2.45431 & 0.23771 \\ 0 & 0.00000 & 0.00000. \end{matrix}$$

Если бы правый нижний элемент не имел значение 0.00000, то под диагональю появился бы еще один ненулевой элемент, там, где в нашей матрице стоит второй указатель 0.00000. Следующее

преобразование U, которое в данном примере мало отличается от единичной матрицы, оперирует с двумя последними строками [1]):

$$\begin{matrix} 35.12152 & 0.63149 & 0 \\ 0 & 2.45431 & 0.23771 \\ 0 & 0 & 0.00000. \end{matrix}$$

Мы вернулись к двухдиагональной матрице. Благодаря первоначальному V-преобразованию оба наддиагональных элемента уменьшились. (Для других матриц некоторые наддиагональные элементы могут на отдельных шагах даже возрастать, но в конечном счете они уменьшаются.)

Процесс повторяется. Следующей будет получена двухдиагональная матрица

$$\begin{matrix} 35.12722 & 0.00309 & 0 \\ 0 & 2.46540 & -0.00014 \\ 0 & 0 & 0.00000. \end{matrix}$$

Заметим, что диагональные элементы изменились мало и что оба наддиагональных элемента уменьшились.

После еще одного шага получим

$$\begin{matrix} 35.12722 & 0.00002 & 0 \\ 0 & 2.46540 & 0.00000 \\ 0 & 0 & 0.00000. \end{matrix}$$

Изменения диагональных элементов происходят лишь в шестом или седьмом десятичном знаке после запятой, которых мы уже не показываем. Последний наддиагональный элемент имеет теперь порядок ошибки округления и может рассматриваться как нуль.

Заключительный шаг уменьшает до пренебрежимо малого значения другой наддиагональный элемент:

$$\begin{matrix} 35.12722 & 0 & 0 \\ 0 & 2.46540 & 0 \\ 0 & 0 & 0.00000. \end{matrix}$$

Три диагональных элемента и есть сингулярные числа.

Мы не указывали явно использованные преобразования, а лишь результаты их воздействия на матрицу A; дело в том, что процесс выполняется посредством операций со строками и столбцами A, а не фактических матричных умножений. Если нужно находить только сингулярные числа, то A — единственный дву-

[1]) Смысл этого преобразования — исключение элемента (3,2).— *Прим. перев.*

мерный массив, участвующий в вычислениях. (В самом деле, в подпрограмме SVD двухдиагональная матрица представляется двумя одномерными массивами, один из которых на выходе содержит сингулярные числа.)

Если нужно вычислять не только сингулярные числа, но и матрицы U и V сингулярных векторов, то запасаются еще два массива, начальным состоянием которых служат единичные матрицы. В дальнейшем всякий раз, когда выполняется операция над строками или столбцами A, та же операция производится с одним из двух других массивов. Когда A в конечном счете приобретает диагональный вид, в остальных массивах содержатся матрицы U и V.

Этот алгоритм типичен для используемых в современных машинных подпрограммах решения различных матричных задач на собственные значения. Если задача не находится уже в специальной форме, то сначала матрица переводится в другую, содержащую много нулей. Затем к приведенной матрице применяется итерационный процесс, позволяющий получить еще больше нулей. Обычно повсюду используются ортогональные преобразования.

Первоначальное приведение не только обеспечивает эффективность итерационного процесса; оно также ускоряет его сходимость. Для алгоритма SVD, например, приведенная матрица двухдиагональная, итерационный процесс сконструирован так, чтобы аннулировать наддиагональные элементы, и теорема Уилкинсона утверждает, что итерации всегда сходятся и сходимость очень быстрая. Теорема Уилкинсона обсуждается в применении к SVD в книге Лоусон, Хэнсон (1974).

9.5. Подпрограмма SVD

АЛГОЛ-процедура для вычисления сингулярного разложения (авторы Голуб и Райнш) была опубликована в собрании матричных алгоритмов под редакцией Уилкинсона и Райнша. В рамках проекта NATS в Национальных лабораториях в Аргонне были разработаны фортран-подпрограммы для ряда матричных задач на собственные значения, в том числе трансляция процедуры SVD. NATS — это сокращение от National Activity to Test Software (Национальная программа тестирования математического обеспечения); соответствующая библиотека подпрограмм получила название EISPACK. Была опубликована инструкция для пользователя, относящаяся к части второй очереди EISPACK (Смит и др. (1976)). Хотя SVD включена во вторую очередь, она не описывается в инструкции издания 1976 г., а будет описана лишь в последующем издании.

Наш вариант SVD отличается от варианта из EISPACK тестом на малость элементов. Вариант из EISPACK требует включения оператора DATA, содержащего машинное эпсилон. Мы исключили машинное эпсилон и заменили тест на малость элементов операторами типа

```
IF (ABS (F) + ANORM .EQ. ANORM) GO TO 565.
```

В подпрограмме имеются три таких теста.

Мы решились на это изменение с некоторой неохотой, потому что подпрограммы, вошедшие в EISPACK, широко и тщательно тестировались, и распространяются и гарантируются проектом NATS. Однако внесенное изменение очень незначительно, и мы полагаем, что достигнутая этим машинная независимость стоила такой переделки.

Входом в SVD являются $m \times n$-матрица A, информация о размерностях и две логические переменные, указывающие, нужно ли вычислять матрицы U и V. Выходом будут сингулярные числа (обычно, хотя и не обязательно, в убывающем порядке), $m \times n$-матрица U, если она заказана, и $n \times n$-матрица V, если она заказана. В случае большой задачи любую из матриц U и V можно формировать на месте матрицы A. Статья Голуба и Райнша указывает, что процедуру легко можно модифицировать таким образом, чтобы получить полную $m \times m$-матрицу U.

Приводимая ниже иллюстрирующая программа основана на примере, которым мы пользовались всюду в этой главе. Сингулярные числа суть 35.127223, 2.465397 и 0.000000. Матрицы U и V приведены в § 9.2. Вычисляются лишь три первых столбца матрицы U; на некоторых машинах знаки столбцов обеих матриц U и V могут быть иными.

```
C  ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ ПОДПРОГРАММЫ SVD
C
      REAL A(5, 3), U(5, 3), V(5, 3), SIGMA(5), WORK(5)
      INTEGER I, IERR, J, M, N, NM
      NM=5
      M=5
      N=3
      DO 1 I=1, M
      DO 1 J=1, N
         A(I, J)=I+(J-1)*M
    1 CONTINUE
      CALL SVD(NM, M, N, A, SIGMA, .TPUE., U, .TRUE., V, IERR,
     1   WORK)
      IF(IERR.NE.0) WRITE (6, 2) IERR
    2 FORMAT (15H TROUBLE.IERR=,I4)
      DO 3 J=1,N
         WRITE (6, 6) SIGMA (J)
    3 CONTINUE
      WRITE (6, 7)
      DO 4 I=1, M
         WRITE (6, 6) (U(I, J), J=1, N)
```

```
      4  CONTINUE
         WRITE (6, 7)
         DO 5 I=1, N
            WRITE (6, 6) (V(I, J), J=1, N)
      5  CONTINUE
      6  FORMAT (3F10. 6)
      7  FORMAT (1H)
         STOP
         END

      SUBROUTINE SVD(NM, M, N, A, W, MATU, U, MATV, V, IERR, RV1)
C
      INTEGER I, J, K, L, M, N, II, II, KK, K1, LL, L1, MN, NM, ITS, IERR
      REAL A(NM, N), W(N), U(NM, N), V(NM, N), RV1(N)
      REAL C, F, G, H, S, X, Y, Z, SCALE, ANORM
      LOGICAL MATU, MATV
C
C     ЭТА ПОДПРОГРАММА ЕСТЬ ТРАНСЛЯЦИЯ АЛГОЛ-ПРОЦЕДУРЫ
C     SVD, ОПУБЛИКОВАННОЙ ГОЛУБОМ И РАЙНШЕМ В ЖУРНАЛЕ
C     NUMERISCHE MATHEMATIK, 14, 403—420 (1970), А ТАКЖЕ
C     В КНИГЕ HANDBOOK FOR AUTOMATIC COMPUTATION, VOL.
C     II — LINEAR ALGEBRA, 134—151 (1971).
C
C     ЭТА ПОДПРОГРАММА ВЫЧИСЛЯЕТ СИНГУЛЯРНОЕ РАЗЛОЖЕ-
C                                                  T
C     НИЕ A=USV ДЕЙСТВИТЕЛЬНОЙ ПРЯМОУГОЛЬНОЙ МАТРИ-
C     ЦЫ A С РАЗМЕРАМИ M И N. ПРИ ЭТОМ ИСПОЛЬЗУЮТСЯ
C     ДВУХДИАГОНАЛИЗАЦИЯ ПОСРЕДСТВОМ ХАУСХОЛДЕРОВЫХ
C     ОТРАЖЕНИЙ И ВАРИАНТ QR-АЛГОРИТМА.
C
C     НА ВХОДЕ.
C
C        NM СЛЕДУЕТ ПОЛОЖИТЬ РАВНЫМ СТРОЧНОЙ РАЗМЕР-
C           НОСТИ ДВУМЕРНЫХ МАССИВОВ, ЗАЯВЛЕННОЙ В ОПЕ-
C           РАТОРЕ РАЗМЕРНОСТИ ВЫЗЫВАЮЩЕЙ ПРОГРАММЫ.
C           ЗАМЕТИМ, ЧТО NM ДОЛЖНО БЫТЬ НЕ МЕНЬШЕ, ЧЕМ
C           МАКСИМУМ ИЗ M И N.
C
C        M — ЧИСЛО СТРОК A (И U).
C
C        N — ЧИСЛО СТОЛБЦОВ A (И U) И ПОРЯДОК V.
C
C        A СОДЕРЖИТ ПРЯМОУГОЛЬНУЮ ВХОДНУЮ МАТРИЦУ,
C           ДЛЯ КОТОРОЙ НАХОДИТСЯ РАЗЛОЖЕНИЕ.
C
C        MATU ДОЛЖЕН ИМЕТЬ ЗНАЧЕНИЕ .TRUE., ЕСЛИ НУЖНО
C           ВЫЧИСЛЯТЬ МАТРИЦУ U ИЗ РАЗЛОЖЕНИЯ, И ЗНАЧЕ-
C           НИЕ .FALSE. В ПРОТИВНОМ СЛУЧАЕ.
C
C        MATV ДОЛЖЕН ИМЕТЬ ЗНАЧЕНИЕ .TRUE., ЕСЛИ НУЖНО
C           ВЫЧИСЛЯТЬ МАТРИЦУ V ИЗ РАЗЛОЖЕНИЯ, И ЗНАЧЕ-
C           НИЕ .FALSE. В ПРОТИВНОМ СЛУЧАЕ.
C
C     НА ВЫХОДЕ.
C
C        A НЕ ИЗМЕНЯЕТСЯ (ЕСЛИ НА ЕЕ МЕСТЕ НЕ ЗАПИСЫ-
C        ВАЮТСЯ U ЛИБО V).
C
```

```
C     W СОДЕРЖИТ N (НЕОТРИЦАТЕЛЬНЫХ) СИНГУЛЯРНЫХ
C       ЧИСЕЛ A (ДИАГОНАЛЬНЫХ ЭЛЕМЕНТОВ S). ОНИ НЕ
C       УПОРЯДОЧЕНЫ. ЕСЛИ ПРОИСХОДИТ ВЫХОД ПО
C       ОШИБКЕ, ТО ДЛЯ ЗНАЧЕНИЙ ИНДЕКСА IERR+1,
C       IERR+2, ..., N СИНГУЛЯРНЫЕ ЧИСЛА ДОЛЖНЫ БЫТЬ
C       ВЕРНЫ.
C
C     U СОДЕРЖИТ МАТРИЦУ U (С ОРТОГОНАЛЬНЫМИ СТОЛБ-
C       ЦАМИ) ИЗ РАЗЛОЖЕНИЯ, ЕСЛИ ДЛЯ ПАРАМЕТРА MATU
C       БЫЛО ЗАДАНО ЗНАЧЕНИЕ .TRUE. В ПРОТИВНОМ СЛУ-
C       ЧАЕ U ИСПОЛЬЗУЕТСЯ КАК ВРЕМЕННЫЙ МАССИВ.
C       U МОЖЕТ СОВПАДАТЬ С A. ЕСЛИ ПРОИСХОДИТ ВЫХОД
C       ПО ОШИБКЕ, ТО СТОЛБЦЫ U, СООТВЕТСТВУЮЩИЕ
C       ИНДЕКСАМ ВЕРНЫХ СИНГУЛЯРНЫХ ЧИСЕЛ, ДОЛЖНЫ
C       ТАКЖЕ БЫТЬ ВЕРНЫ.
C
C     V СОДЕРЖИТ МАТРИЦУ V (ОРТОГОНАЛЬНУЮ) ИЗ РАЗЛО-
C       ЖЕНИЯ, ЕСЛИ ДЛЯ ПАРАМЕТРА MATV БЫЛО ЗАДАНО
C       ЗНАЧЕНИЕ .TRUE. В ПРОТИВНОМ СЛУЧАЕ НА V НЕ
C       ПРОИЗВОДИТСЯ ССЫЛОК. V ТАКЖЕ МОЖЕТ СОВПА-
C       ДАТЬ С A, ЕСЛИ U НЕ ВЫЧИСЛЯЕТСЯ. ЕСЛИ ПРОИС-
C       ХОДИТ ВЫХОД ПО ОШИБКЕ, ТО СТОЛБЦЫ V, СООТ-
C       ВЕТСТВУЮЩИЕ ИНДЕКСАМ ВЕРНЫХ СИНГУЛЯРНЫХ
C       ЧИСЕЛ, ДОЛЖНЫ БЫТЬ ТАКЖЕ ВЕРНЫ.
C
C     IERR РАВНО
C        НУЛЮ, ЕСЛИ ПРОИСХОДИТ НОРМАЛЬНЫЙ ВЫХОД ИЗ
C              ПОДПРОГРАММЫ,
C        K,      ЕСЛИ K-Е СИНГУЛЯРНОЕ ЧИСЛО НЕ БЫЛО
C              ОПРЕДЕЛЕНО ПОСЛЕ 30 ИТЕРАЦИЙ.
C
C     RV1—ЭТО МАССИВ ПРОМЕЖУТОЧНОГО ХРАНЕНИЯ.
C
C   ВОПРОСЫ И КОММЕНТАРИИ НУЖНО НАПРАВЛЯТЬ ПО АД-
C   РЕСУ B. S. GARBOW, APPLIED MATEMATICS DIVISION, ARGONNE
C   NATIONAL LABORATORY
C
C     ПОДПРОГРАММА МОДИФИЦИРОВАНА С ЦЕЛЬЮ ИСКЛЮ-
C     ЧИТЬ ПЕРЕМЕННУЮ МАСНЕР
C
      IERR=0
C
      DO 100 I=1, M
C
         DO 100 J=1, N
            U(I, J)=A(I, J)
  100 CONTINUE
C     ..... ХАУСХОЛДЕРОВО ПРИВЕДЕНИЕ К ДВУХДИАГОНАЛЬ-
C                                          НОЙ ФОРМЕ .....
      G=0. 0
      SCALE=0. 0
      ANORM=0. 0
C
      DO 300 I=1, N
         L=I+1
         RV1(I)=SCALE*G
         G=0. 0
```

```
      S=0. 0
      SCALE=0. 0
      IF (I .GT. M) GO TO 210
C
      DO 120 K=I, M
  120 SCALE=SCALE+ABS(U(K, I))
C
      IF(SCALE .EQ. 0. 0) GO TO 210
C
      DO 130 K=I, M
         U(K, I)=U(K, I)/SCALE
         S=S+U(K, I)**2
  130 CONTINUE
C
      F=U(I, I)
      G=-SIGN(SQRT(S), F)
      H=F*G-S
      U(I, I)=F-G
      IF (I .EQ. N) GO TO 190
C
      DO 150 J=L, N
         S=0. 0
C
         DO 140 K=I, M
  140    S=S+U(K, I)*U(K, J)
C
         F=S/H
C
         DO 150 K=I, M
            U(K, J)=U(K, J)+F*U(K, I)
  150 CONTINUE
C
  190 DO 200 K=I, M
  200 U(K, I)=SCALE*U(K, I)
C
  210 W(I)=SCALE*G
      G=0. 0
      S=0. 0
      SCALE=0. 0
      IF(I .GT. M .OR. I .EQ. N) GO TO 290
C
      DO 220 K=L, N
  220 SCALE=SCALE+ABS(U(I, K))
C
      IF(SCALE. EQ. 0. 0) GO TO 290
C
      DO 230 K=L, N
         U(I, K)=U(I, K)/SCALE
         S=S+U(I, K)**2
  230 CONTINUE
C
      F=U(I, L)
      G=-SIGN(SQRT(S), F)
      H=F*G-S
      U(I, L)=F-G
C
      DO 240 K=L, N
```

```
  240     RV1(K)=U(I, K)/H
C
        IF (I .EQ. M) GO TO 270
C
        DO 260 J=L, M
           S=0. 0
C          DO 250 K=L, N
  250      S=S+U(J, K)*U(I, K)
C
           DO 260 K=L, N
              U(J, K)=U(J, K)+S*RV1(K)
  260     CONTINUE
C
  270     DO 280 K=L, N
  280     U(I, K)=SCALE*U(I, K)
C
  290     ANORM=AMAX1(ANORM, ABS(W(I))+ABS(RV1(I)))
  300 CONTINUE
C     .....НАКОПЛЕНИЕ ПРАВОСТОРОННИХ ПРЕОБРАЗОВАНИЙ
      IF (.NOT. MATV) GO TO 410
C     ..... ДЛЯ I=N С ШАГОМ -1 ДО 1 ВЫПОЛНИТЬ-.....
      DO 400 II=1, N
         I=N+1-II
         IF (I.EQ.N) GO TO 390
         IF (G. EQ. 0. 0) GO TO 360
C
         DO 320 J=L, N
C     ..... ДВОЙНОЕ ДЕЛЕНИЕ ОБХОДИТ ВОЗМОЖНЫЙ
C     МАШИННЫЙ НУЛЬ.....
  320    V(J, I)=(U(I, J)/U(I, L))/G
C
         DO 350 J=L, N
            S=0. 0
C
            DO 340 K=L, N
  340       S=S+U(I, K)*V(K, J)
C
            DO 350 K=L, N
               V(K, J)=V(K, J)+S*V(K, I)
  350    CONTINUE
C
  360    DO 380 J=L, N
            V(I, J)=0. 0
            V(J, I)=0. 0
  380    CONTINUE
C
  390    V(I, I)=1. 0
         G=RV1(I)
         L=I
  400 CONTINUE
C     .....НАКОПЛЕНИЕ ЛЕВОСТОРОННИХ ПРЕОБРАЗОВА-
C     НИЙ.....
  410 IF (. NOT. MATU) GO TO 510
C     .....ДЛЯ I=MIN(N, M) С ШАГОМ -1 ДО 1 ВЫПОЛНИТЬ-.....
      MN=N
      IF (M .LT. N) MN=M
```

```
C
      DO 500 II=1, MN
         I=MN+1-II
         L=I+1
         G=W(I)
         IF (I .EQ. N) GO TO 430
C
         DO 420 J=L, N
  420    U(I, J)=0. 0
C
  430    IF(G. EQ. 0. 0) GO TO 475
         IF(I. EQ. MN) GO TO 460
C
         DO 450 J=L, N
            S=0. 0
C
            DO 440 K=L, M
  440       S=S+U(K, I)*U(K, J)
C     .....ДВОЙНОЕ ДЕЛЕНИЕ ОБХОДИТ ВОЗМОЖНЫЙ МАШИН-
C     НЫЙ НУЛЬ.....
            F=(S/U(I, I))/G
C
            DO 450 K=I, M
               U(K, J)=U(K, J)+F*U(K, I)
  450    CONTINUE
C
  460    DO 470 J=I, M
  470    U(J, I)=U(J, I)/G
C
         GO TO 490
C
  475    DO 480 J=I, M
  480    U(J, I)=0. 0
C
  490    U(I, I)=U(I, I)+1. 0
  500 CONTINUE
C     .....ДИАГОНАЛИЗАЦИЯ ДВУХДИАГОНАЛЬНОЙ ФОРМЫ.....
C     .....ДЛЯ K=N С ШАГОМ -1 ДО 1 ВЫПОЛНИТЬ-.....
  510 DO 700 KK=1, N
         K1=N-KK
         K=K1+1
         ITS=0
C     .....ПРОВЕРКА ВОЗМОЖНОСТИ РАСЩЕПЛЕНИЯ. ДЛЯ L=K
C     С ШАГОМ -1 ДО 1 ВЫПОЛНИТЬ-.....
  520    DO 530 LL=1, K
            L1=K-LL
            L=L1+1
            IF (ABS(RV1(L))+ANORM .EQ. ANORM) GO TO 565
C     .....RV1(1) ВСЕГДА РАВНО НУЛЮ. ПОЭТОМУ ВЫХОДА
C     ЧЕРЕЗ КОНЕЦ ЦИКЛА НЕ БУДЕТ.....
            IF(ABS(W(L1))+ANORM .EQ. ANORM) GO TO 540
  530    CONTINUE
C     .....ЕСЛИ L БОЛЬШЕ, ЧЕМ 1, ТО RV1(L) ПРИСВАИВАЕТСЯ
C     НУЛЕВОЕ ЗНАЧЕНИЕ.....
  540    C=0. 0
         S=1. 0
```

```
C
      DO 530 I=L, K
         F=S*RV1(I)
         RV1(I)=C*RV1(I)
         IF (ABS(F)+ANORM .EQ. ANORM) GO TO 565
         G=W(I)
         H=SQRT(F*F+G*G)
         W(I)=H
         C=G/H
         S=-F/H
         IF(.NOT. MATU) GO TO 560
C
         DO 550 J=1, M
            Y=U(J, L1)
            Z=U(J, I)
            U(J, L1)=Y*C+Z*S
            U(J, I)=-Y*S+Z*C
550   CONTINUE
C
560      CONTINUE
C     ......ПРОВЕРКА СХОДИМОСТИ.....
565      Z=W(K)
         IF (L .EQ. K) GO TO 650
C     .....СДВИГ ВЫБИРАЕТСЯ ИЗ НИЖНЕГО УГЛОВОГО
C     МИНОРА ПОРЯДКА 2.....
         IF(ITS .EQ. 30) GO TO 1000
         ITS=ITS+1
         X=W(L)
         Y=W(K1)
         G=RV1(K1)
         H=RV1(K)
         F=((Y-Z)*(Y+Z)+(G-H)*(G+H))/(2. 0*H*Y)
         G=SQRT(F*F+1. 0)
         F=((X-Z)*(X+Z)+H*(Y/(F+SIGN(G, F))-H))/X
C     .....СЛЕДУЮЩЕЕ QR-ПРЕОБРАЗОВАНИЕ.....
         C=1. 0
         S=1. 0
C
         DO 600 I1=L, K1
            I=I1+1
            G=RV1(I)
            Y=W(I)
            H=S*G
            G=C*G
            Z=SQRT(F*F+H*H)
            RV1(I1)=Z
            C=F/Z
            S=H/Z
            F=X*C+G*S
            G=-X*S+G*C
            H=Y*S
            Y=Y*C
            IF (.NOT. MATV) GO TO 575
C
            DO 570 J=1, N
               X=V(J, I1)
               Z=V(J, I)
```

```
                V(J, I1)=X*C+Z*S
                V(J, I)=-X*S+Z*C
  570       CONTINUE
C
  575       Z=SQRT(F*F+H*H)
            W(I1)=Z
C     ......ВРАЩЕНИЕ МОЖЕТ БЫТЬ ПРОИЗВОЛЬНЫМ, ЕСЛИ
C       Z РАВНО НУЛЮ.....
            IF (Z .EQ. 0. 0) GO TO 580
            C=F/Z
            S=H/Z
  580       F=C*G+S*Y
            X=-S*G+C*Y
            IF (.NOT. MATU) GO TO 600
C
            DO 590 J=1, M
              Y=U(J, I1)
              Z=U(J, I)
              U(J, I1)=Y*C+Z*S
              U(J, I)=-Y*S+Z*C
  590       CONTINUE
C
  600   CONTINUE
C
        RV1(L)=0.0
        RV1(K)=F
        W(K)=X
        GO TO 520
C     .....СХОДИМОСТЬ.....
  650   IF (Z. GE. 0. 0) GO TO 700
C       .....W(K) ДЕЛАЕТСЯ НЕОТРИЦАТЕЛЬНЫМ.....
        W(K)=-Z
        IF(.NOT. MATV) GO TO 700
C
        DO 690 J=1, N
  690   V(J, K)=-V(J, K)
C
  700 CONTINUE
C
      GO TO 1001
C     .....УСТАНОВИТЬ ЗНАЧЕНИЕ ПРИЗНАКА ОШИБКИ —
C     ПОСЛЕ 30 ИТЕРАЦИЙ НЕТ СХОДИМОСТИ К СИНГУЛЯРНОМУ
C     ЧИСЛУ.....
 1000 IERR=K
 1001 RETURN
      END
```

УПРАЖНЕНИЯ

9.1. Постройте 11 точек, беря $t_i=(i-1)/10$ и $y_i=\mathrm{erf}\,(t_i)$, $i=1, \ldots, 11$. Функцию $\mathrm{erf}\,(t)$ можно вычислять посредством любого из методов, описанных в упр. 1.1, 5.1 или 6.1.

(а) Выровняйте эти точки в смысле наименьших квадратов, беря полиномы со степенями от 1 до 10. Сравните построенные полиномы с $\mathrm{erf}\,(t)$ для ряда промежуточных (по отношению к исходным) значений t и исследуйте зависимость максимальной ошибки от числа n коэффициентов полинома.

(б) Поскольку erf (t) — нечетная функция t, т. е. erf (t)=—erf $(-t)$, то разумно для выравнивания тех же точек применить линейную комбинацию нечетных степеней t:

$$\text{erf}\,(t) \approx c_1 t + c_2 t^3 + \ldots + c_n t^{2n-1}.$$

Снова исследуйте, как ошибка в промежуточных точках зависит от n. Так как t в этой задаче ограничено интервалом [0, 1], то нет необходимости в использовании других базисных полиномов.

(в) Полиномы не слишком хороши в качестве приближающих функций для erf (t), потому что они не ограничены при растущем t, в то время как erf (t) асимптотически стремится к 1. Поэтому при тех же точках испробуйте модель вида

$$\text{erf}\,(t) \approx c_1 + e^{-t^2}(c_2 + c_3 z + c_4 z^2 + c_5 z^3),$$

где $z=1/(1+t)$. Как выглядит ошибка в промежуточных точках в сравнении с полиномиальными моделями?

(г) Возьмите ту же модель, что и в п. (в), но для

$$z = \frac{1}{1+\lambda t},$$

где λ — нелинейный параметр, который можно найти, используя подпрограмму FMIN, как это описывается в упр. 9.3.

9.2. В упр. 4.7, где задано $m=7$ точек, примените выравнивание посредством полиномов со степенями от 0 до 6. Начертите полученные полиномы, равно как и сплайн, интерполирующий эти точки.

9.3. *Разделимые наименьшие квадраты*

Предположим, что m точек (t_i, y_i), $i=1, \ldots, m$, нужно выровнять в смысле наименьших квадратов посредством следующей функции от t:

$$y(t) = c_1 + c_2 t + c_3 t^2 + c_4 e^{\lambda t}.$$

Эта функция содержит пять параметров: c_1, c_2, c_3, c_4 и λ. Четыре коэффициента c_i входят линейно, а коэффициент λ входит нелинейным образом.

Пусть $A(\lambda)$ — матрица размера $m\times 4$ с элементами

$$a_{ij} = t_i^{j-1}, \quad j = 1, 2, 3,$$
$$a_{i4} = e^{\lambda t_i}.$$

Пусть y — вектор порядка m, составленный из заданных значений y_i, а c — вектор из четырех неизвестных коэффициентов c_j. Мы получаем тогда такую оптимизационную задачу:

$$\min_{\lambda} \min_{c} \| A(\lambda) c - y \|^2.$$

Внутренний минимум, зависящий от четырех линейных параметров, можно найти для любого λ по подпрограмме SVD. *Внешний* минимум, зависящий от единственного нелинейного параметра λ, можно находить, используя подпрограмму FMIN. Заметим, что к SVD обращается подпрограмма-функция, вызываемая программой FMIN.

Вот два набора данных, к которым нужно применить этот метод. В первом случае не должно встретиться трудностей, но второй набор приводит к вырождению, которое можно выявить, контролируя величину внутреннего минимума и сингулярные числа матрицы $A(\lambda)$. Найдите значения c_j и λ, которые дают наилучшее приближение. Единственны ли коэффициенты c_j? Считайте, что данные верны лишь в двух десятичных знаках после запятой, так что y_i могут иметь ошибки, достигающие 0,005.

t	y, Первый набор	y, Второй набор
0.0	20.00	20.00
0.25	51.58	24.13
0.50	68.73	26.50
0.75	75.46	27.13
1.00	74.36	26.00
1.25	67.09	23.13
1.50	54.73	18.50
1.75	37.98	12.13
2.00	17.28	4.00

9.4. (а) Выровняйте данные переписей из упр. 4.5, пользуясь полиномами различных степеней. Используйте полученные приближения для прогноза численности населения в 1980 г. Как влияет на прогноз численности населения ваш выбор базисных полиномов? А выбор границы для пренебрежимо малых сингулярных чисел? А точность арифметики, если у вас есть выбор?

(б) Выровняйте те же данные переписей посредством модели

$$y(t) \approx c_1 + c_2(t - 1900) + c_3 e^{\lambda(t-1900)}$$

с переменным λ в соответствии с упр. 9.3. Предскажите численность населения в 1980 г.

(в) Попробуйте приблизить данные переписей квадратичным полиномом

$$y(t) \approx c_1 + c_2 t + c_3 t^2,$$

используя нормальные уравнения. Каково число обусловленности полученной матрицы? Каков прогноз численности населения на 1980 г.?

9.5. *Сингулярный анализ криптограмм*

Эта задача основана на статье Моулер, Моррисон (1977). Цель — отделить гласные от согласных в кодированном сообщении.

Пусть кто-нибудь приготовит для вас криптограмму, взяв текст из нескольких сотен знаков, опустив в нем пробелы и знаки препинания и выполнив простую взаимно-однозначную подстановку букв. Например, пусть исходный текст был такой:

NOW IS THE TIME FOR MEN TO AID THEIR COUNTRY.

Используя код «2001», в котором каждая буква заменяется своей предшественницей в алфавите (вспомните, что компьютер в фильме Стэнли Кубрика «2001» именовался HAL [1]), получим кодированное сообщение

MNVHRSGDSHLDENQLDMSNZHCSGDHQBNTMSQX.

Напишите программу, которая обрабатывает подобные тексты и формирует матрицу размера 26×26, элементы которой естественней индексировать не целыми числами, а буквами. Элемент (x, y) указывает, сколько раз в кодированном тексте встретилась пара «XY». В данном примере $a_{m,n} = 1$, $a_{n,v} = 1, \ldots, a_{s,g} = 2$ и т. д. При этом нужно считать, что последняя буква текста сопровождается первой, так что $a_{x,m} = 1$. Эта матрица называется *матрицей частот диграфа*.

Вычислите SVD для матрицы частот диграфа вашего текста.

Выведите компоненты столбцов U и V, соответствующих наибольшему сингулярному числу. Как правило, это компоненты первых столбцов $u_{x,1}$ и $v_{x,1}$, $x = a, b, \ldots, z$. Все компоненты этих векторов должны иметь одинаковый знак

[1]) Что декодируется как IBM.— *Прим. перев.*

и быть примерно пропорциональными частотам отдельных букв. Таким образом, в некодированном сообщении с типичным распределением букв компонента e будет наибольшей, компонента t — следующей по величине и т. д. Отсутствующим буквам будут соответствовать почти нулевые компоненты.

Рассмотрим теперь компоненты столбцов U и V, отвечающих *второму* сингулярному числу. Если сингулярные числа упорядочены, то это будут $u_{x,2}$ и $v_{x,2}$. Свяжем с каждой буквой x точку двумерной плоскости, декартовыми координатами которой являются компоненты x этих векторов, т. е. точку $(u_{x,2}, v_{x,2})$. Поскольку компоненты находятся в интервале $[-1, 1]$, мы получим 26 точек, расположенных в квадрате со стороной 2 и центром в начале координат.

Большинству букв будут соответствовать точки второго и четвертого квадрантов плоскости. Другими словами, будет лишь несколько букв, для которых соответствующие компоненты второго правого и второго левого сингулярных векторов имеют одинаковый знак. Кроме того, бóльшая часть согласных должна быть в одном квадранте, а гласных — в другом. Это связано с тем обстоятельством, что пары гласная-согласная или согласная-гласная встречаются более часто, чем пары гласная-гласная или согласная-согласная.

Будут и некоторые исключения. Например, букве H обычно предшествует согласная, а именно T, а сопровождает ее гласная. Буква N часто сопровождается другой согласной. В действительности подобные исключения могут помочь в распознавании таких букв в криптограмме.

Интересно было бы использовать тексты и не на английском языке. Например, для немецкого языка так хорошо не получилось бы, потому что он содержит сравнительно много пар гласная-гласная или согласная-согласная.

9.6. *Псевдообратная*

Напишите подпрограмму

SUBROUTINE PSEUDO (NM, M, N, A, TOL, APLUS, WORK),

которая бы, используя SVD, вычисляла эффективную псевдообратную для $m \times n$-матрицы A. Сингулярные числа, меньшие, чем TOL, должны рассматриваться как пренебрежимо малые. Проверьте подпрограмму на примере, указанном в этой главе, а также на других матрицах по своему выбору. Найдите пример, в котором изменение TOL приводило бы к нескольким различным результатам.

Та часть вашей подпрограммы, которая фактически формирует A^+, может выглядеть так:

```
      DO 20 I=1, N
      DO 20 J=1, M
         T=0. 0
         DO  10 K=1, N
            IF (SIGMA (K) . GT. TOL)
     *                T=T+V (I, K)*U (J, K)/SIGMA (K)
10       CONTINUE
         APLUS (I, J)=T
20    CONTINUE.
```

Попытайтесь организовать вашу программу так, чтобы, помимо A и APLUS требовалось как можно меньше дополнительной памяти.

9.7. *Псевдообратная.*

Докажите, что эффективная псевдообратная матрица удовлетворяет четырем условиям:

$$XAX = X;$$
$$AX \quad \text{симметрична;}$$
$$XA \quad \text{симметрична;}$$
$$\| AXA - A \| \leqslant \tau.$$

10. ВЫРАБОТКА СЛУЧАЙНЫХ ЧИСЕЛ И МЕТОДЫ МОНТЕ-КАРЛО

Существует много задач, в которых естественным образом присутствует случайный элемент. К примеру, если вы хотите исследовать влияние числа мясников на качество обслуживания в мясном магазине, то вам придется каким-то способом моделировать непредсказуемость моментов входа посетителей в магазин. Другим примером являются случайные помехи. Многие типы сбоев в коммуникационной системе моделируются лучше всего как случайные эффекты.

Есть другие задачи, которые сами по себе не содержат элемента случайности, но где по тем или иным причинам полезно брать случайные выборки. Например, в проводимой раз в десять лет переписи населения Соединенных Штатов было бы слишком дорого, да и просто излишне предоставлять каждому лицу наиболее полный набор вопросов. Обычно такой полный набор оставляют лишь для выборки 1 человек из 10 или 40, при этом выборка берется случайная, чтобы избежать какой-либо систематической ошибки. (Каждый десятый дом может оказаться на углу, и тем самым будут отобраны более зажиточные люди).

Еще одна задача, где случайный выбор оказывается удивительно полезным,— это грубая оценка объемов сложных областей в евклидовом пространстве более чем четырех или пяти измерений. Разумеется, сюда входит и приближенное вычисление интегралов. Обозначим область через R; обычно она определяется рядом неравенств. Предположим, что R — подмножество n-мерного единичного куба K. Идея *Монте-Карло* состоит в том, чтобы случайным образом выбрать в K большое число N точек, которые с одинаковой вероятностью могут оказаться в любой части K. Затем подсчитывают число M точек, попавших в R, т. е. удовлетворяющих неравенствам, определяющим R. Тогда M/N есть оценка объема R. Студенты, знакомые с биномиальным распределением, знают, что дисперсия оценки довольно большая, так что ее точность невелика. Все же выборка из 10 000 точек обеспечит точность около 1%, если только объем не

слишком близок к 0 или 1. Такой точности часто бывает достаточно, и добиться лучшего другими методами может оказаться очень трудно.

Для любого из подобных приложений, связанных с машиной, необходимо вырабатывать большие последовательности чисел, ведущих себя как случайные выборки из заданного распределения. Имеются три класса методов для выработки таких случайных чисел.

1. Можно использовать случайный физический процесс, например момент эмитирования электрона горячим катодом, интерпретируемый как фаза в некотором временном цикле. Такие эффекты легко получать, однако очень трудно добиться, чтобы связанные с ними результаты замеров времени не содержали систематической ошибки, чему приходится противодействовать другими методами. В любом случае физический процесс, будучи случайным, невоспроизводим, и потому машинную программу, основанную на нем, отлаживать нелегко.

2. Можно вычислять случайные числа автономно от их последующего потребления программой; хранить их можно в виде файла на диске или ленте. Подход, основанный на этой идее, был широко распространен в не знавшую ЭВМ эпоху, когда было опубликовано несколько книг с таблицами случайных чисел. Остается открытым вопрос о том, как выработать файл из случайных чисел. Если вы как программист не заготовите своего собственного файла, то вполне вероятно, что имеющийся файл будет слишком эксплуатироваться. Одно из возможных решений состоит в том, чтобы начать использовать файл случайным образом, но это возвращает нас к проблеме, как вырабатывать случайную позицию в файле. В прошлом статистики изобрели изощренные методы пользования таблицами случайных чисел.

3. Третий и наиболее распространенный метод заключается в применении алгоритма, вырабатывающего «случайные» числа, немедленно потребляемые машинной программой. Числа выдаются лишь в случае надобности в них, и они вполне воспроизводимы, что важно при проверке программы. Поскольку такие числа генерируются алгоритмом, они вовсе не случайные, и следовало бы называть их *псевдослучайными*. Тем не менее, данный подход, по-видимому, наиболее полезен, и в этих заметках будет обсуждаться только он.

10.1. Выработка равномерно распределенных чисел

Методы, вырабатывающие случайные числа с тем или иным распределением, обычно основаны на первоначальном получении случайного действительного числа, равномерно распределенно-

го на интервале [0,1]. Такое число позднее преобразуется к другому распределению. Поскольку целочисленная арифметика изучена лучше, то начинают с получения целого числа, равномерно распределенного в некоторой области неотрицательных целых чисел; например, для Системы 360 эта область простирается от 0 до $2^{31}-1$. Такое целое число можно преобразовать в число с плавающей точкой из [0,1]. (Однако отказ от обращения и умножения привел бы к значительной экономии времени, и для многих целей случайные целые числа столь же полезны, как и случайные действительные.)

Почти повсеместно используемый метод выработки псевдослучайных целых чисел состоит в выборе некоторой функции f, отображающей множество целых чисел в себя. Берем какое-нибудь x_0 и порождаем каждое следующее целое число посредством функции $x_{k+1}=f(x_k)$. Поначалу функции f выбирались как можно более сложные и трудно понимаемые: например, $f(x)$ определялась как целое число, двоичное представление которого составляет средний 31 разряд 62-разрядного квадрата числа x. Но отсутствие теории относительно f приводило к катастрофическим последствиям. В методе середины квадрата, например, $f(x)$ могла иногда, причем совершенно непредсказуемо, обратиться в нуль, и тогда все последующие x_k были равны 0. Поэтому уже довольно давно перешли к использованию функций, свойства которых известны вполне. Всякая последовательность целых чисел из интервала $(0,2^{31}-1)$ должна содержать повторения самое большое после $2^{31}\approx 10^9$ элементов. Используя теорию чисел, можно выбрать такую функцию f, для которой наперед будет известно, что ее период максимально возможный или близкий к максимальному. Этим избегается преждевременное окончание или зацикливание последовательности. Дальнейшее использование теории чисел может более или менее предсказать характер последовательности, давая пользователю некоторую степень уверенности в том, что она будет достаточно хорошо моделировать случайную последовательность чисел.

Чаще всего сейчас применяют функцию f вида

$$f(x)=ax+c \pmod m,$$

где для t-разрядных двоичных целых чисел m обычно равно 2^t. Здесь x_0, a и c — сами целые числа из той же области. Некоторые выборы для a и c хороши, другие — нет. (Пример: очевидно, что выбор $a=c=1$ ужасен.) В книге Кнут (1969), с. 101 и 193, следующим образом подытожены теоретико-числовые ограничения на выбор a и c_1:

1. x_0 может быть произвольно. Для проверки программы возможно $x_0=1$; в дальнейшем в качестве x_0 можно брать цифровую

версию времени прогона программы, чтобы обеспечить различные последовательности для различных прогонов.

2. Выбор a должен удовлетворять трем требованиям (для двоичных машин):

а) $a(\mathrm{mod}\ 8)=5$,

б) $m/100<a<m-\sqrt{m}$,

в) двоичные знаки a не должны иметь очевидного шаблона.

3. В качестве c следует выбирать нечетное число, такое, что

$$\frac{c}{m}\approx\frac{1}{2}-\frac{1}{6}\sqrt{3}\approx 0.21132.$$

Нужно помнить, что наименее значимые двоичные цифры x_k будут «не очень случайными». Поэтому, если, например, вы хотите использовать число x_k для случайного выбора одной из 16 возможных ветвей, берите наиболее значимые разряды x_k, а не наименее значимые. Наконец, для большей надежности полезно предварительно испытать случайные числа на какой-либо задаче с известным ответом, схожей с реальным приложением.

Мы запрограммировали простой датчик равномерно распределенных случайных чисел, следуя предложениям Кнута. Соответствующая подпрограмма, написанная на ФОРТРАНе, стандарт ANSI, приведена в следующем параграфе. Поскольку подпрограмма составлялась так, чтобы быть относительно независимой от конкретной машины, то мы назвали ее URAND, что означает как Universal RANDom number generator, так и Uniform RANDom number generator [1]).

URAND вырабатывает последовательность целых чисел, давая при n-м обращении к ней

$$Y_{n+1}=aY_n+c(\mathrm{mod}\ m),\quad n\geqslant 1.$$

Эти числа обращаются в числа с плавающей точкой из интервала [0,1), которые и будут значениями URAND. Полученное значение Y_{n+1} возвращается посредством параметра IY и при последующем вызове должно использоваться как фактический параметр. При первом обращении к URAND параметру IY нужно присвоить в качестве начального произвольное целочисленное значение.

Значения m, a и c вычисляются автоматически при первом входе в программу. Главное допущение, сделанное здесь, заключается в том, что машина использует двоичное представление целого числа, а умножение выполняется по модулю m, где m — степень двойки. Это предположение упрощает вычисления. URAND находит значение $m/2$, проверяя последовательные сте-

[1]) То есть «универсальный датчик случайных чисел» и «датчик равномерно распределенных случайных чисел».— *Прим. перев.*

пени двойки до тех пор, пока умножение на 2 уже больше не дает приращения величины. Предполагается также, что сложение целых чисел выполняется по модулю m либо сохраняются по крайней мере $\log_2(m)$ значимых разрядов. Значения a и c вычисляются в соответствии с указанными выше рекомендациями Кнута. В подпрограмме a соответствует параметр IA, а c — параметр IC. Случайный шаблон разрядов для a достигается обращением DATAN (1.DO), дающим значение с двойной точностью числа $\pi/4$, которое на двоичной машине совпадает со сдвинутым шаблоном для π. Деление на 8.DO и умножение на $m/2$ также, надо полагать, не изменят этого шаблона. Наконец, значение с двойной точностью преобразуется в целое число, умножается на 8 и увеличивается на 5, чтобы обеспечить условие $a(\mathrm{mod}\ 8)=5$. Полученное значение a равно примерно $(m/8)\pi \approx m/2$. Этим удовлетворяются ограничения в виде неравенств. Значение c вычисляется прямо из определения 3. Некоторые трансляторы с ФОРТРАНа не обращают константы типа 8.DO в точное представление с плавающей точкой, но это едва ли будет иметь серьезные последствия.

Как доказано в теореме А (Кнут (1969), стр. 29), последовательность $\{Y_n\}$ обязательно имеет период максимальной длины m. Однако наименее значимые двоичные разряды Y_n будут «не очень случайными». Когда Y_n преобразуются в числа с плавающей точкой, то эти наименее значимые разряды обычно не важны. Чтобы вычислять случайные целые числа между 0 и $K-1$, нужно использовать функцию IFIX(K*URAND(IY)).

Для машин серии 360 фирма IBM вместе со своей фортранной системой поставляет SSP[1]) — пакет подпрограмм для научных расчетов. Эти подпрограммы могут автоматически вызываться из трансляторов с ФОРТРАНа. В пакет входит и датчик случайных чисел под названием RANDU. *Мы настоятельно призываем вас использовать* URAND, *но не использовать* RANDU по причинам, которые сейчас будут объяснены [2]).

При генерировании Y_n RANDU полагает $a=65539$ и $c=0$. Период полученной последовательности вполне удовлетворителен — 2^{29}, т. е. четверть всей области целых чисел. (Период последовательности, вырабатываемой подпрограммой URAND на IBM 360, равен максимально возможному значению 2^{31}.) Трудности связаны с тем обстоятельством, что $65\,539=2^{16}+3$. Возможно, это число было выбрано для ускорения умножения, но оно приводит к катастрофическим свойствам последовательности.

---

[1]) То есть scientific subroutine package.— *Прим. перев.*

[2]) Этот призыв относится, разумеется, к американским читателям книги— *Прим. перев.*

Следующие выкладки проводятся по модулю 2^{31}:

$$\begin{aligned} x_{k+2} &= (2^{16}+3)\,x_{k+1} = (2^{16}+3)^2\,x_k \\ &= (2^{32}+6\cdot 2^{16}+9)\,x_k = [6\cdot(2^{16}+3)-9]\,x_k \\ &= 6x_{k+1} - 9x_k. \end{aligned}$$

Итак,

$$x_{k+2} = 6x_{k+1} - 9x_k \text{ для всех } k.$$

Как результат этого в последовательности, вырабатываемой подпрограммой RANDU, наблюдается крайне высокая корреляция между тремя подряд идущими случайными целыми числами. Например, каждый раз, когда x_k близко к нулю, x_{k+2} близко (по модулю 2^{31}) к $6x_{k+1}$.

Мы проверили от 10 000 до 100 000 троек чисел (x_k, x_{k+1}, x_{k+2}), генерируемых обеими подпрограммами: RANDU и URAND. Эти тройки можно раскладывать по 1000 ящикам, соответственно тому, в какую из десяти равных частей интервала $[0, 2^{31}-1]$ попадет каждая из переменных. (Это равносильно тому, что для проверки URAND интервал $[0,1)$ разделили на десять равных частей.) Если числа, вырабатываемые датчиком, действительно случайные, то каждая тройка должна иметь одинаковые шансы попасть в любой из 1000 ящиков. К распределению троек по ящикам можно применить критерий χ^2 (Кнут (1969)). Результаты показали, что подпрограмма URAND вполне удовлетворительна, в то время как RANDU совершенно дискредитировала себя в качестве датчика случайных чисел. Мы полагаем, что RANDU настолько плоха, что во многих случаях моделирование, основанное на ней, вероятно, имело определенный режим, особенно если использовались тройки чисел. Поскольку RANDU почти неизменно применяется пользователями Системы 360, то мы серьезно сомневаемся в результатах большого числа моделирований.

Марсалья (1968) доказал, что все датчики случайных чисел, использующие рекуррентные соотношения, в определенной степени страдают корреляцией между последовательными числами, однако для хорошо составленных датчиков, таких, как URAND, этот эффект гораздо меньше, чем для многих других.

Было изобретено много других тестов для датчиков случайных чисел, но обычно самая лучшая проверка датчика состоит в применении его к модельной задаче (из вашей области приложений), ответ которой вам известен.

В настоящее время растет интерес к методам построения последовательностей чисел x_k из $[0,1)$, которые не выглядят случайными, однако оказываются хорошо приспособленными для конкретного применения. Например, существуют последовательности x_k, которые равномерно распределены на интервале $[0,1)$ с гораздо большей регулярностью, чем подлинно случайные чис-

ла. Они могут быть полезны для интегрирования по методу Монте-Карло вследствие меньшей дисперсии оценок.

Если скорость выработки равномерно распределенных случайных чисел жизненно важна для вашей программы, вам нужно изучить специализированные программы для вашего класса машин, публикуемые в текущей литературе. Например, имеется программа на ФОРТРАНе (Марсалья, Брэй (1968)) и программа для машин серии 360, написанная в машинном коде (Серафин (1969)). Время выполнения последней на машине 360/67 составляет 11.9 микросекунд плюс 4.2 микросекунды на вызов подпрограммы. Для подпрограммы URAND на IBM 360/67 требуется около 90 микросекунд (FORTPAN (H), (OPT=2)).

10.2. Подпрограмма URAND

Следующая программа иллюстрирует тривиальное применение URAND. Печатается десять строк; каждая из них содержит в случайном порядке слова heads и tails.

```
C   Иллюстрирующая программа для URAND
C
      IY=0
      DO 10 I=1, 10
         IF(URAND(IY) .LT. 0. 5) GO TO 5
         WRITE(6, 1)
         GO TO 10
    5    WRITE(6, 2)
   10 CONTINUE
    1 FORMAT(6H HEADS)
    2 FORMAT(6H TAILS)
      STOP
      END
```

Другое начальное значение для переменной IY приведет к другой последовательности случайных чисел. Более того, даже при одном и том же начальном значении для IY на машинах с различным размером слова будут получены разные последовательности.

```
      REAL FUNCTION URAND(IY)
      INTEGER IY
C
C     URAND—ЭТО ДАТЧИК РАВНОМЕРНО РАСПРЕДЕЛЕННЫХ
C  СЛУЧАЙНЫХ ЧИСЕЛ, ОСНОВАННЫЙ НА ТЕОРИИ И ПРЕДЛО-
C  ЖЕНИЯХ, СОДЕРЖАЩИХСЯ В КНИГЕ КНУТ (1969), ТОМ 2.
C  ПЕРЕД ПЕРВЫМ ОБРАЩЕНИЕМ К URAND ЦЕЛОЙ ПЕРЕМЕН-
C  НОЙ IY СЛЕДУЕТ ПРИСВОИТЬ ПРОИЗВОЛЬНОЕ ЦЕЛОЧИСЛЕН-
C  НОЕ НАЧАЛЬНОЕ ЗНАЧЕНИЕ. ВЫЗЫВАЮЩАЯ ПРОГРАММА
C  НЕ ДОЛЖНА ИЗМЕНЯТЬ ЗНАЧЕНИЕ IY МЕЖДУ ПОСЛЕДОВА-
C  ТЕЛЬНЫМИ ВЫЗОВАМИ. ЗНАЧЕНИЯ ФУНКЦИИ URAND ЯВЛЯ-
C  ЮТСЯ ЧИСЛАМИ ИЗ ИНТЕРВАЛА (0, 1).
C
```

```
      INTEGER IA, IC, ITWO, M2, M, MIC
      DOUBLE PRECISION HALFM
      REAL S
      DOUBLE PRECISION DATAN, DSQRT
      DATA M2/0/, ITWO/2/
      IF(M2. NE. 0) GO TO 20
C
C     ЕСЛИ ЭТО ПЕРВЫЙ ВХОД, ТО ВЫЧИСЛИТЬ ДЛИНУ ЦЕЛО-
C     ЧИСЛЕННОГО МАШИННОГО СЛОВА
C
      M=1
   10 M2=M
      M=ITWO*M2
      IF(M .GT. M2) GO TO 10
      HALFM=M2
C
C     ВЫЧИСЛИТЬ МНОЖИТЕЛЬ И ПРИРАЩЕНИЕ ЛИНЕЙНОГО КОН-
C     ГРУЭНТНОГО МЕТОДА
C
      IA=8*IDINT(HALFM*DATAN(1. D0)/8. D0)+5
      IC=2*IDINT(HALFM*(0. 5D0—DSQRT(3. D0)/6. D0))+1
      MIC=(M2—IC)+M2
C
C     S—МАСШТАБИРУЮЩИЙ МНОЖИТЕЛЬ ДЛЯ ПРЕОБРАЗОВАНИЯ
C     В ЧИСЛО С ПЛАВАЮЩЕЙ ТОЧКОЙ
C
      S=0. 5/HALFM
C
C     ВЫЧИСЛИТЬ СЛЕДУЮЩЕЕ СЛУЧАЙНОЕ ЧИСЛО
C
   20 IY=IY*IA
C
C     СЛЕДУЮЩИЙ ОПЕРАТОР—ДЛЯ МАШИН, КОТОРЫЕ НЕ ДОПУ-
C     СКАЮТ ПЕРЕПОЛНЕНИЯ ЦЕЛЫХ ЧИСЕЛ ПРИ СЛОЖЕНИИ
C
      IF (IY .GT. MIC)IY=(IY—M2)—M2
C
      IY=IY+IC
C
C     СЛЕДУЮЩИЙ ОПЕРАТОР—ДЛЯ МАШИН, У КОТОРЫХ ДЛИНА
C     СЛОВА ДЛЯ СЛОЖЕНИЯ БОЛЬШЕ, ЧЕМ ДЛЯ УМНОЖЕНИЯ
C
      IF (IY/2 .GT. M2) IY=(IY—M2)—M2
C
C     СЛЕДУЮЩИЙ ОПЕРАТОР—ДЛЯ МАШИН, У КОТОРЫХ ПЕРЕ-
C     ПОЛНЕНИЕ ЦЕЛОГО ЧИСЛА ВЛИЯЕТ НА ЗНАКОВЫЙ РАЗРЯД
C
      IF (IY .LT. 0) IY=(IY+M2)+M2
      URAND=FLOAT(IY)*S
      RETURN
      END
```

10.3. Выбор из других распределений

Часто выработка случайного числа из [0,1) является просто средством принятия некоторого случайного решения. В качестве очень простого примера предположим, что в некой программе моделирования желательно при случайном выборе попадать на какую-то ветвь одну десятую часть всего времени. Этого легко добиться, выбирая случайное число y на [0,1) и беря данную ветвь, если $y<0.1$. Более быстрый способ — выбирать случайное целое число в $[0, 2^{31}-1]$ и брать эту ветвь при $x<2^{31}/10$. Таким же образом из равномерно распределенных случайных целых чисел можно получить любое конечное распределение с различными весами.

Гораздо более сложно вычисление случайного числа из неравномерного непрерывного распределения. Если для функции распределения $F(x)$ не имеется каких-либо специальных приемов, то общий метод заключается в следующем: берут случайное число, равномерно распределенное на [0,1), а затем, пользуясь обратной функцией, образуют $y=F^{-1}(x)$. Трудной проблемой при этом является достаточно быстрое и достаточно точное формирование F^{-1}.

Некоторые часто используемые и важные распределения уже довольно полно исследованы. Вероятно, наиболее известным является *нормальное распределение* N (0,1) с нулевым средним значением и дисперсией 1. Оперировать с обратным отображением F^{-1} сравнительно трудно и медленно. Распространенный подход таков: берут 12 равномерно распределенных на [0,1) чисел y_i, затем по формуле $x_i=2y_i-1$ получают 12 случайных чисел, равномерно распределенных на [—1,1), и складывают их. Результат будет довольно хорошим приближением к выбору из распределения N (0,1). Метод является относительно медленным.

Намного более изобретательный метод принадлежит Боксу и Мюллеру. Приводим его в более поздней модификации Джеймса Белла из Стэнфорда.

1. Образовать два случайных числа U_1 и U_2, равномерно распределенных на [0,1) (например, пользуясь подпрограммой URAND).

2. Образовать $V_1=2U_1-1$ и $V_2=2U_2-1$; эти числа равномерно распределены на [—1,1).

3. Образовать $S=V_1^2+V_2^2$. Если $S>1$, отбросить V_1 и V_2 и вернуться к шагу 1. (Здесь мы теряем 22% эффективности.) Если $S\leqslant 1$, то (V_1, V_2) — случайная точка, равномерно распределенная в единичном круге.

4. Образовать

$$X_1=V_1\sqrt{\frac{-2\ln S}{S}},\quad X_2=V_2\sqrt{\frac{-2\ln S}{S}},$$

где ln — натуральный логарифм (на этот шаг затрачивается бо́льшая часть времени). В книге Кнут (1969), с. 130, доказано, что X_1 и X_2 — независимые случайные величины, имеющие нормальное распределение $N(0,1)$. Незначительная модификация этого метода была опубликована в виде программы языка АЛГОЛ 60 (Пайк (1965)).

Значительно более быстрые методы можно найти в статье Аренс, Дитер (1972). Родственный, но иной метод описан в отчете Форсайт (1972); случайная нормально распределенная величина получается здесь в среднем за 4.036 обращения к подпрограмме URAND плюс несколько сравнений.

УПРАЖНЕНИЯ

10.1 Если x — случайная величина с нормальным распределением $N(0,1)$, а z — некоторая константа, то можно показать, что вероятность того, что x меньше z (обозначаемая через $P[x<z]$), равна

$$P[x<z]=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z}e^{-t^2/2}\,dt$$
$$=\frac{1}{2}\left(1+\operatorname{erf}\frac{z}{\sqrt{2}}\right).$$

Пусть M — большое целое число, например 10 000. Выработайте M случайных чисел x_i из распределения $N(0,1)$, пользуясь подпрограммой URAND и схемой Бокса — Мюллера, описанной в § 10.3. Храните их в памяти машины в виде массива. Пусть z принимает значения от —3.0 до 3.0 с шагом 0.2. Для каждого z выведите на печать и сравните две величины: (число тех x_i, которые $<z)/M$ и $\frac{1}{2}(1+\operatorname{erf}(z/\sqrt{2}))$. Значения функции erf можно вычислять любым из методов, изложенных в упр. 1.1, 5.1 и 6.1. Попробуйте поэкспериментировать с различными начальными значениями для URAND. Если у вас достаточно машинного времени, испытайте бо́льшие значения M.

10.2. Возьмите два датчика случайных чисел — URAND и RANDU; примените критерий χ^2 к распределению троек по 1000 ящиков, упомянутому в § 10.1. Чтобы применение критерия χ^2 имело смысл, необходимо, чтобы каждая тройка использовала три «свежих» случайных числа; не пытайтесь включать одно и то же число в различные тройки.

10.3. С помощью метода Монте-Карло найдите площадь круга радиуса 1.

10.4. Используя метод Монте-Карло, найдите объем области четырехмерного евклидова пространства, описываемой следующими неравенствами:

$$0\le x_1\le 1,$$
$$0\le x_2\le 1,$$
$$0\le x_3\le 1,$$
$$0\le x_4\le 1,$$
$$x_1+x_2^2+x_3^3+x_4^4\le 0.7,$$
$$x_1+\sin x_4\le 0.8,$$
$$x_1x_3\le 0.5.$$

10.5. В книге Кнут (1969) обсуждается ряд тестов для проверки эффективности датчика случайных чисел. Используя по возможности больше из этих тестов, попытайтесь определить качество случайных чисел, вырабатываемых подпрограммой URAND на вашей машине.

10.6. Эта задача связана со случайными блужданиями по решеткам одного, двух и трех измерений. Случайное блуждание есть последовательность единичных шагов, где каждый шаг производится в направлении одной из координатных осей и все возможные направления равновероятны. Например, в двумерном случае при старте из точки с целочисленными координатами (x, y) единичный шаг с одинаковой вероятностью приведет в одну из четырех соседних — $(x+1, y)$, $(x-1, y)$, $(x, y+1)$, $(x, y-1)$. В одномерном случае возможных соседей два, а в трехмерном — шесть.

Подпрограмму URAND можно использовать, чтобы экспериментально определить приближенные ответы на следующие вопросы относительно случайных блужданий, исходящих из начала координат.

(а) В случае одномерного блуждания: Каково ожидаемое число попаданий в некоторую фиксированную ненулевую точку до первого возвращения в начало координат? Зависит ли это число от расположения точки?

(б) В случае двумерного блуждания: Существует ли конечная граница для ожидаемого числа шагов до первого возвращения в начало координат?

(в) В случае трехмерного блуждания: Какова вероятность возвращения в начало координат?

Для некоторых из этих вопросов получение даже двух верных цифр в ответе стоит очень большого машинного времени. Для всех известны аналитические решения. См. книги Феллер (1950) или Спицер (1964).

Ваша программа будет содержать два больших целых числа: M — число выполняемых случайных блужданий и N — максимальное число шагов в одном блуждании. Если в данном блуждании произведено N шагов, то считается, что оно не вернется в начало. Проверьте вашу программу при относительно малых значениях M и N; затем проведите счет для бо́льших значений. Было бы желательно исследовать влияние M и N на результаты, но это потребует очень много машинного времени.

СПИСОК ЛИТЕРАТУРЫ

Абрамовиц, Стиган (ред.) (Abramowitz M., Stegun I.)
(1964) Handbook of mathematical functions, graphs, and mathematical tables. Washington, D. C. (Русский перевод: Абрамовиц М., Стиган И. Справочник по специальным функциям.— М.: Наука, 1979.)

Алберг, Нильсон, Уолш (Ahlberg H. J., Nilson E. N., Walsh J. L.)
(1967) The theory of splines and their application, New York: Academic Press. (Русский перевод: Алберг Дж., Нильсон Э., Уолш Дж. Теория сплайнов и ее приложения. Пер. с англ.— М.: Мир, 1972.)

Аноним
(1966) «U. S. A. standard Fortran», USASX3.9—1966. United States of America Standards Institute.

Аренс, Дитер (Ahrens J. H., Dieter U.)
(1972) Computer methods for sampling from the exponential and normal distributions, *Comm. ACM*, **15**, 873—882.

Браун, Контэ (Brown K. M., Conte S. D.)
(1967) The solution of simultaneous nonlinear equations, Proc. 22nd National Conference of ACM, 111—114.

Брент (Brent R. P.)
(1973) Algorithms for minimization without derivatives. Englewood Cliffs, N. J.: Prentice-Hall.

Булирш, Стоэр (Bulirsch R., Stoer J.)
(1966) Numerical treatment of ordinary differential equations by extrapolation methods, *Numer. Math.*, **8**, 1—13.

Вендрофф (Wendroff B.)
(1966) Theoretical numerical analysis. New York: Academic Press.
(1969) First principles of numerical analysis: An undergraduate text. Reading, Mass:. Addison-Wesley.

Вильде (Wilde D. J.)
(1964) Optimum seeking methods. Englewood Cliffs, N. J.: Prentice-Hall.

Гарднер (Gardner M.)
(1961) Mathematical puzzles and diversions. New York: Simon and Schuster. (Русский перевод: Гарднер М. Математические головоломки и развлечения. Пер. с англ.— М.: Мир, 1971.)

Гилл, Мюррэй (Gill P. E., Murray W.)
(1974) Numerical methods for constrained optimization. New York: Academic Press. (Русский перевод: Численные методы условной оптимизации. Редакторы Ф. Гилл и У. Мюррэй. Пер. с англ.— М.: Мир, 1977.)

Гилл, Мюррэй, Питфилд (Gill P. E., Murray W., Pitfield R. A.)
(1972) The implementation of two quasi-Newton algorithms for unconstrained optimization, DNAC 11. National Physical Laboratory.

Гилпин (Gilpin M. E.)
(1972) Enriched predictor-prey systems: theoretical stability, *Science*, **177**, 902—904.

Гильберт (Hilbert D.)
(1894) Ein beitrag zur theorie des Legendre'schen polynoms, *Acta Math.*, **18**, 155—160.

Гир (Gear C. W.)
(1971) Numerical initial value problems in ordinary differential equations. Englewood Cliffs, N. J.: Prentice-Hall.

Голуб, Райнш (Golub G. H., Reinsch C.)
(1971) Singular value decomposition and least squares solution, in J. H. Wilkinson and C. Reinsch (eds.), Handbook for automatic computation, vol. II: Linear algebra. Heidelberg: Springer.

Грэг, Стьюарт (Gragg W. B., Stewart G. W.)
(1976) A stable variant of the secant method for solving nonlinear equations, *SIAM J. Numer. Anal.*, **13**, 889—903.

Данциг (Dantzig G. B.)
(1963) Linear programming and extensions. Princeton, N. J.: Princeton University Press. (Русский перевод: Данциг Дж. Б. Линейное программирование, его применения и обобщения. Пер. с англ.— М.: Прогресс, 1966.)

Деккер (Dekker T. J.)
(1969) Finding a zero by means of successive linear interpolation, in B. Dejon and P. Henrici (eds.), Constructive aspects of the fundamental theorem of algebra. New York: Wiley-Interscience.
(1971) A floating-point technique for extending the available precision, *Numer. Math.*, **18**, 224—242.

Дженкинс, Трауб (Jenkins M. A., Traub J. F.)
(1972) Zeros of a complex polynomial, Algorithm 419, *Comm. ACM*, **15**, 97—99.

Джойс (Joyce D. C.)
(1971) Survey of extrapolation processes in numerical analysis, *SIAM Review*, **13**, 435—490.

Дэвис (Davis H. T.)
(1962) Introduction to nonlinear differential and integral equations. New York: Dover.

Келлер (Keller H. B.)
(1968) Numerical methods for two-point boundary value problems. Waltham, Mass.: Ginn/Blaisdell.

Кнут (Knuth D. E.)
(1969) Seminumerical algorithms. The art of computer programming, vol. 2. Reading Mass.: Addison-Wesley. (Русский перевод: Кнут Д. Е. Искусство программирования. Том 2. Получисленные алгоритмы. Пер. с англ.— М.: Мир, 1977.)

Крог (Krogh F. T.)
(1973) On testing a subroutine for the numerical integration of ordinary differential equations, *J. Assoc. Comput. Mach.*, **20**, 545—562.

Ланцош (Lanczos C.)
(1957) Applied analysis. Englewood Cliffs, N. J.: Prentice-Hall. (Русский перевод: Ланцош К. Практические методы прикладного анализа. Пер. с англ.—М.: Физматгиз, 1961.)

Линес (Lyness J. N.)
(1969a) Notes on the adaptive Simpson quadrature routine, *J. ACM*, **16**, 483—495.
(19696) The effect of inadequate convergence criteria in automatic routines, *Computer J*, **12**, 279—281.
(1970) SQUANK (Simpson quadrature used adaptively noise killed) Algorithm 379, *Comm. ACM*, **13**, No. 1, April, 260—262.

Линес, Моулер (Lyness J. N., Moler C. B.)
(1967) Numerical differentiation of analytic functions, *SIAM J. Num. Anal.*, 4, 202—210.

Линес, Санде (Lyness J. N., Sande G.)

(1971) Evaluation of normalized Taylor coefficients: Algorithm 413, ENTCAF and ENTCRE, *Comm. ACM*, **14**, 669—675.
Лотка (Lotka A. J.)
(1956) Elements of mathematical biology. New York: Dover.
Лоусон, Хэнсон (Lawson C. L., Hanson R. J.)
(1974) Solving least squares problems. Englewood Cliffs, N. J. Prentice-Hall.
Маккиман (McKeeman W. M.)
(1962) Adaptive numerical integration by Simpson's rule, algorithm 145, *Comm. ACM*, 5, 604.
Малкольм (Malcolm M. A.)
(1972) Algorithms to reveal properties of floating-point arithmetic, *Comm. ACM*, **15**, 949—951.
Малкольм, Симпсон (Malcolm M. A., Simpson R. B.)
(1975) Local vs. global strategies in adaptive quadrature, *Trans. on Math. Software*, **1**, 129—146.
Марсалья (Marsaglia G.)
(1968) Random numbers fall mainly in the planes, *Proc. Nat. Acad. Sci.*, **61**, 25—28.
Марсалья, Брэй (Marsaglia G., Bray T. A.)
(1968) One-line random number generators and their use in combinations, *Comm. ACM*, **11**, 757—759.
Милн (Milne W. E.)
(1953) Numerical solution of differential equations. New York: Wiley. (Книгу можно заказать в издательстве Dover, Нью-Йорк.)
Мозес (Moses J.)
(1972) Toward a general theory of special functions, *Comm. ACM*, 15, No. 7, 550—554.
Моулер (Moler C. B.)
(1972) Matrix computations with Fortran and paging, *Comm. ACM*, **15**, No. 4, April, 268—270.
Моулер, Моррисон (Moler C. B., Morrison D. R.)
(1977) Singular value decomposition and cryptography. (Рукопись, подготовленная для публикация в Университете Нью-Мексико.)
Моулер, Соломон (Moler C. B., Solomon L. P.)
(1970) Use of splines and numerical integration in geometrical acoustics, *J. Acoustical Soc. Amer.*, 48, No. 3, May, 739—744.
Мэй (May R. M.)
(1972) Limit cycles in preditor-prey communities, *Science*, **177**, 900—902.
Мюллер (Muller D. E.)
(1956) A method for solving algebraic equations using an automatic computer, *Math. Tables and Other Aids to Computation*, 10, 208—215.
Мюррэй (Murray W.)
(1972) Numerical methods for unconstrained optimization. New York: Academic Press.
фон Нейман, Голдстайн (von Neumann J., Goldstine H. H.)
(1947) Numerical inverting of matrices of high order, *Bull. Amer. Math. Soc.*, **53**, 1021—1099, and *Proc. Amer. Math. Soc.*, **2**(1951), 188—202.
Ортега, Рейнболдт (Ortega J. M., Rheinboldt W. C.)
(1970) Iterative solution of nonlinear equations in several variables. New York: Academic Press. (Русский перевод: Ортега Дж., Рейнболдт В. Итерационные методы решения нелинейных систем уравнений со многими неизвестными. Пер. с англ.— М.: Мир, 1975.)
Орчард-Хейс (Orchard-Hays W.)
(1968) Advanced linear-programming computing techniques. New York: McGraw-Hill.
Осборн (Osborne M. R.)

(1968) A new method for the integration of stiff systems of ordinary differential equations, in Proceedings IFIP congress 68. Amsterdam: North-Holland.

Островский (Ostrowski A.)
(1966) Solution of equations and systems of equations, 2nd ed. New York: Academic Press. (Русский перевод: Островский А. М. Решение уравнений и систем уравнений. Пер. с англ.— М.: ИЛ, 1963.)

Пайк (Pike M. C.)
(1965) Random normal deviate, Algorithm 267, *Comm. ACM*, **8**, 606.

Райдер (Ryder B. G.)
(1974) The PFORT verifier, Software — Practice and Experience, **4**, 359—377.

Райнш (Reinsch C.)
(1967) Smoothing by spline functions, *Numer. Math.*, **10**, 177—183.
(1971) Smoothing by spline functions II, *Numer. Math.*, **16**, 451—454.

Райс (Rice J. R.)
(1975) A metaalgorithm for adaptive quadrature. *J. ACM*, **22**, 61—82.

Ричардсон, Гонт (Richardson L. F., Gaunt J. A.)
(1927) The deferred approach to the limit. *Trans. Roy. Soc. London*, 226A, 300.

Рэлстон (Ralston A.)
(1965) A first course in numerical analysis. New York: McGraw-Hill.

Рэлстон, Уилф (ред.) (Ralston A., Wilf H.)
(1960) Mathematical methods for digital computers, vol. 1. New York: Wiley.
(1967) Mathematical methods for digital computers, vol. 2. New York: Wiley.

Седгвик (Sedgwick A. E.)
(1973) An effective variable order, variable step Adams method, Ph. D. thesis, Department of Computer Science, Tech. Report No. 53, University of Toronto.

Серафин (Seraphin D. S.)
(1969) A fast random number generator for IBM 360, *Comm. ACM*, **12**, 695.

Смит, Бойл, Донгарра Гарбоу, Икебе, Клема, Моулер (Smith B. T., Boyle J. M., Dongarra J., Garbow B. S., Ikebe X., Klema V. C., Moler C. B.)
(1976) Matrix eigensystem routines — EISPACK guide. Heidelberg: Springer.

Спицер (Spitzer F. L.)
(1964) Principles of random walk, Princeton: Van Nostrand. (Русский перевод: Спицер Ф. Принципы случайного блуждания. Пер. с англ.— М.: Мир, 1969.)

Стиган, Абрамовиц (Stegun I. A., Abramowitz M.)
(1956) Pitfalls in computation, *J. Soc. Indust. Appl. Math.*, **4**, 207—219.

Стьюарт (Stewart G. W.)
(1973) Introduction to matrix computation. New York: Academic Press.

Уилкинсон (Wilkinson J. H.)
(1963) Rounding errors in algebraic processes. Englewood Cliffs, N. J.: Prentice-Hall.
(1965) The algebraic eigenvalue problem. Oxford: Clarendon Press. (Русский перевод: Уилкинсон Дж. Алгебраическая проблема собственных значений. Пер. с англ.— М.: Наука, 1970.)
(1967) Two algorithms based on successive linear interpolation, Tech. Report STAN-CS-67-60. Stanford, Calif.: Computer Science Department, Stanford University.

Уилкинсон, Райнш (Wilkinson J. H., Reinsch C.)
(1971) Handbook for automatic computation. Heidelberg: Springer. (Русский перевод: Уилкинсон Дж., Райнш. Справочник алгоритмов

на языке АЛГОЛ. Линейная алгебра. Пер. с англ.— М.: Машиностроение, 1976.)

Фаддеев Д. К., Фаддеева В. Н.
(1963) Вычислительные методы линейной алгебры. Изд. 2-е.— М.-Л.: Физматгиз.

Феллер (Feller W.)
(1950) An introduction to probability theory and its applications, New York: Wiley. (Русский перевод: Феллер В. Введение в теорию вероятностей и ее приложения. Пер. с англ.— М.: Мир, 1967.)

Фельберг (Fehlberg E.)
(1970) Klassische Runge-Kutta-formeln vierter und niedregerer ordnung mit schrittweitenkontrolle und ihre anwendung auf warmeleitungsprobleme, *Computing*, **6**, 61—71.

Флетчер, Пауэлл (Fletcher R., Powell M. J. D.)
(1963) A rapidly convergent descent method for minimization, *Computer J.*, **6**, 163—168.

Форсайт (Forsythe G. E.)
(1969) What is a satisfactory quadratic equation solver, in B. Dejon and P. Henrici (eds.), Constructive aspects of the fundamental theorem of algebra. New York: Wiley-Interscience, pp. 53—61.
(1972) Von Neumann's comparison method for random sampling from the normal and other distributions, Tech. Report STAN-CS-254-72. Stanford, Calif.: Computer Science Department, Stanford University.

Форсайт, Вазов (Forsythe G. E., Wasow W. R.)
(1960) Finite difference methods for partial differential equations. New York: Wiley. (Русский перевод: Вазов В. Р., Форсайт Дж. Разностные методы решения дифференциальных уравнений в частных производных. Пер. с англ.— М.: ИЛ, 1963.)

Форсайт, Моулер (Forsythe G. E., Moler C. B.)
(1967) Computer solution of linear algebraic systems. Englewood Cliffs, N. J.: Prentice-Hall. (Русский перевод: Форсайт Дж. Е., Молер К. Численное решение систем линейных алгебраических уравнений. Пер. с англ.— М.: Мир, 1969.)

Хал, Энрайт, Феллен, Седгвик (Hull T. E., Enright W. H., Fellen B. M., Sedgwick A. E.)
(1972) Comparing numerical methods for ordinary differential equations, *SIAM J. Numer. Anal.*, 9.

Хемминг (Hamming R. W.)
(1962) Numerical methods for scientists and engineers. New York: McGraw-Hilll. (Русский перевод: Хемминг Р. В. Численные методы для научных работников и инженеров. Пер. с англ.— М.: Наука, 1972.)

Хенричи (Henrici P.)
(1962) Discrete variable methods in ordinary differential equations. New York: Wiley.

Шампэнь, Гордон (Shampine L. F., Gordon M. K.)
(1975) Computer solution of ordinary differential equations. San Francisco: W. H. Freeman.

Шампэнь, Уотс, Давенпорт (Shampine L. F., Watts H. A., Davenport S. M.)
(1976) Solving non-stiff ordinary differential equations—the state of the art, *SIAM Review*, **18**, 376—441.

Эндрюс, Паттерсон (Andrews H. C., Patterson C. L.)
(1975) Outer product expansions and their uses in digital image processing, *American Mathematical Monthly*, **82**, 1—12.

Энрайт, Хал (Enright W. H., Hull T. E.)
(1976) Test results on initial value methods for non-stiff ordinary differential equations, *SIAM J. Numer. Anal.*, **13**, 944—961.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., 2, издательство «Мир».

Дж. Форсайт, М. Малькольм, К. Моулер

МАШИННЫЕ МЕТОДЫ МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ

Научный редактор И. А. Маховая
Младший научный ред. Н. С. Полякова
Художник А. С. Гейнце
Художественный редактор В. И. Шаповалов
Технический редактор Е. С. Потапенкова
Корректор Н. А. Гиря

ИБ № 1715

Сдано в набор 04.01.80. Подписано к печати 26.03.80.
Формат 60×90¹/₁₆. Бумага типографская № 2.
Гарнитура латинская. Печать высокая.
Объем 8,75 бум. л. Усл. печ. л. 17,50. Уч.-изд. л. 15,07.
Изд. № 1/0332. Тираж 31 000 экз. Зак. № 1076. Цена 1 р. 10 к.

ИЗДАТЕЛЬСТВО «МИР»
Москва, 1-й Рижский пер., 2

Ордена Октябрьской Революции
и ордена Трудового Красного Знамени
Первая Образцовая типография имени А. А. Жданова
Союзполиграфпрома при Государственном комитете СССР
по делам издательств, полиграфии и книжной торговли.
Москва, М-54, Валовая, 28