ИЗДАТЕЛЬСТВО
«МИР»

COMPUTER SOLUTION OF LINEAR ALGEBRAIC SYSTEMS

GEORGE E. FORSYTHE
Professor of Computer Science
Stanford University

CLEVE B. MOLER
Assistant Professor of Mathematics
University of Michigan

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

1967

Дж. Форсайт, К. Молер

ЧИСЛЕННОЕ РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Перевод с английского
В. П. ИЛЬИНА и Ю. И. КУЗНЕЦОВА

Под редакцией
Г. И. МАРЧУКА

ИЗДАТЕЛЬСТВО «МИР»
Москва 1969

УДК 517.9 + 512

Авторы этой небольшой книги — ведущие американские специалисты в области прикладной математики. В ней описаны современные методы решения линейных алгебраических систем на электронных вычислительных машинах. Изложение характеризуется как высоким теоретическим уровнем, так и конкретной практической направленностью.

Книга будет весьма полезна всем, кто связан с работой на вычислительных машинах, а также студентам, инженерам и научным работникам различных специальностей.

Редакция литературы по математическим наукам

Инд. 2-2-3
16-69

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Книга Дж. Форсайта и К. Молера «Численное решение систем линейных алгебраических уравнений» представляет собой весьма интересное явление в научной и технической литературе. Дело в том, что авторы поставили своей целью рассмотрение не общих вопросов теории решения линейных алгебраических систем, а только вычислительных ее аспектов, которые обычно отсутствуют в монографической литературе.

Между тем именно такие вопросы в конечном итоге определяют успех реализации вычислительных алгоритмов. Авторы книги являются крупными специалистами в области численного анализа, поэтому они особое внимание уделяли наиболее важным вопросам теории вычислений и приложениям.

Тщательность и глубина алгоритмической проработки отдельных задач делают книгу практическим руководством при решении задач линейной алгебры на современных ЭВМ. Для большинства рассматриваемых задач приведены не только сами численные алгоритмы, но и программы для них, написанные на нескольких алгоритмических языках. Такой стиль изложения материала позволяет использовать его в работе непосредственно, без дополнительных усилий. Мы надеемся, что этот стиль будет одобрен читателями.

Хотя книга невелика по объему, она содержит весьма большую информацию и, по нашему мнению, отражает современные тенденции в вычислительной математике в связи с использованием ЭВМ.

Эта книга, предлагаемая теперь вниманию читателей в русском издании, является хорошим введением в алгоритмические аспекты теории линейных алгебраических уравнений. Она может служить учебным пособием для студентов, инженеров и научных работников, имеющих дело с приложениями математики. Несомненно, что книга будет интересной для широкого круга читателей.

Г. И. Марчук

Посвящается Дж. Уилкинсону

ПРЕДИСЛОВИЕ

Настоящая монография написана главным образом как пособие для студентов, изучающих численный анализ и вычислительные машины. Она возникла из курса численного анализа для студентов старших курсов Стэнфордского университета, который велся по учебнику, где почти не рассматривались матричные задачи. Автор этого учебника рекомендовал читателю пользоваться готовыми программами вычислительного центра, а в сомнительных случаях — консультациями специалиста. Мы, наоборот, считаем, что студенты должны знать матричные программы, с тем чтобы уметь применять или модифицировать их в нужных случаях. Более того, наши студенты впоследствии сами писали или транслировали программы для вычислительных центров. Чтобы заполнить существующий пробел, мы подготовили заметки по матричным вычислениям, и часть этих заметок легла в основу данной монографии. Необходимые предварительные сведения, приведенные в разд. 1, включают элементы линейной алгебры и некоторые основы программирования.

Мы думаем, что знакомство с большей частью представленного здесь материала было бы полезно студентам и сотрудникам, занимающимся математическим программированием, статистикой, инженерными задачами и многими другими вопросами, в которых встречаются матрицы и вычисления.

Решение системы линейных алгебраических уравнений является одной из задач, наиболее часто встречающихся в расчетах. Так как линейные функции изучены лучше всего, то наиболее распространенной моделью функциональной зависимости является линейная. И это приводит к системам линейных уравнений. Кроме того, большинство методов решения

нелинейных задач сводится к последовательности линейных систем. Ясно поэтому, что каждая расчетная лаборатория должна иметь возможность быстро и точно решать линейные алгебраические системы. Но как это ни удивительно, такое требование часто не удовлетворяется.

Как мы показываем в разд. 6, линейные системы могут сильно отличаться друг от друга. Один важный тип систем, которому в этой монографии отдается предпочтение, — это системы с плотными хранящимися матрицами коэффициентов. Полное исследование важного алгоритма — гауссовского исключения с итерационным уточнением — составляет центральную тему книги. В последних разделах дается краткое описание других методов решения линейных и нелинейных систем.

По сравнению с другими конструктивными разделами математики численный анализ обладает следующими двумя особенностями:

(i) интерес к таким проблемам, как машинное время и требования алгоритмов к памяти,

(ii) анализ ошибок, вызываемых различными формами арифметических действий с ограниченным числом разрядов в машинах.

Первое соображение делает гауссовское исключение методом, предпочтительным для решения линейных систем. Однако существует много вариантов методов исключения, и анализ ошибок служит руководством к выбору одного из них.

В добавление к нашему описанию гауссовского исключения с численными примерами и анализом ошибок мы приводим конкретный вычислительный алгоритм, который считаем наилучшим. Для большей доступности мы даем его на трех языках: АЛГОЛ-60, ASA ФОРТРАН и PL/I. В разд. 17 мы обращаем особое внимание на те места, где различия в языках или процессах вызывают трудности при транслировании программ. Мы думаем, что эти программы и сопровождающее их обсуждение являются важной составной частью.

Мы посвящаем нашу книгу Уилкинсону, который особенно много сделал для численного анализа. Специалисту лучше

всего изучить две его книги по матричным вычислениям. Однако менее подготовленному студенту в обширном материале этих книг разобраться нелегко. Главная часть настоящей монографии посвящена разъяснению и мотивировке некоторых алгоритмов и средств анализа, предложенных Уилкинсоном.

Мы хотим выразить признательность за особую помощь Хеммингу, который показал нам, как мало мы знаем о масштабировании матриц, Херриоту, ознакомившемуся с первоначальными заметками и высказавшему ряд критических замечаний, Мак-Киману, опубликовавшему ранние варианты программы на АЛГОЛе из разд. 16, Парлетту, который прорецензировал готовую рукопись, а также Бауэру, Бузингеру, Голубу, Кахану и Уилкинсону, предложившим уточнения в разных частях рукописи. Мы также благодарны сотрудникам вычислительных центров Стэнфордского и Мичиганского университетов, Швейцарской высшей технической школы в Цюрихе и Исследовательской лаборатории в Рюшликоне, Швейцария, за их помощь в опробовании вычислительных программ.

Джордж Е. Форсайт
Клив Б. Молер

1. О ПРЕДПОЛАГАЕМОМ ЧИТАТЕЛЕ И ЦЕЛИ КНИГИ

Мы предполагаем, что читатель прослушал курс линейной алгебры, обычно читаемый студентам-математикам. Следовательно, мы ожидаем, что он знаком с матрицами, представляющими линейные преобразования в данной системе координат, с векторами и их компонентами. Мы рассчитываем, что он знаком с понятиями эквивалентности, конгруэнтности и подобия, а также с канонической формой матриц соответствующих преобразований. Мы предполагаем, что он усвоил основные сведения об определителях и системах линейных уравнений, о собственных значениях и собственных векторах матриц. Хороший обзор этого материала содержится в гл. I книги Фаддеевой [35].

Мы предполагаем, что читатель знаком с программированием на автоматических цифровых мащинах на языках ФОРТРАН, АЛГОЛ-60, PL/I, что он знает кое-что о численных расчетах и связанных с ними ошибках, но ничего не знает о вычислениях с матрицами.

Цель этой книги — дать возможность такому читателю как можно быстрее разобрать несколько хороших программ решения систем линейных уравнений, а также получить представление о возникающих при этом ошибках.

2. НОРМЫ ВЕКТОРОВ И МАТРИЦ

Мы предполагаем, что читатель достаточно знаком с алгеброй матриц, представляющих линейные преобразования. Для понимания численных матричных методов необходимо также знакомство с геометрическими представлениями и свойствами матриц как линейных преобразований, так как в расчетах мы непосредственно интересуемся значениями различных вычисляемых величин. Мы вводим эти понятия в этом и следующем разделах. Более подробное изложение с доказательствами можно найти в книге В. Н. Фаддеевой [35], однако теоремы (3.1) там нет.

Пусть $x=(x_1, x_2, \ldots, x_n)^T$ означает вектор-столбец в вещественном n-мерном пространстве R_n. Пусть x^T означает вектор-строку, транспонированную к x. Мы вводим понятие евклидовой длины, или *нормы*, вектора x, обозначаемой через $\|x\|$, определяя ее по формуле

$$\|x\| = \sqrt{|x_1|^2 + \ldots + |x_n|^2} = \sqrt{x^T x}. \tag{2.1}$$

Норма обладает следующими свойствами, аналогичными свойствам обычной длины в пространстве двух или трех изменений:

(2.2) $\|cx\| = |c| \cdot \|x\|$ для всех вещественных c и всех векторов x;

(2.3) $\|\theta\| = 0$ и $\|x\| > 0$, если $x \neq \theta$ (здесь θ означает нулевой вектор);

(2.4) $\|x+y\| \leqslant \|x\| + \|y\|$ для всех векторов x и y.

(Равенство в формуле (2.4) имеет место тогда и только тогда, когда x и y являются линейно зависимыми, т. е. коллинеарными векторами.)

Свойства (2.2) и (2.3) следуют непосредственно из (2.1). Доказательство свойства (2.4) мы оставляем читателю. Для того чтобы его провести, достаточно доказать известное неравенство Коши — Шварца — Буняковского

$$|x^T y| \leqslant \|x\| \cdot \|y\| \tag{2.5}$$

для любых x, y. Для доказательства неравенства (2.5) заметим, что оно является необходимым условием неотрицательности квадратичной формы

$$(\alpha x + \beta y)^T (\alpha x + \beta y) = \alpha^2 \|x\|^2 + 2\alpha\beta x^T y + \beta^2 \|y\|^2$$

для всех вещественных α и β.

Теперь мы определим норму вещественной матрицы A, имеющей n строк и n столбцов:

(2.6) $$\|A\| = \max_{x \neq \theta} \frac{\|Ax\|}{\|x\|}.$$

Отсюда непосредственно следует, что

(2.7) $\|cA\| = |c| \cdot \|A\|$ для всех вещественных c и всех A;

(2.8) $\|\Theta\| = 0$ и $\|A\| > 0$, если $A \neq \Theta$

(здесь Θ означает нулевую матрицу).

(2.9) $\|A+B\| \leqslant \|A\| + \|B\|$ для всех $(n \times n)$-матриц A и B;

Это точно те же свойства, что и свойства (2.2)—(2.4) для векторов. Таким образом, множество $(n \times n)$-матриц A может быть интерпретировано как нормированное векторное пространство размерности n^2. Из определения (2.6) также непосредственно вытекает, что

(2.10) $\|Ax\| \leqslant \|A\| \cdot \|x\|$ для всех A, x.

Более того, из него же следует, что неравенство (2.10) «неулучшаемо», т. е. для каждой матрицы A существует вектор x, такой, что

(2.11) $$\|Ax\| = \|A\| \cdot \|x\|.$$

Читателю предоставляется доказать, что из (2.6) следует

(2.12) $\|AB\| \leqslant \|A\| \cdot \|B\|$ для всех матриц A, B.

Именно свойства (2.10) и (2.12) делают нормы векторов и матриц столь полезными при изучении линейных отображений и в особенности при анализе ошибок в решении систем линейных уравнений.

Квадратная матрица A представляет линейное преобразование (отображение) каждого вектора x одного n-мерного пространства X в вектор $y = Ax$ другого n-мерного пространства Y. Определение (2.6) может быть выражено в следующей эквивалентной форме:

(2.13) $$\|A\| = \max_{\|x\|=1} \|Ax\|.$$

Можно считать, что формула (2.13) определяет $\|A\|$ как наибольшую длину вектора в образе $\{Ax\}$ единичной сферы $\{x : \|x\| = 1\}$ при преобразовании $x \to Ax$.

Напомним, что ортогональная матрица U характеризуется свойством $U^T U = U U^T = I$, где I — единичная матрица. Отображение $x \to Ux$ представляет вращение n-мерного простран-

ства, сопровождаемое иногда отражением относительно некоторой гиперплоскости.

Евклидова норма (2.1) соответствует обычному понятию длины в двух- и трехмерных пространствах и позволяет использовать нашу геометрическую интуицию в пространствах более высоких размерностей. Эта длина сохраняется ортогональными матрицами, так что $\|x\|=\|Ux\|$ для всех ортогональных матриц U.

Мы рассмотрели только одну из возможных векторных норм. Две другие, обычно используемые в численном анализе, таковы:

$$\|x\|_1=\sum_{i=1}^{n}|x_i| \tag{2.14}$$

и

$$\|x\|_\infty=\max_{1\leqslant i\leqslant n}|x_i|. \tag{2.15}$$

Эти нормы векторов удовлетворяют условиям (2.2)—(2.4) и, в свою очередь, порождают соответствующие новые матричные нормы, согласно определению (2.6). Известно (см., например, В. Н. Фаддеева [35]), что

$$\|A\|_1=\max_{1\leqslant j\leqslant n}\sum_{i=1}^{n}|a_{ij}|, \tag{2.16}$$

$$\|A\|_\infty=\max_{1\leqslant i\leqslant n}\sum_{j=1}^{n}|a_{ij}|. \tag{2.17}$$

Использование этих двух норм не позволяет нам так свободно пользоваться геометрической интуицией, и, кроме того, преобразования с сохранением этих норм осуществляются менее общим классом матриц, чем класс ортогональных матриц. Однако нормы (2.16) и (2.17) легче вычислить, чем евклидову норму, и они очень полезны во многих приложениях.

Почти все сказанное в этом разделе может быть перенесено на комплексные векторы и матрицы. Необходимо только заменить транспонирование x^T и A^T комплексно сопряженным транспонированием $x^H=\bar{x}^T$ и $A^H=\bar{A}^T$.

(2.18) Упражнение. Доказать (2.4), (2.12), (2.16) и (2.17).

(2.19) Упражнение. Показать, что для любой $(n\times n)$-матрицы A

$$\max_{i,j}|a_{ij}|\leqslant\|A\|\leqslant n\max_{i,j}|a_{ij}|.$$

3. ДИАГОНАЛЬНАЯ ФОРМА МАТРИЦЫ ПРИ ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЯХ С ОРТОГОНАЛЬНЫМИ МАТРИЦАМИ

Мы хотим изложить некоторые геометрические соображения относительно квадратной матрицы A, связанные с тем, что эта матрица представляет линейное преобразование одного евклидова n-мерного пространства в другое такое же пространство. Предположим, что в обоих пространствах заданы ортогональные системы координат. По мнению авторов следующая теорема (3.1) приводит к простейшей интерпретации линейного преобразования, представленного матрицей. Напомним, что ортогональная матрица была определена в разд. 2.

(3.1) Теорема. *Для любой вещественной $(n\times n)$-матрицы A существуют две вещественные ортогональные $(n\times n)$-матрицы U и V, такие, что U^TAV диагональная матрица D. Более того, можно выбрать U и V так, чтобы диагональные элементы D имели вид*

$$\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_r > \mu_{r+1} = \ldots = \mu_n = 0,$$

где r — ранг матрицы A. В частности, если A невырожденна, то

$$\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_n > 0.$$

Тогда числа $\mu_1, \ldots, \mu_n$ называются *сингулярными числами* матрицы A. Как будет показано в этом разделе, они представляют собой неотрицательные квадратные корни (обязательно неотрицательных) собственных значений симметричной матрицы AA^T, где A^T — матрица, транспонированная к A. Практически все сведения для решения системы уравнений с матрицей A связаны с характером множества сингулярных чисел этой матрицы. Доказательство теоремы (3.1) дано в разд. 4.

Авторы руководств по линейной алгебре, касаясь преобразований эквивалентности, обычно при формулировке теоремы (3.1) рассматривают вместо U^T, V произвольные невырожденные матрицы P^{-1}, Q, а вместо $\mu_1, \ldots, \mu_r$ — единицы. Для вычислительных целей ортогональные матрицы имеют гораздо большую ценность, так как $\|Ux\| = \|x\|$ для любых ортогональных матриц U и любых векторов x. Следовательно, умножение на ортогональные матрицы сохраняет длину вектора, тогда как умножение на произвольную невырожденную матрицу P может изменить ее очень сильно. Это важное отличие.

Чтобы понять значение теоремы (3.1), рассмотрим матрицу A, представляющую линейное преобразование одного n-мерного пространства X в другое такое же пространство Y. Таким образом, $y=Ax$ принадлежит Y для любого x из X. Представляя линейное преобразование с помощью матрицы A, мы предполагаем, что как в X, так и в Y заданы ортогональные системы координат. Теперь рассмотрим ортогональное преобразование системы координат в пространстве X, в результате которого вектор x получит новое представление x', где $x=Vx'$. Таким же образом, применяя другое ортогональное преобразование координат в Y, мы получим новые координаты для y, именно y', где $y=Uy'$. Здесь как U, так и V — матрицы из (3.1).

В результате изменения базисов в X и Y преобразование, первоначально представленное матрицей A, будет представляться, как мы покажем, матрицей D. Действительно,

$$y' = U^T y = U^T A x = U^T A (V x') = \\ = (U^T A V) x' = D x'.$$

Таким образом, $y'=Dx'$, что и требовалось доказать.

В новой ортогональной системе координат преобразование имеет очень простое представление. Получаем для компонент

$$\begin{cases} y'_1 = \mu_1 x'_1, \\ y'_2 = \mu_2 x'_2, \\ \vdots \\ y'_r = \mu_r x'_r, \\ y'_{r+1} = 0, \\ \vdots \\ y'_n = 0. \end{cases} \tag{3.2}$$

Преобразование теперь просто отображает первую координатную ось X в первую координатную ось Y с коэффициентом растяжения $\mu_1>0$. То же самое проделывается со 2-й, 3-й, ..., r-й координатными осями пространства X, причем $\mu_2, \mu_3, \ldots, \mu_r$ играют роль коэффициентов растяжения. Последующие $(r+1)$-я, ..., n-я координатные оси X отображаются в нулевой вектор пространства Y.

Пусть A^T — матрица, транспонированная к A. Тогда

$$D^T D = (U^T A V)^T (U^T A V) = V^T A^T U U^T A V = V^T A^T A V.$$

Таким образом, $V^T(A^TA)V=D^TD$ есть диагональная матрица с диагональными элементами $\mu_1^2, \ldots, \mu_r^2, 0, \ldots, 0$. Так как V ортогональна, то $V^T=V^{-1}$ и преобразование $V^T(A^TA)V$ сохраняет собственные значения матрицы A^TA, которые, следовательно, равны $\mu_1^2, \mu_2^2, \ldots, \mu_r^2, 0, \ldots, 0$. Таким образом, сингулярные числа матрицы A есть неотрицательные квадратные корни собственных значений матрицы A^TA. Это последнее свойство часто используется для определения сингулярных чисел.

С помощью соотношений (3.2) мы можем показать, что D отображает единичную сферу $S=\{x': \|x'\|=1\}$ в r-мерный «гиперэллипсоид» $E=DS$ векторов y', таких, что

$$\frac{y_1'^2}{\mu_1^2}+\frac{y_2'^2}{\mu_2^2}+\ldots+\frac{y_r'^2}{\mu_r^2}\leqslant 1 \quad \text{и} \quad y'_{r+1}=\ldots=y'_n=0,$$

если $r<n$, и

$$\frac{y_1'^2}{\mu_1^2}+\frac{y_2'^2}{\mu_2^2}+\ldots+\frac{y_n'^2}{\mu_n^2}=1,$$

если $r=n$. Точка из E, наиболее удаленная от начала координат θ, имеет координаты $(\mu_1, 0, \ldots, 0)$. Если $r<n$, то E содержит начало координат θ. Если $r=n$, то E не содержит начала координат и точка из E, ближайшая к θ, имеет координаты $(0, \ldots, 0, \mu_n)$. Если $r<n$, то D и, следовательно, A — вырожденные матрицы. Если $r=n$, то D и A невырожденны и имеют обратные. В этом случае непосредственно из (3.2) мы видим, что

$$D^{-1}=\begin{bmatrix} \mu_1^{-1} & & 0 \\ & \ddots & \\ 0 & & \mu_n^{-1} \end{bmatrix}.$$

Таким образом, сингулярными числами матрицы A^{-1} являются $\mu_1^{-1}, \ldots, \mu_n^{-1}$.

Из предыдущего обсуждения и определения (2.6) мы заключаем, что

$$\|A\|=\|D\|=\mu_1. \tag{3.3}$$

Если $r=n$, то

$$\|A^{-1}\|=\|D^{-1}\|=\mu_n^{-1}. \tag{3.4}$$

Выделим теперь наиболее существенные свойства невырожденной квадратной матрицы A с сингулярными числами $\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_n > 0$. Имеется прямая L_1 в X, такая, что матрица A растягивает (или сжимает) L_1 в μ_1 раз. Существует другая прямая L_n, ортогональная к L_1, такая, что матрица A растягивает (или сжимает) ее в μ_n раз. Более того, AL_1 и AL_n ортогональны в Y. Единичная окружность в плоскости L_1 и L_n отображается матрицей A в эллипс с полуосями μ_1 и μ_n. Это наибольшее искажение, которое может произойти с любой окружностью в X.

Заметим, что определитель A, $\det(A)$, удовлетворяет условию

$$|\det(A)| = \mu_1\mu_2 \ldots \mu_n.$$

Доказательство следует из теоремы (3.1), из равенства

$$\det(A) = \det(U^T) \cdot \det(D) \cdot \det(V),$$

а также из факта, что определитель ортогональной матрицы равен ± 1.

(3.5) Упражнение. Пусть A — симметричная матрица. Доказать, что в теореме (3.1) сингулярные числа μ_i есть числа $|\lambda_i|$, взятые в должном порядке, где λ_i — собственные значения матрицы A.

4. ДОКАЗАТЕЛЬСТВО ТЕОРЕМЫ О ПРИВЕДЕНИИ К ДИАГОНАЛЬНОЙ ФОРМЕ

Для доказательства (3.1) заметим, что $B=AA^T$ — симметричная матрица. Более того, так как для любого вектора x

$$x^TBx = x^TAA^Tx = (A^Tx)^T(A^Tx) = y^Ty = \|y\|^2 \geqslant 0,$$

где $y=A^Tx$, мы видим, что B — положительно полуопределенная матрица. Это означает, что все ее n собственных значений неотрицательны, так что мы можем обозначать их через $\mu_1^2, \mu_2^2, \ldots, \mu_n^2$, где

$$\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_n \geqslant 0.$$

Предположим, что $\mu_r > 0$, но либо $r=n$, либо

$$\mu_{r+1} = \ldots = \mu_n = 0.$$

Из теории вещественных симметричных матриц следует, что мы можем найти ортогональную матрицу U, такую, что

$$U^TBU = D^2,$$

где D — диагональная матрица с диагональными элементами

$$d_{ii} = \mu_i \geqslant 0 \qquad (i=1, 2, \ldots, n).$$

Теперь определим $(n\times n)$-матрицу F как

$$F = U^TA. \tag{4.1}$$

Тогда $FF^T = (U^TA)(A^TU) = D^2$, так что матрица

$$FF^T = D^2 \tag{4.2}$$

является диагональной. При этом i-й диагональный элемент в равенстве (4.2) указывает, что норма i-й строки f_i матрицы F равна μ_i $(i=1, 2, \ldots, n)$, т. е. $\|f_i\| = \mu_i$. Более того, равенство нулю недиагональных элементов в (4.2) показывает, что различные строки матрицы F ортогональны друг другу. Так как $\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_r > 0$, первые r строк матрицы F — ненулевые попарно ортогональные векторы $f_1, \ldots, f_r$. Если $r<n$, то остальные строки $f_{r+1}, \ldots, f_n$ являются нулевыми векторами θ.

Мы теперь построим ортонормальную систему вектор-строк $v_1, \ldots, v_n$ следующим образом:

$$\text{для} \quad i=1, \ldots, r \quad \text{пусть} \quad v_i = \frac{1}{\mu_i}\cdot f_i. \tag{4.3}$$

Следовательно, $\|v_1\| = \ldots = \|v_r\| = 1$. Для $i=r+1, \ldots, n$ выберем в качестве v_i такие векторы с единичной нормой, чтобы

векторы $v_1, \ldots, v_n$ были взаимно ортогональны. Этот процесс построения ортонормального базиса, как доказывается в линейной алгебре, всегда возможен.

Пусть V^T — матрица, строками которой являются векторы $v_1, \ldots, v_n$:

$$V^T = \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ \cdot \\ v_n \end{bmatrix}.$$

Тогда ортонормальность векторов $\{v_i\}$ означает, что $V^TV = I$, так что V является ортогональной матрицей. Более того, мы видим из (4.3), что

$$f_i = \mu_i v_i \qquad (i = 1, \ldots, n).$$

Следовательно,

$$F = DV^T. \tag{4.4}$$

Но из (4.1) и (4.4) получаем, что

$$U^TA = DV^T,$$

откуда

$$U^TAV = D$$

и, таким образом, теорема (3.1) доказана.

Теорема (3.1) может быть распространена с небольшими изменениями на произвольную прямоугольную матрицу A. Для полноты мы сформулируем этот результат.

(4.5) Теорема. *Для любой вещественной матрицы A ранга r, имеющей n строк и k столбцов, существуют вещественная ортогональная $(n \times n)$-матрица U и вещественная ортогональная $(k \times k)$-матрица V, такие, что U^TAV является $(n \times k)$-матрицей вида*

$$D = \begin{bmatrix} \mu_1 & & & & & & 0 \\ & \mu_2 & & & & & \\ & & \cdot & & & & \\ & & & \cdot & & & \\ & & & & \mu_r & & \\ & & & & & 0 & \\ 0 & & & & & & 0 \end{bmatrix},$$

где $\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_r > 0$.

(4.6) Упражнение. Доказать (4.5). *Указание*: Следовать доказательству теоремы (3.1). Пусть

$$U^T A A^T U = \begin{bmatrix} D & \Theta \\ \Theta & \Theta \end{bmatrix}$$

и

$$U^T A = \begin{bmatrix} F \\ G \end{bmatrix}.$$

Показать, что $G = \Theta$ и т. д.

(4.7) Упражнение. Для произвольной невырожденной матрицы A найти расстояние до ближайшей вырожденной матрицы, т. е. найти минимум величины $\|A - S\|$ для всех вырожденных матриц S. (Ответ: μ_n.) Является ли ближайшая матрица единственной?

(4.8) Упражнение. Для произвольной вырожденной матрицы A найти расстояние до ближайшей невырожденной матрицы.

5. ТИПЫ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ В ЛИНЕЙНОЙ АЛГЕБРЕ

Мы сначала перечислим типы задач, рассматриваемых в этой книге или имеющих к ним отношение, а в дальнейшем обсудим их более детально. К вычислительным задачам относятся следующие.

(a) Решить линейную систему $Ax=b$, где A — данная невырожденная квадратная матрица порядка n (вещественная или, возможно, комплексная), b — данный вектор-столбец с n компонентами и x — неизвестный вектор-столбец с n компонентами.

(b) В предыдущей задаче иногда задается несколько правых частей b, например k, и требуется найти также k неизвестных векторов x. Если взять в качестве B $(n\times k)$-матрицу правых частей, а в качестве X — соответствующую $(n\times k)$-матрицу решений, то мы должны решить систему $AX=B$, где матрица A определена в п. (a).

(c) Найти обратную матрицу A^{-1} для данной невырожденной матрицы A.

(d) Для данной вещественной симметричной матрицы A найти некоторые или все ее (обязательно вещественные) собственные значения, а также, может быть, соответствующие собственные векторы x.

Иногда ставится та же задача для комплексной эрмитовой матрицы A (т. е. матрицы, которая равна транспонированной к ее комплексно сопряженной). В этом случае каждое собственное значение λ является вещественным, но собственные векторы x обычно комплексны.

(e) Для заданных вещественной симметричной матрицы A и вещественной симметричной положительно определенной матрицы C найти все или часть обобщенных собственных значений. *Обобщенным собственным значением* называется число λ, такое, что существует вектор x, удовлетворяющий уравнению $Ax=\lambda Cx$. Иногда необходимо вычислить также соответствующие векторы x.

(f) Для данной произвольной вещественной (или даже комплексной) матрицы A найти некоторые или все собственные значения λ и, возможно, также соответствующие собственные или корневые векторы. (Даже для вещественной матрицы A все λ могут быть комплексными.)

Напомним, что для каждого собственного значения λ матрицы A существует по крайней мере один собственный вектор x, такой, что $Ax=\lambda x$. Если λ имеет алгебраическую кратность m, бо́льшую единицы, то не обязательно существуют еще какие-либо собственные векторы, соответствующие λ и линейно не зависящие от x. Если x является единственным собственным вектором, принадлежащим λ, то существуют так называемые *корневые векторы* $y_1, y_2, \ldots, y_{m-1}$ (не однозначно определенные), такие, что

$$(5.1)\qquad \begin{cases} Ax=\lambda x, \\ Ay_1=\lambda y_1+x, \\ Ay_k=\lambda y_k+y_{k-1} \qquad (k=2, \ldots, m-1). \end{cases}$$

Этот общий случай осложняется наличием блоков в канонической жордановой форме матрицы A.

(g) Иногда мы должны отыскивать некоторые или все λ, для которых уравнение

$$\lambda^2 Ax+\lambda Bx+Cx=\theta$$

имеет вектор-решение x, где A, B и C — данные квадратные матрицы. Когда A и C вырожденные, проблема становится значительно сложнее.

(h) Для заданных $(n\times k)$-матрицы C, где $n>k$, и вектор-столбца d с n компонентами найти вектор-столбец x с k компонентами, такой, чтобы норма $\|Cx-d\|$ вектора невязки $Cx-d$ была по возможности меньше. Этот вектор x является *решением по методу наименьших квадратов* (обычно) несовместной системы уравнений $Cx=d$. Если ранг матрицы C не равен k, то существует бесконечное число векторов x, являющихся решением по методу наименьших квадратов. Иногда требуется найти среди них решение x, обладающее наименьшей нормой $\|x\|$. Такой вектор всегда единственный.

(i) Существует большое число задач, связанных с линейными неравенствами вида $Ax\geqslant b$, где неравенство справедливо для каждой компоненты. Вот одна из задач: для заданных $(n\times k)$-матрицы A $(n>k)$, вектор-столбца b с n компонентами и вектор-столбца c с k компонентами найти вектор-столбец x, такой, чтобы $Ax\geqslant b$, $x\geqslant\theta$ и чтобы число c^Tx было по возможности меньше. Поскольку эти задачи требуют совершенно другого подхода, а также по историческим причинам, мы их не будем здесь рассматривать. Они изучаются в

руководствах по линейному программированию, например у Данцига [13].

(5.2) Упражнение. Пользуясь теоремой (3.1), доказать утверждения из п. (h), т. е. показать, что если ранг матрицы меньше k, то существует бесконечное число решений x по методу наименьших квадратов, но что наименьшее решение единственно.

(5.3) Упражнение. Показать, что в (5.1) мы можем всегда выбрать векторы $x, y_1, \ldots, y_{m-1}$ так, чтобы они были взаимно ортогональны.

6. ТИПЫ МАТРИЦ, ВСТРЕЧАЮЩИХСЯ В ПРАКТИЧЕСКИХ ЗАДАЧАХ

Квадратная матрица A порядка n состоит из n^2 элементов a_{ij}. Если только немногие элементы a_{ij} отличны от нуля, матрица называется *редкой*. Ясно, что при соответствующей кодировке ее можно представить много меньшим, чем n^2, количеством вещественных чисел, так как нулевые элементы нет необходимости запоминать. Матрица, большинство элементов которой отлично от нуля, называется *плотной* матрицей. Слово *плотность* используется для обозначения отношения числа ненулевых элементов к n^2.

Иногда, даже если ни один элемент не равен нулю, элементы a_{ij} могут вычисляться через аргументы i, j с помощью простого алгоритма. Такая матрица называется *порождающейся* матрицей, и ее элементы не требуют хранения n^2 вещественных чисел в памяти вычислительной машины. Если, наоборот, элементы матрицы представлены как n^2 вещественных чисел, матрица называется *хранящейся*. При этом не важно, являются ли некоторые элементы нулями или нет, так как нули тоже нужно запоминать.

Можно предложить более гибкую меру трудности представления матрицы, основанную на количественном измерении сложности алгоритма для вычисления матрицы. Такую меру *информационного содержания* матрицы можно грубо характеризовать числом ячеек запоминающего устройства, требующихся для восстановления всех элементов a_{ij}. Например, хранящиеся матрицы требуют приблизительно n^2 ячеек, тогда как матрица, определенная простой формулой типа

$$a_{ij} = 1/(i + j - 1) \qquad (i,\ j = 1,\ \ldots,\ n),$$

требует только нескольких ячеек. В дальнейшем мы не будем касаться этого вопроса.

Порядки n матриц, встречающихся на практике, весьма различны, от единицы до нескольких тысяч. Дж. Данциг упоминает о задачах линейного программирования с миллионом неравенств для 40 000 неизвестных. Варга [7] ссылается на Беттиса из Лаборатории атомной энергии, решившего систему линейных уравнений со 108 000 неизвестных. Несомненно, что могут быть решены и бо́льшие системы уравнений.

Очевидно, трудность представления матрицы с очень большим n существенно зависит от ее плотности или по крайней мере от ее информационного содержания. Если матрица

должна полностью запоминаться, то существует два критических параметра. Один определяется объемом оперативного запоминающего устройства вычислительной машины, обычно изготовленного из магнитных сердечников, а второй характеризуется объемом внешнего запоминающего устройства. Машины с оперативной памятью 32 000 кодов ограничивают n примерно 150-ю, так как обычно несколько тысяч кодов требуется для остающихся в памяти программ. Если n^2 превосходит объем оперативного запоминающего устройства, то должна использоваться внешняя память (барабаны, ленты, диски), причем возникают существенные проблемы организации обмена между оперативной и внешней памятью, связанные с разделением времени, накоплением на буфере и т. д.

При использовании внешней памяти можно решать задачи с плотными матрицами вплоть до 1000-го порядка. В действительности практическое ограничение при работе с матрицами в большей степени обуславливается временем выполнения операций, нежели временем обмена с внешними устройствами. (Если магнитные ленты используются для обращения плотной матрицы порядка 1000, то они изнашиваются настолько, что становятся непригодными для чтения прежде, чем завершится обращение.) В случаях когда n значительно превышает 1000, существенно, чтобы матрица была редкой.

Алгоритмы для решения различных вычислительных задач линейной алгебры различаются в соответствии с тем, меняют ли они в процессе счета элементы матрицы или нет. Наиболее подходящие методы для хранящихся матриц небольшого порядка, как правило, меняют матрицу. Однако такие методы, будучи применены к редким матрицам, в процессе счета увеличивают плотность матрицы. Следовательно, если матрица имеет высокий порядок и редкая, то в общем случае невозможно использовать методы, наиболее пригодные для небольших хранящихся матриц.

Говорят, что A — *ленточная матрица*, если $a_{ij}=0$ для всех $|i-j|>m$, потому что ненулевые элементы образуют полосу, или ленту, вдоль главной диагонали; число диагоналей в полосе равно $2m+1$, см. разд. 23. Ленточные матрицы возникают в тех случаях, когда каждая неизвестная величина связана только с несколькими другими. Примерами служат система уравнений, описывающих электрическую цепь, имеющую много точек, но относительно мало взаимосвязей, а также конечно-разностная аппроксимация дифференциальных уравнений, как обыкновенных, так и в частных производных.

(6.1) Определение. Матрица A называется *матрицей с диагональным преобладанием*, если

(6.2) $$|a_{ij}| \geqslant \sum_{j \neq i} |a_{ij}| \quad \text{для всех} \quad i,$$

причем строгое неравенство имеет место хотя бы для одного i.

(6.3) Определение. Матрица A называется *разложимой*, если существуют перестановка строк и (обычно другая) перестановка столбцов, такие, что матрица, полученная в результате этих перестановок, имеет вид

$$\begin{bmatrix} B & \Theta \\ C & D \end{bmatrix},$$

где B и D являются квадратными матричными блоками, а Θ — нулевая матрица.

(6.4) Определение. Матрица A называется *неразложимой*, если она не является разложимой.

(6.5) Упражнение. Доказать, что неразложимая матрица с диагональным преобладанием не может быть вырожденна. (Одно из доказательств см. в приложении.)

(6.6) Задача для исследования. Написать эффективную вычислительную программу для определения разложимости хранящейся матрицы.

7. ИСТОЧНИКИ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

Опираясь на классификацию разд. 5, мы укажем некоторые источники вычислительных задач линейной алгебры.

Решение систем линейных алгебраических уравнений, задача (a), является, по-видимому, наиболее часто встречающейся в вычислительном центре задачей. Математик-прикладник часто оказывается перед необходимостью выбора параметров при обработке данных. Например, он может интерполировать функцию по n заданным значениям с помощью полинома, n коэффициентов которого являются параметрами. Так как коэффициенты полинома влияют на значение полинома линейно, эта интерполяционная задача сводится к решению системы линейных алгебраических уравнений. В более сложных задачах, где параметры входят нелинейно, уравнения получаются нелинейные. Однако типичный путь решения нелинейной системы уравнений — это ее линеаризация и последующее решение линеаризованной системы, т. е. снова задача типа (a).

Чаще всего источником системы линейных уравнений является аппроксимация непрерывного функционального уравнения конечно-разностной задачей. Например, задачу Дирихле для дифференциального оператора Лапласа можно аппроксимировать большой системой простых конечно-разностных уравнений (см. разд. 24). Матрицы, связанные с разностными уравнениями, почти всегда большие и редкие.

Второй очень существенный источник систем линейных уравнений — решение линейных задач методом наименьших квадратов. Предположим, что матрица C, определенная в п. (h) разд. 5, имеет ранг k. Мы докажем ниже, что C^TC также имеет ранг k и, следовательно, является невырожденной и положительно определенной. Тогда задача состоит в том, чтобы минимизировать выражение

$$
\begin{aligned}
(7.1)\qquad \|Cx-d\|^2 &= (Cx-d)^T(Cx-d) = \\
&= x^TC^TCx - 2x^TC^Td + d^Td = \\
&= (C^TCx - C^Td)^T(C^TC)^{-1}(C^TCx - C^Td) - \\
&\quad - d^TC(C^TC)^{-1}C^Td + d^Td.
\end{aligned}
$$

Читатель может проверить эти равенства простым перемножением. Так как $(C^TC)^{-1}$ является положительно определенной матрицей, минимум в (7.1) достигается, когда $C^TCx - C^Td = 0$, т. е. когда x удовлетворяет так называемому

нормальному уравнению

$$C^TCx = C^Td.$$

Это уравнение представляет собой задачу типа (a) разд. 5.

Чтобы показать, что C^TC имеет ранг k, мы докажем более общий результат, а именно, что

(7.2) ранг C^TC равен рангу C.

Мы воспользуемся теоремой (4.5) о существовании ортогональных матриц U, V, таких, что

(7.3) $$U^TCV = D.$$

Мы видим теперь, что $C = UDV^T$, так что

$$C^TC = (VD^TU^T)\,UDV^T = VD^TDV^T.$$

Но ясно, что матрицы D^TD и D имеют один и тот же ранг r. Таким образом, матрицы C^TC и C имеют один и тот же ранг.

Системы линейных уравнений, возникающие из нормальных уравнений, обычно имеют малый порядок и плотные. Однако использование нормальных уравнений далеко не всегда является наиболее эффективным или точным способом решения задачи методом наименьших квадратов; см. Голуб [1], Бузингер и Голуб [6] и Голуб и Кахан [12].

Часто задачи, приводящие к системе линейных уравнений, характеризуются одинаковыми функциональными отношениями, но различными данными. Например, система конечно-разностных уравнений может иметь несколько вариантов граничных условий для тех же самых уравнений внутри области или обработка данных по методу наименьших квадратов может быть проделана для нескольких заданных векторов d при одних и тех же регулирующих параметрах, образующих матрицу C. Такие ситуации приводят к системам линейных уравнений вида $AX = B$, т. е. к задаче типа (b).

Мы увидим ниже в связи с итерационным уточнением, что иногда различные столбцы матрицы B будут приводить к различной длительности процесса решения. Например, второй столбец b_2 может быть вычислен через первый столбец b_1.

Обращение матрицы, задача (c), чаще всего встречается в статистических вычислениях, где обратная матрица важна сама по себе для оценки некоторых статистических параметров. В большинстве других практических задач нахождение обратной матрицы не является действительно необходимым, хотя может представлять большой интерес ее норма,

Если для данной матрицы A имеется много правых частей b, то обратная матрица A^{-1} фактически представляет «оператор влияния», который прямо переводит b в решение x системы $Ax=b$, т. е. $x=A^{-1}b$. По этой причине часто желательно получить заранее A^{-1} с тем, чтобы новый вектор b преобразовывался в x перемножением $A^{-1}b$. Однако, если A — редкая матрица, то обратная матрица A^{-1} обыкновенно является плотной, так что хотя A может храниться в малом количестве ячеек, обратная матрица A^{-1} будет требовать чрезмерного объема памяти. К счастью, имеются пути хранения данных, с помощью которых вектор $A^{-1}b$ для данного b может быть быстро вычислен без запоминания элементов матрицы A^{-1} и с меньшими ошибками округления, чем при умножении A^{-1} на b; см. разд. 18.

Задачи (f) и (g) обычно возникают при решении системы линейных однородных обыкновенных дифференциальных уравнений с постоянными коэффициентами. Если мы представим такую систему в виде $dz/dt=Az$, где $z=z(t)$ есть n-мерный вектор, то попытка найти экспоненциальное решение в виде $z(t)=x\cdot\exp(\lambda t)$, где x — постоянный n-мерный вектор, приводит прямо к задаче (f) для определения λ.

Таким же образом система второго порядка

$$A\frac{d^2z}{dt^2}+B\frac{dz}{dt}+Cz=\theta \tag{7.4}$$

приводит к задаче (g). Эта система особенно часто встречается при исследовании неконсервативных динамических систем, подобных системам автоматического регулирования, где имеется приток энергии и когда нет уверенности, что системы будут устойчивы. Процессы, происходящие в колебательных контурах, описываются системами типа (7.4), причем A часто является матрицей индуктивностей, B — матрицей сопротивлений, а C — матрицей полных проводимостей. В аналогичных механических системах A является матрицей масс (инерциальных коэффициентов), B — матрицей коэффициентов сопротивлений, а C — матрицей, характеризующей силы.

Во многих задачах $B=\Theta$, а A и C — симметричные положительно определенные матрицы. Тогда система (7.4) принимает вид

$$A\frac{d^2z}{dt^2}+Cz=\theta. \tag{7.5}$$

Пусть $z(t)=x\cdot\exp(i\omega t)$, где x — постоянный вектор, а ω — частота собственных колебаний системы. Тогда мы приходим к

системе

$$-\omega^2 Ax + Cx = \theta,$$

или, если положить $\lambda = 1/\omega^2$,

$$Ax = \lambda Cx, \tag{7.6}$$

т. е. к задаче (e) разд. 5. Таким образом, задача (e) обычно соответствует определению частот собственных колебаний консервативных динамических систем.

Во многих частных случаях матрица масс A является единичной матрицей. Тогда обычно полагают $\lambda = \omega^2$ и получают обыкновенную задачу на нахождение собственных значений (d):

$$Cx = \lambda x, \tag{7.7}$$

где C — положительно определенная матрица.

Другим источником задач типа (d) являются некоторые задачи факторного анализа. Здесь для заданной симметричной положительно полуопределенной матрицы R порядка n (матрица корреляций) ищется представление, насколько это возможно, в виде ff^T, где f — искомый вектор-столбец. Обозначим элемент R с индексами i, j через r_{ij}, а соответствующий элемент ff^T — через $f_i f_j$. Тогда мы ищем такое f, чтобы

$$\sum_{i,j=1}^{n} (r_{ij} - f_i f_j)^2 = \text{minimum}.$$

Простые вычисления показывают, что минимум имеет место для некоторого вектора f, такого, что Rf пропорционально f, т. е. f является собственным вектором R. Можно показать, что если собственными значениями R являются

$$\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n > 0$$

с соответствующими собственными векторами $u_1, \ldots, u_n$, то наилучшим выбором f является $\sqrt{\lambda_1} u_1$. Таким образом, опуская детали, мы утверждаем, что эта область факторного анализа приводит к задаче на нахождение собственных значений для вещественной симметричной матрицы, т. е. к задаче (d).

(7.8) Упражнение. Доказать приведенные выше утверждения относительно f.

В этой книге мы ограничимся численным решением задач типа (a), (b) и (c).

8. ОБУСЛОВЛЕННОСТЬ ЛИНЕЙНОЙ СИСТЕМЫ

Рассмотрим систему уравнений $Ax=b$, где A — невырожденная матрица порядка n, т. е. $\det(A)\neq 0$ или, что эквивалентно, в теореме (3.1) все $\mu_i>0$. Так как такая матрица представляет взаимно однозначное отображение R_n в R_n, она имеет единственную обратную матрицу. Это означает, что система имеет единственное решение x, которое мы можем записать в форме $A^{-1}b$.

Предположим, что исходные данные (элементы A и b) в какой-то степени неопределенны, и мы хотим знать, как эта неопределенность сказывается на решении x. Для начала предположим, что матрица A известна точно, а вектор b — с некоторой неопределенностью. Например, $b+\delta b$ — другая правая часть, близкая к b, и $A(x+\delta x)=b+\delta b$. Посмотрим, как велико может быть δx. Имеем

$$\delta x = A^{-1}\delta b.$$

Следовательно,

$$\|\delta x\| \leqslant \|A^{-1}\|\cdot\|\delta b\|, \tag{8.1}$$

и для некоторых векторов δb возможно равенство. Так как $b=Ax$, то

$$\|b\| \leqslant \|A\|\cdot\|x\|. \tag{8.2}$$

Умножим (8.1) на (8.2):

$$\|\delta x\|\cdot\|b\| \leqslant \|A\|\cdot\|A^{-1}\|\cdot\|x\|\cdot\|\delta b\|. \tag{8.3}$$

Предполагая только, что $b\neq\theta$, мы находим из (8.3), что

$$\frac{\|\delta x\|}{\|x\|} \leqslant \|A\|\cdot\|A^{-1}\|\cdot\frac{\|\delta b\|}{\|b\|}. \tag{8.4}$$

Для любой невырожденной матрицы A определим теперь ее *число обусловленности*, обозначаемое $\operatorname{cond}(A)$, как произведение $\|A\|\cdot\|A^{-1}\|$. (Таким образом, число обусловленности зависит от используемой нормы.) Напомним, что из условий (3.3) и (3.4) для евклидовой нормы следует, что

$$\operatorname{cond}(A) = \|A\|\cdot\|A^{-1}\| = \mu_1/\mu_n \geqslant 1, \tag{8.5}$$

где μ_1, μ_n — наибольшее и наименьшее сингулярные числа матрицы A соответственно. Таким образом, $\operatorname{cond}(A)$ предста-

вляет собой меру максимального искажения единичной сферы при применении линейного преобразования с матрицей A. Из неравенства (8.4) получаем, что

$$\frac{\|\delta x\|}{\|x\|} \leqslant \operatorname{cond}(A) \cdot \frac{\|\delta b\|}{\|b\|}. \tag{8.6}$$

Теперь $\|\delta b\|/\|b\|$ можно интерпретировать как меру относительной неопределенности в задании вектора b. Если, например, элементы b известны до пяти значащих десятичных цифр, то $\|\delta b\|/\|b\|$ равно приблизительно 10^{-4} или 10^{-5}. Аналогично $\|\delta x\|/\|x\|$ можно интерпретировать как относительную неопределенность решения x, обусловленную неопределенностью вектора b. Неравенство (8.6) означает, что $\operatorname{cond}(A)$ ограничивает сверху отношение относительной неопределенности решения x к относительной неопределенности вектора b. Так как, согласно (8.5), $\operatorname{cond}(A) \geqslant 1$, мы видим, что эта граница не может быть меньше 1.

Мы хотим подчеркнуть тот факт, что при подходящем выборе направлений векторов b и δb в формулах (8.1) и (8.2) возможно равенство, следовательно, равенство возможно также и в (8.6). Таким образом, нельзя дать более точную оценку, чем (8.6), для произвольных векторов b и δb независимо от их величины. Поскольку принципиально важно, чтобы читатель понял причины этого факта, мы продемонстрируем, каким образом может получиться равенство в (8.1), (8.2) и (8.6). С помощью теоремы (3.1) мы доказываем существование матриц U и V, таких, что $U^TAV=D$. Затем, как в разд. 3, мы вводим ортогональные преобразования координат в пространстве X решений x и в пространстве Y правых частей b, так что уравнение $Ax=b$ переходит в

$$Dx' = b', \tag{8.7}$$

или

$$\begin{cases} \mu_1 x'_1 = b'_1, \\ \mu_2 x'_2 = b'_2, \\ \cdot\ \cdot\ \cdot\ \cdot \\ \mu_n x'_n = b'_n. \end{cases} \tag{8.8}$$

В то же время уравнение $A\delta x=\delta b$ для неопределенностей в новой системе координат будет иметь вид

$$D\,\delta x' = \delta b', \tag{8.9}$$

или

$$(8.10)\qquad \begin{cases} \mu_1 \delta x'_1 = \delta b'_1, \\ \mu_2 \delta x'_2 = \delta b'_2, \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ \mu_n \delta x'_n = \delta b'_n. \end{cases}$$

Так как $\mu_1 \geqslant \mu_2 \geqslant \ldots \geqslant \mu_n > 0$, $\|A\| = \|D\| = \mu_1$ и $\|A^{-1}\| = \|D^{-1}\| = \mu_n^{-1}$, то видно, что в (8.1), (8.2) и (8.6) возможны равенства. Действительно, пусть $b' = (\beta, 0, \ldots, 0)^T$, так что $b'_1 = \beta$, но $b'_i = 0$ $(i \geqslant 2)$. Тогда

$$x' = (\mu_1^{-1}\beta,\ 0,\ \ldots,\ 0)^T,$$

и мы имеем в формуле (8.2) равенство независимо от величины ненулевой константы β. С другой стороны, пусть $\delta b' = (0, \ldots, 0, \gamma)^T$, так что $\delta b'_1 = 0, \ldots, \delta b'_{n-1} = 0$, $\delta b'_n = \gamma$. Тогда

$$\delta x' = (0,\ \ldots,\ 0,\ \mu_n^{-1}\gamma)^T,$$

и мы имеем равенство в (8.1) независимо от величины ненулевой константы γ.

Короче говоря, мы получаем равенство в (8.6), если правая часть b имеет направление, которое растягивается матрицей A более всего (и, таким образом, растягивается матрицей A^{-1} менее всего), а возмущение δb имеет направление, которое растягивается матрицей A менее всего (и более всего растягивается матрицей A^{-1}). Эти два направления обязательно ортогональны, если $\mu_1 \neq \mu_n$. Число обусловленности $\operatorname{cond}(A)$ инвариантно относительно умножения матрицы A на константу и является довольно эффективной мерой того, насколько матрица A «хороша» по отношению к вычислениям, возникающим при решении уравнения. Заметим, что $\operatorname{cond}(A) = \operatorname{cond}(A^{-1})$.

Если $\operatorname{cond}(A) = 1$, то матрицы A и A^{-1} растягивают все направления одинаково, так как $\mu_1 = \mu_2 = \ldots = \mu_n$, и A^{-1} не может изменить длину вектора δb более, чем длину вектора b. Тогда $\|\delta x\|$ так же относится к $\|x\|$, как $\|\delta b\|$ к $\|b\|$. Относительная неопределенность x, таким образом, точно равна относительной неопределенности b. Однако, если $\operatorname{cond}(A) = 10^6$, то A^{-1} растягивает одно направление в миллион раз больше, чем другое. Если вектор b получает наименьшее растяжение, а вектор δb — наибольшее, то ясно, что $\|\delta x\|/\|x\|$ будет в миллион раз больше, чем $\|\delta b\|/\|b\|$. Если $\operatorname{cond}(A)$ относительно мало, то матрица A является *хорошо обусловленной* (по от-

ношению к задаче решения линейных уравнений). Если $\operatorname{cond}(A)$ относительно велико, то матрица A является *плохо обусловленной* (по отношению к этой же задаче).

Широко распространено заблуждение, что малость $\det(A)$ влечет за собой плохую обусловленность матрицы A. Из наших рассуждений следут, что это не так. Если, например,

$$\mu_1 = \mu_2 = \ldots = \mu_n = 10^{-30},$$

то $\det(A) = 10^{-30n}$ (очень маленькое число), но относительная неопределенность $\|\delta x\|/\|x\|$ точно равна $\|\delta b\|/\|b\|$. Если μ_1 и n фиксированы, то, действительно, стремление к иулю $\det(A)$ влечет за собой стремление к нулю μ_n и, следовательно, $\operatorname{cond}(A) \to \infty$. Таким образом, малость $\det(A)$ приобретает некоторую связь с плохой обусловленностью матрицы A фиксированного порядка n, если мы нормируем A как-нибудь так, чтобы μ_1 оставалось фиксированным. Чтобы посмотреть, насколько, однако, слабой может быть эта связь, предположим, что $n = 101$, $\mu_1 = 1$ и $\mu_2 = \ldots = \mu_n = 10^{-1}$. Тогда $\det(A) = \mu_1 \mu_2^{n-1} = 10^{-100}$, в то время как $\operatorname{cond}(A) = \mu_1 \mu_2^{-1} = 10$. Таким образом, матрица A очень хорошо обусловлена, хотя ее определитель мал.

До сих пор мы предполагали, что матрица A известна точно, а вектор b имеет неопределенность, и нашли, что $\operatorname{cond}(A)$ являлось решающим критерием. Оказывается, что $\operatorname{cond}(A)$ столь же важно, если A имеет неопределенность, а вектор b известен точно. Действительно, если $x = A^{-1}b$ и

$$x + \delta x = (A + \delta A)^{-1} b, \tag{8.11}$$

то мы видим, что

$$\delta x = \left[(A + \delta A)^{-1} - A^{-1}\right] b. \tag{8.12}$$

Полагая $B = A + \delta A$ в тождестве

$$B^{-1} - A^{-1} = A^{-1}(A - B) B^{-1}, \tag{8.13}$$

находим из (8.12), что

$$\delta x = -A^{-1}(\delta A)(A + \delta A)^{-1} b = -A^{-1}(\delta A)(x + \delta x). \tag{8.14}$$

Переходя в формуле (8.14) к нормам, получаем

$$\|\delta x\| \leqslant \|A^{-1}\| \cdot \|\delta A\| \cdot \|x + \delta x\|.$$

Окончательно,

$$\frac{\|\delta x\|}{\|x+\delta x\|} \leqslant \operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|}. \tag{8.15}$$

Таким образом, неопределенность x, отнесенная к $x+\delta x$, ограничена относительной неопределенностью матрицы A, умноженной на $\operatorname{cond}(A)$. Неравенство (8.15) нельзя заменить строгим.

8.16) Упражнение. Показать, что если $\|\delta A\|/\|A\|$ достаточно мало, то

$$\frac{\|\delta x\|}{\|x\|} \leqslant \operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|} \quad \text{(приближенно)}.$$

8.17) Упражнение. Напомним, что дифференциал dx есть линейная часть δx как функции элементов матрицы δA. Доказать, что

$$\frac{\|dx\|}{\|x\|} \leqslant \operatorname{cond}(A)\frac{\|\delta A\|}{\|A\|} \quad \text{(точно)}.$$

(8.18) Упражнение. Доказать неравенство

$$\frac{\|B^{-1}-A^{-1}\|}{\|B^{-1}\|} \leqslant \operatorname{cond}(A)\frac{\|A-B\|}{\|A\|},$$

показывающее, насколько мала норма $\|B^{-1}-A^{-1}\|$ в сравнении с нормой $\|A-B\|$. Неравенство было использовано Бауэром [2] для обоснования определения $\operatorname{cond}(A)$.

Заметим, что до сих пор мы не упоминали ошибки округления. Мы только обсуждали неопределенность решения линейной системы, обусловленную неопределенностью данных. Эта чувствительность решения к данным в задаче играет роль шумового фона; бесполезно считать решение x более правильным, если оно выражено с большей точностью, чем та, которую гарантирует область неопределенности. Фактически, как показано в разд. 20 и 21, мы можем включить ошибку округления в неопределенность данных, так как ошибка округления может анализироваться посредством $\operatorname{cond}(A)$.

Важно понять, что $\operatorname{cond}(A)$ может быть огромным — даже в простых задачах!

(8.19) Упражнение. Доказать, что если матрица A ортогональна, то $\operatorname{cond}(A)=1$ независимо от величины n.

(8.20) Пример. Пусть

$$A=\begin{bmatrix} 1 & 0{,}99 \\ 0{,}99 & 0{,}98 \end{bmatrix}.$$

Тогда $\lambda_2 \simeq -0{,}00005$, $\lambda_1 \simeq 1{,}98005$. Так как $A=A^T$, то $AA^T=A^2$ имеет собственные значения $(0{,}00005)^2$, $(1{,}98005)^2$. Тогда $\mu_2 \simeq 0{,}00005$, $\mu_1 \simeq 1{,}98005$ и $\operatorname{cond}(A)=$ $=39\,600$. Таким образом, A сильно искажает плоскость, и можно ожидать при решении серьезных неприятностей. Рассмотрение продолжим на системе

$$\begin{cases} x_1+0{,}99x_2=1{,}99, \\ 0{,}99x_1+0{,}98x_2=1{,}97. \end{cases}$$

Точным решением служит $x_1=1$ и $x_2=1$. Однако $x_1=$ $=3{,}0000$ и $x_2=-1{,}0203$ являются решением системы

$$\begin{cases} x_1+0{,}99x_2=1{,}989903, \\ 0{,}99x_1+0{,}98x_2=1{,}970106. \end{cases}$$

Таким образом,

$$\text{изменение } \delta b=\begin{bmatrix} -0{,}000097 \\ +0{,}000106 \end{bmatrix} \text{ дает } \delta x=\begin{bmatrix} +2{,}0000 \\ -2{,}0203 \end{bmatrix}.$$

Так как $\|\delta x\| \simeq 2\sqrt{2} \simeq 2{,}82$ и $\|x\|=\sqrt{2}$, мы имеем $\|\delta x\|/\|x\| \simeq 2$ и $\|\delta b\| \simeq 10^{-4}\sqrt{2}$, $\|b\| \simeq 2\sqrt{2}$, $\|\delta b\|/\|b\| \simeq$ $\simeq 10^{-4}/2$. Следовательно, $\|\delta x\|/\|x\|=40\,000\ \|\delta b\|/\|b\|$. Так как $\operatorname{cond}(A) \simeq 39\,600$, то мы нашли почти наихудший случай. Если компоненты вектора b известны с неопределенностью около 0,0001, то вектор x может быть вычислен только с ошибкой порядка 2. Это очень плохо обусловленная задача, потому что две линии, пересечения которых мы ищем, практически совпадают.

Стоит подчеркнуть еще раз, что значение $\operatorname{cond}(A)$ является гораздо более важным критерием трудности решения линейной системы $Ax=b$, чем малость $\det(A)$, либо громадность порядка n.

Если известно, что $\|A\|=1$, то очевидно, что $\operatorname{cond}(A)=$ $=\|A^{-1}\|$. Грубо говоря, если элементы матрицы $\|A\|$ находятся в области от 0,1 до 1,0, то обусловленность A определяется величиной элементов A^{-1}. Таким образом, если матрица масштабирована так, что ее элементы близки к единице, надежным признаком плохой обусловленности A является тот факт, что некоторые или все элементы A^{-1} велики.

(8.21) Упражнение. Если $A=\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ невырожденна, то определим

$$\sigma=\frac{a^2+b^2+c^2+d^2}{2\,|\,ad-bc\,|}.$$

Доказать, что $\operatorname{cond}(A)=\sigma+\sqrt{\sigma^2-1}$. (Набросок решения см. в приложении.)

(8.22) Упражнение. Доказать, что матрица

$$\begin{bmatrix} 100 & 99 \\ 99 & 98 \end{bmatrix}$$

и ее всевозможные перестановки имеют наихудшую обусловленность из всех невырожденных матриц второго порядка, элементы которых являются положительными целыми числами $\leqslant 100$. (Таким образом, пример (8.20) был выбран отнюдь не случайно!)

(8.23) Упражнение. Пусть A и A^{-1} имеют следующий вид:

$$A=\begin{bmatrix} 6 & 13 & -17 \\ 13 & 29 & -38 \\ -17 & -38 & 50 \end{bmatrix},\quad A^{-1}=\begin{bmatrix} 6 & -4 & -1 \\ -4 & 11 & 7 \\ -1 & 7 & 5 \end{bmatrix}.$$

Собственные значения A приближенно равны

$$\lambda_1=0{,}0588,\quad \lambda_2=0{,}2007,\quad \lambda_3=84{,}74.$$

(a) Описать множество $S=\{Ax\colon \|x\|=1\}$, т. е. образ единичной сферы при преобразовании A. Для проверки дать какие-либо числа, характеризующие размеры S.

(b)
$$\|A\|=?$$
$$\|A^{-1}\|=?$$
$$\operatorname{cond}(A)=?$$

(c) Рассмотрим систему уравнений $Ax=b$. Предположим, что мы имеем векторы b и x, относительно которых известно только, что

$$\|b-Ax\|\leqslant 0{,}01.$$

(i) Какова наименьшая верхняя граница для абсолютной ошибки $\|x-A^{-1}b\|$?

(ii) Какова наименьшая верхняя граница для относительной ошибки

$$\|x-A^{-1}b\|/\|A^{-1}b\|?$$

9. ГАУССОВСКИЙ МЕТОД ИСКЛЮЧЕНИЯ И LU-РАЗЛОЖЕНИЕ

Для системы линейных уравнений $Ax=b$ с плотными матрицами, элементы которых хранятся в оперативной памяти, не найдено алгоритмов решения, для которых доказано, что они лучше по времени или по точности, чем методы последовательного исключения Гаусса. Поэтому в каждой вычислительной библиотеке желательно иметь хорошие программы решения линейных систем методом исключения. В данном разделе описывается этот класс алгоритмов, а также даются некоторые их частные модификации.

Гауссовское исключение существует во многих вариантах, которые алгебраически тождественны. Методы отличаются характером хранения матриц, порядком исключения, способами предупреждения больших погрешностей округления и тем, как уточняются вычисленные решения. Имеются также варианты, специально приспособленные для систем с симметричными положительно определенными матрицами, которые хранятся в примерно вдвое меньшем объеме.

(9.1) Определение. *Нижняя треугольная* матрица есть квадратная матрица $C=(c_{ij})$, такая, что $c_{ij}=0$ для $i<j$. Аналогично, если $c_{ij}=0$ для $i>j$, C есть *верхняя треугольная* матрица.

Алгебраической основой гауссовского исключения является следующая теорема:

(9.2) LU-теорема. *Пусть дана квадратная матрица A порядка n и A_k означает главный минор матрицы, составленный из первых k строк и столбцов. Предположим, что* $\det(A_k)\neq 0$ *для* $k=1, 2, \ldots, n-1$. *Тогда существуют единственная нижняя треугольная матрица* $L=(m_{ij})$, *где* $m_{11}=m_{22}=\ldots=m_{nn}=1$, *и единственная верхняя треугольная матрица* $U=(u_{ij})$, *такие, что* $LU=A$. *Более того,* $\det(A)=u_{11}u_{22}\ldots u_{nn}$.

Для доказательства этой теоремы мы используем метод математической индукции. Если $n=1$, очевидно, представление $a_{11}=1\cdot u_{11}$ однозначно и $\det(A)=u_{11}$. Предположим, что теорема верна для $n=k-1$. Для $n=k$ мы разлагаем A на подматрицы:

$$A=\begin{bmatrix} A_{k-1} & c \\ r & a_{kk} \end{bmatrix},$$

где r есть строка из $k-1$ элементов, c — столбец с $k-1$ компонентами. Запишем

$$L=\begin{bmatrix} L_{k-1} & 0 \\ m & 1 \end{bmatrix}, \quad U=\begin{bmatrix} U_{k-1} & u \\ 0 & u_{kk} \end{bmatrix}.$$

Тогда

$$LU=\begin{bmatrix} L_{k-1}U_{k-1} & L_{k-1}u \\ mU_{k-1} & mu+u_{kk} \end{bmatrix}.$$

Согласно гипотезе, L_{k-1} и U_{k-1} определены однозначно и $L_{k-1}U_{k-1}=A_{k-1}$. Более того, ни L_{k-1}, ни U_{k-1} не вырожденны (иначе A_{k-1} будет вырожденна, что противоречит предположению). Тогда требование $LU=A$ эквивалентно тому, что $L_{k-1}u=c$, $mU_{k-1}=r$ и $mu+u_{kk}=a_{kk}$. Отсюда u, m и u_{kk} выражаются единственным образом, а L и U определены однозначно. Наконец,

$$\det(A)=\det(L)\cdot\det(U)=1\cdot\det(U_{k-1})\cdot u_{kk}=$$
$$=u_{11}\cdot u_{22}\ldots u_{k-1,\,k-1}\cdot u_{kk},$$

что завершает доказательство (9.2).

Аналогичным образом можно при тех же предположениях доказать, что A имеет единственное разложение вида $A=LDU$, где L и U суть верхняя и нижняя треугольные матрицы с единицами на диагоналях, а D — диагональная матрица. Более того, $\det(A)=d_{11}\ldots d_{nn}$. Действительно, U из LU-теоремы есть просто DU в LDU-теореме. Далее, можно показать, что если A симметрична и удовлетворяет условиям LU-теоремы, то из равенства $A=LDU$ следует, что $U=L^T$ (где L^T — матрица, транспонированная к L). Если A положительно определена, то по одной из теорем теории матриц $\det(A_r)>0$ для $r=1,2,\ldots,n$ и, следовательно, в LDU-теореме $d_{rr}>0$. Если мы положим

$$D^{1/2}=\begin{bmatrix} \sqrt{d_{11}} & & 0 \\ & \ddots & \\ 0 & & \sqrt{d_{nn}} \end{bmatrix},$$

то можно записать $G=LD^{1/2}$, и тогда $A=GG^T$. Итак, получаем:

(9.3) Следствие. *Если A — симметричная положительно определенная матрица, то она имеет единственное разложение вида GG^T, где G есть нижняя треугольная матрица с положительными диагональными элементами.*

Представление матрицы A в виде произведения LU является основной идеей гауссовских схем исключения, так как тогда система $Ax=b$ может быть записана как

$$LUx=b$$

и сводится к двум системам с треугольными матрицами

$$Ly=b \quad \text{и} \quad Ux=y,$$

которые легко решить. Компоненты промежуточного решения y могут быть получены из первой системы непосредственно, так как первое уравнение содержит только y_1, второе — только y_1 и y_2 и т. д. Затем компоненты x могут быть аналогично получены из второй системы в таком порядке: $x_n, x_{n-1}, \ldots, x_1$. Вычисление L и U вместе с решением системы $Ly=b$ обычно называется *прямым исключением*, а решение системы $Ux=y$ — *обратной подстановкой*. Мы будем также называть вычисление L и U *треугольным разложением*.

Различные методы отличаются порядком, в котором выполняются операции в прямом исключении. Важное различие состоит в том, переставляются ли уравнения (строки A) или нет, а также переставляются ли переменные (столбцы A). Перестановки будут обсуждаться в разд. 10, после чего мы опишем исключение более детально. Далее, иногда матрица L хранится, иногда нет. Важность запоминания L может быть легко продемонстрирована в случае, когда A есть общая хранящаяся матрица. Диагональ L не нужно запоминать, так как известно, что она состоит из единиц. Часть L, лежащую ниже диагонали, вместе с U можно расположить на месте, первоначально занятом A. Никакой промежуточной памяти не требуется, так как элементы L вычисляются в то же время, когда элементы A затираются. Почти все расчетное время, необходимое для решения системы $Ax=b$, тратится на нахождение L и U; вычисления, действительно использующие b, относительно малы. Поэтому, если впоследствии необходимо решить другую систему с той же матрицей A, но с новой правой частью b, имеет смысл сохранить L и U и таким образом избежать повторного треугольного разложения. Новые системы с той же матрицей A возникают, например, при вычислении обратной матрицы A^{-1}. Они также встречаются при уточнении первоначального решения методом итераций для нахождения x с высокой точностью. Этот метод мы опишем в разд. 13.

Свяжем предыдущее рассмотрение с обыкновенным методом исключения. Пусть даны матрицы A и вектор b четвертого порядка; проведем элементарные операции со строками

для того, чтобы элементы A ниже главной диагонали сделать равными нулю.

Предположим, что $a_{11} \neq 0$. Пусть $m_{i1} = a_{i1}/a_{11}$ $(i=2, 3, 4)$.

Для $i=2, 3, 4$ умножаем на m_{i1} первое уравнение и вычитаем из i-го уравнения, а также из b_i вычитаем m_{i1}, умноженное на b_1. Таким образом, мы получаем три уравнения, не содержащих x_1. Эти три новые уравнения вместе с первым уравнением в его первоначальном виде можно записать так:

$$A^{(2)}x = b^{(2)},$$

где $A^{(2)} = (a_{ij}^{(2)})$ и $a_{i1}^{(2)} = 0$ для $i=2, 3, 4$. Если M_1 есть нижняя треугольная матрица,

$$M_1 = \begin{bmatrix} 1 & & & 0 \\ -m_{21} & 1 & & \\ -m_{31} & 0 & 1 & \\ -m_{41} & 0 & 0 & 1 \end{bmatrix},$$

то $A^{(2)} = M_1 A$, $b^{(2)} = M_1 b$.

Теперь предположим, что $a_{22}^{(2)} \neq 0$. Пусть $m_{i2} = a_{i2}^{(2)}/a_{22}^{(2)}$ $(i=3, 4)$. Тогда умножаем $A^{(2)}$ и $b^{(2)}$ на

$$M_2 = \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & -m_{32} & 1 & \\ 0 & -m_{42} & 0 & 1 \end{bmatrix}$$

для получения $A^{(3)} = M_2 A^{(2)}$ и $b^{(3)} = M_2 b^{(2)}$. Это соответствует исключению x_2 из двух последних уравнений. Наконец, предполагая, что $a_{33}^{(3)} \neq 0$, берем $m_{43} = a_{43}^{(3)}/a_{33}^{(3)}$ и умножаем $A^{(3)}$ и $b^{(3)}$ на

$$M_3 = \begin{bmatrix} 1 & & & 0 \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & -m_{43} & 1 \end{bmatrix}.$$

Тогда $A^{(4)} = M_3 A^{(3)} = M_3 M_2 M_1 A$ есть ~~нижняя~~ верхняя треугольная матрица, которую мы обозначим через U. Исходная система теперь принимает форму $Ux = A^{(4)}x = b^{(4)} = M_3 b^{(3)}$ и имеет следую-

щую структуру:

$$(9.4)\qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ & & a_{33}^{(3)} & a_{34}^{(3)} \\ & 0 & & a_{44}^{(4)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{bmatrix}.$$

Пусть $M=M_3M_2M_1$. Тогда так как $MA=U$, то $A=M^{-1}U$. Но $M^{-1}=M_1^{-1}M_2^{-1}M_3^{-1}$. Нетрудно видеть, что M_k^{-1} есть просто матрица M_k, у которой знаки недиагональных элементов изменены на противоположные. Далее, произведение $M_1^{-1}M_2^{-1}M_3^{-1}$ есть просто

$$L=M^{-1}=\begin{bmatrix} 1 & & & 0 \\ m_{21} & 1 & & \\ m_{31} & m_{32} & 1 & \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix}.$$

Следовательно, M^{-1} есть L из LU-теоремы, т. е. $A=M^{-1}U=LU$. Заметим, что матрица M на самом деле не образовывалась, в действительности она выражается довольно сложно через m_{ij}. Достоинством метода исключения является то, что элементы m_{ij} матрицы L хранятся на месте поддиагональных элементов A, а элементы u_{ij} матрицы U располагаются на месте диагональных и наддиагональных элементов A. В конце мы получаем таблицу

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ m_{21} & u_{22} & u_{23} & u_{24} \\ m_{31} & m_{32} & u_{33} & u_{34} \\ m_{41} & m_{42} & m_{43} & u_{44} \end{bmatrix},$$

хранящуюся на месте A.

Преобразование правой части, т. е. переход от b к $b^{(4)}$, часто проводится одновременно с преобразованием A, но так как мы храним все необходимые множители, его можно с тем же успехом проводить позже как отдельную операцию.

Мы рассмотрели обыкновенное гауссовское исключение без перестановок. В треугольном разложении матрицы A L есть просто матрица множителей (m_{ij}) с единицами на диагонали, а U — матрица $A^{(4)}$ из (9.4). Заметим также, что промежуточным решением y является $b^{(4)}$, т. е. правая часть системы (9.4).

(9.5) Упражнение. Количество арифметических действий в матричных алгоритмах обычно измеряется числом мультипликативных операций, т. е. умножений и делений, так как аддитивных операций обычно производится столько же. (i) Подсчитать число мультипликативных операций, требуемых для решения системы $Ax = b$ n-го порядка с помощью гауссовского исключения и показать, что оно имеет вид полинома от n, в котором главный член равен $^1/_3 n^3$. (ii) Сколько мультипликативных операций требуется для решения системы с новой правой частью $Ax=b'$, если треугольное разложение запоминается? (iii) Если каждое умножение и деление занимают 25 *мксек*, то сколько времени будут выполняться действия в (i) и (ii) при решении системы сотого порядка? А тысячного?

(9.6) Упражнение. Вспомним, что правило Крамера выражает каждую компоненту решения x в виде отношения двух $(n\times n)$-определителей. Допустим, что каждый определитель находится как сумма $n!$ произведений, каждое по n сомножителей. Сколько мультипликативных операций потребуется в этом случае для порядков 100 и 1000? Сколько времени?

Клюев и Коковкин-Щербак [16] доказали, что в общем случае система линейных алгебраических уравнений не может быть решена с помощью меньшего числа операций, чем то, которое требуется в гауссовском исключении.

(9.7) Упражнение. Сформулировать и доказать UL-теорему, аналогичную LU-теореме (9.2)

(9.8) Упражнение. Доказать, что если A — симметричная вещественная невырожденная (но, возможно, неопределенная) матрица, то

$$A = GDG^T,$$

где G есть нижняя треугольная матрица с положительными диагональными элементами, а D есть диагональная матрица, у которой $|d_{ii}|=1$ для всех i.

10. ТРЕБОВАНИЯ К ПЕРЕСТАНОВКАМ СТРОК

В предыдущем алгоритме мы предполагали $a_{11} \neq 0$, $a_{22}^{(2)} \neq 0$, $a_{33}^{(3)} \neq 0$ (заметим, что $\det(A_r) = a_{11} \cdot a_{22}^{(2)} \ldots a_{rr}^{(r)}$, см. LU-теорему (9.2)). Мы не можем продолжать исключение в установленной форме, если какое-либо из этих чисел близко к нулю. Например, пусть «главный» элемент a_{11} равен нулю. Так как $\det(A) \neq 0$, мы знаем, что $a_{i1} \neq 0$ для некоторого $i > 1$. Если мы поменяем местами i-ю строку $[A, b]$ с первой строкой $[A, b]$, то получим эквивалентную систему уравнений с $a_{11} \neq 0$. После этого мы можем проводить алгоритм, описанный в разд. 9. Аналогичная процедура возможна на любой стадии, когда $a_{rr}^{(r)} = 0$.

Теоретически эта перестановка необходима только в том случае, когда главный элемент точно равен нулю. Однако вычислительные соображения подсказывают, что если действия с нулевым главным элементом a_{11} невозможны, то будет неосторожно использовать близкий к нулю главный элемент a_{11} при выполнении арифметических операций с ограниченной точностью. Чтобы убедиться в этом, рассмотрим следующий пример:

(10.1) Пример. Будем решать следующую систему, выполняя арифметические операции с тремя десятичными знаками на машине с плавающей запятой:

$$\begin{cases} 0{,}000100\ x_1 + 1{,}00\ x_2 = 1{,}00, \\ 1{,}00\ x_1 + 1{,}00\ x_2 = 2{,}00. \end{cases} \tag{10.2}$$

Истинное решение, округленное до указанных десятичных знаков, есть

$$x_1 = \frac{10\,000}{9\,999} = 1{,}00010; \quad x_2 = \frac{9\,998}{9\,999} = 0{,}99990.$$

Вот решение, полученное с помощью гауссовского исключения без перестановок:

$$\begin{cases} 0{,}000100\ x_1 + 1{,}00\ x_2 = 1{,}00, \\ \qquad\qquad -10\,000\ x_2 = 10\,000, \\ \qquad\qquad\qquad\qquad x_2 = 1{,}00, \\ \qquad\qquad\qquad\qquad x_1 = 0{,}00 \text{ (ужасно!)}. \end{cases}$$

А вот результат применения гауссовского исключения с перестановкой:

$$\begin{cases} 1{,}00\ x_1 + 1{,}00\ x_2 = 2{,}00, \\ \qquad\qquad 1{,}00\ x_2 = 1{,}00, \end{cases}$$
$$x_2 = 1{,}00,$$
$$x_1 = 1{,}00 \text{ (прекрасно!)}.$$

Эти примеры поясняют, что мы должны избегать малых по абсолютной величине главных элементов $a_{rr}^{(r)}$. Детальный анализ ошибок округления в разд. 21 показывает, что для общих матриц желательно поддерживать числа m_{ij} меньшими или равными единице по абсолютной величине. Поэтому следует выбрать в качестве главного элемента максимальное по абсолютному значению из чисел $a_{ir}^{(r)}$, $i \geqslant r$ (или одно из наибольших, если таких несколько). Это достигается на каждом этапе перестановкой r-й и i-й строк, после того как определено, что $a_{ir}^{(r)}$ есть максимальный по абсолютному значению элемент в столбце. Такая процедура эквивалентна перестановке соответствующих уравнений в исходной системе. Если множители m_{ij} не запоминаются и если $\det(A)$ вычисляться не должен, то никакой информации о перестановках хранить не нужно. Однако если множители нужно сохранить для дальнейшего использования, то необходимо знать, в какой строке находился главный элемент при исключении j-го столбца. И, конечно, знак $\det(A)$ меняется при каждой перестановке.

Следует отметить, что при автоматических вычислениях не обязательно переставлять строки в A и b. В самом деле, мы можем образовать вектор из целых чисел p_i $(i=1, 2, \ldots, n)$ и вызвать $a_{p_i j}$ вместо a_{ij}. Таким образом, компоненты вектора перестановок p_i могут применяться вместо действительных перестановок. Этот метод зависит от используемой вычислительной системы.

Если главный элемент выбирается максимальным по абсолютной величине в столбце, то метод называется *стратегией частичного упорядочивания* Уилкинсона [31] (иногда мы будем ее называть стратегией частичного выбора главного элемента). Исключение более сложного вида использует *стратегию полного упорядочивания* (стратегию полного выбора главного элемента), при которой в качестве главного берется элемент, максимальный по абсолютной величине во всей матрице оставшихся уравнений. Хотя некоторые положения легче доказать для стратегии полного упорядочивания, на практике

стратегия частичного упорядочивания оказывается вполне удовлетворительной; мы сосредоточим наше внимание на ней.

Существуют некоторые классы матриц, для которых полный удовлетворительный анализ округления может быть дан, даже если выбор главного не делается вовсе. Характерным примером является положительно определенная симметричная матрица A.

Анализ в разд. 21 показывает, что ошибки округления удивительно малы, если проводятся перестановки. Грубо говоря, решение x, вычисленное на машине с плавающей запятой, есть в действительности решение матричного уравнения $(A+\delta A)x=b$, где элементы δA не больше величины, равной приблизительно n отдельным ошибкам округления. Мы не можем сказать, что x близко к $A^{-1}b$. Однако если данные значения A и b имеют некоторую неопределенность, то неопределенность в x, вызванная погрешностью округлений в процессе решения, не превосходит величины, примерно в n раз большей, чем неопределенность в x, обусловленная грубостью введенных в машину данных.

Назовем $(n\times n)$-*матрицей перестановок* такую матрицу, у которой каждая строка и каждый столбец содержат точно одну единицу и $n-1$ нулей. Если P — матрица перестановок, то PA есть матрица A с переставленными строками, а AP является матрицей A с переставленными столбцами.

Учитывая возможность перестановки строк, мы можем в LU-теореме (9.2) опустить предположение об определителе и получаем следующее утверждение:

(10.3) Теорема. *Если — A произвольная квадратная матрица, то существует матрица перестановок P, нижняя треугольная матрица L с единичной диагональю и верхняя треугольная матрица U, такие, что*

$$LU=PA.$$

Однако L и U не всегда определяются однозначно матрицами P и A.

Конструктивное доказательство дано с помощью алгоритма *DECOMPOSE* в разд. 16, а также Вендроффом [8].

11. МАСШТАБИРОВАНИЕ УРАВНЕНИЙ И НЕИЗВЕСТНЫХ

В системе линейных уравнений $Ax=b$ неизвестные x_j и члены в правой части b_i часто имеют физический смысл. Возможно, x_j есть смещения, измеряемые в сантиметрах, а b_i есть силы, измеряемые в динах. Так как физические единицы могут быть произвольны, естественно спросить: что, если смещение, определяемое одним x_j, взять в километрах? В этом частном случае мы должны, конечно, сделать подстановку $x_j = 10^5 x'_j$. Тогда необходимо каждый член $a_{ij}x_j$ заменить на $10^5 a_{ij}x'_j$, а весь j-й столбец матрицы A умножить на 10^5.

Допустим, что в общем случае x_j заменено на $d_j^{(2)}x'_j$ для $j=1,\ldots,n$. Пусть D_2 — невырожденная диагональная матрица

$$D_2 = \begin{bmatrix} d_1^{(2)} & & 0 \\ & \ddots & \\ 0 & & d_n^{(2)} \end{bmatrix}. \tag{11.1}$$

Подстановки можно записать в виде

$$x = D_2 x'. \tag{11.2}$$

Аналогично, предположим, что

$$D_1 = \begin{bmatrix} d_1^{(1)} & & 0 \\ & \ddots & \\ 0 & & d_n^{(1)} \end{bmatrix} \tag{11.3}$$

есть вторая невырожденная матрица и что мы делаем подстановку

$$b = D_1 b' \tag{11.4}$$

для членов правой части системы $Ax=b$. Тогда система принимает вид

$$AD_2x' = D_1b'$$

или

$$D_1^{-1}AD_2x' = b'. \tag{11.5}$$

Таким образом, эта простая замена переменных дает нам новую линейную систему, матрица которой есть $A' = D_1^{-1}AD_2$, а правая часть $b' = D_1^{-1}b$.

(11.6) Определение. Матрица A' называется *диагонально эквивалентной* A, если существуют неособенные диагональные матрицы D_1 и D_2, такие, что

$$A' = D_1^{-1}AD_2. \tag{11.7}$$

Таким образом, $a'_{ij} = [d_i^{(1)}]^{-1}a_{ij}d_j^{(2)}$.

Очевидно, диагональная эквивалентность есть эквивалентность в общем математическом смысле (т. е. обладает свойством симметричности и транзитивности). Это также частный случай эквивалентности в том значении, которое используется в теории матриц (потому что D_1 и D_2 — специальные невырожденные матрицы).

Мы видим, что AD_2 есть просто матрица A со столбцами, умноженными на коэффициенты $d_1^{(2)}, \ldots, d_n^{(2)}$. Матрица $D_1^{-1}A$ есть матрица A со строками, умноженными на $1/d_1^{(1)}, \ldots, 1/d_n^{(1)}$.

Матрица $A' = D_1^{-1}AD_2$ есть результат преобразований столбцов и строк матрицы A. Так как произведение матриц ассоциативно, не имеет значения, производится ли раньше умножение строк или умножение столбцов. Более того, так как диагональные матрицы коммутативны, можно перейти от матрицы A к A' путем $2n$ умножений строк или столбцов матрицы A в любом порядке.

Обычно говорят, что $D_1^{-1}AD_2$ *эквивалентна по масштабу с* A. В частности, AD_2 *эквивалентна по масштабу столбцов с* A, *а* $D_1^{-1}A$ *эквивалентна по масштабу строк с* A.

Такие операции масштабирования обычно используются при решении систем линейных уравнений. Хотя масштабирование может иметь большое значение, обычно нет надобности подбирать $d_i^{(1)}$ и $d_j^{(2)}$ точно. При вычислениях с плавающей запятой с основанием β обычно достаточно взять $d_i^{(1)}$ и $d_j^{(2)}$ равными целым степеням β. Это изменяет только показатель степени числа с плавающей запятой a_{ij}, оставляя его *значащие цифры* (т. е. дробную часть, или мантиссу) неизменной. Таким образом, такое масштабирование не вносит погрешности округления.

(11.8) Определение. Матрица A' называется *эквивалентной по масштабу с основанием* β с матрицей A, если $A' = D_1^{-1}AD_2$, где D_1 и D_2 — диагональные невырожденные матрицы, элементы которых есть целые степени основания β.

Так как строки A могут переставляться практически без действительного перемещения в машинной памяти, строки и столбцы могут масштабироваться без действительного изменения a_{ij}. Необходимо запомнить только $2n$ чисел $d_i^{(1)}$ и $d_j^{(2)}$ или даже только показатели их степеней, если они равны целым степеням β.

Естественно, мы хотим знать, как влияет масштабирование на решение линейной системы. Этот вопрос не очень хорошо изучен, однако мы сформулируем один результат Бауэра [4].

(11.9) Теорема. *Пусть матрицы A и A', представленные в режиме с плавающей запятой, эквивалентны по масштабу с основанием β в смысле определения* (11.8). *Предположим, что $b = D_1b'$. Тогда, если индексы главных элементов и порядок их выбора зафиксированы заранее, то решения систем $Ax = b$ и $A'x' = b'$ по гауссовскому исключению при использовании арифметики с плавающей запятой будут вычисляться точно с теми же значащими разрядами во всех результатах и промежуточных величинах (если только показатель степени не вызовет переполнения или появления машинного нуля).*

Основное предположение здесь то, что порядок выбора главных элементов предопределен и одинаков для обеих систем. Естественно, решения x и x' связаны соотношением (11.2).

Докажем (11.9). Мы предполагаем, что округление, за исключением случая переполнения или появления машинного нуля, относится только к значащим цифрам, но не к показателю степени. Значащие цифры A и A' идентичны согласно гипотезе. Так как главные элементы в A и A' выбираются на одинаковых местах, арифметические операции на одинаковых стадиях решения выполняются с соответствующими элементами A, b и A', b'. Масштабирование столбца A вызывает только компенсирующее изменение показателя степени соответствующего неизвестного.

Масштабирование строки $[A, b]$ только вводит множитель в одно уравнение, и в процессе исключения появляются компенсирующие изменения показателей степеней. Мы не будем

больше говорить о масштабировании столбцов, а выделим эффект масштабирования строки путем рассмотрения стадии исключения x_1. На этой стадии мы имеем строки

$$a_i = [a_{i1}, \ldots, a_{in}, b_i] \quad \text{и} \quad a_k = [a_{k1}, \ldots, a_{kn}, b_k].$$

Предположим, что a_{i1} есть главный элемент для первого столбца. При исключении x_1 из k-й строки мы заменяем a_k на

$$a_k^{(2)} = a_k - \frac{a_{k1}}{a_{i1}} \cdot a_i.$$

Теперь предположим, что матрица $[A', b']$ получена умножением a_i на β^r, а a_k — на β^s. Тогда при решении системы $A'x' = b'$ мы образуем новую строку

$$\beta^s a_k - \frac{\beta^s a_{k1}}{\beta^r a_{i1}} \cdot \beta^r a_i,$$

которая точно равна $\beta^s a_k^{(2)}$. Следовательно, при масштабировании строки новая строка $a_k^{(2)}$ только умножается на β^s, множитель β^r не появляется. Такое масштабирование не оказывает существенного влияния на последующее исключение или на обратную подстановку. Это завершает доказательство (11.9).

Мы заключаем из теоремы (11.9), что единственно возможный эффект масштабирования матрицы целыми степенями β при исключении состоит в изменении выбора главных элементов. Дадим теперь два примера, чтобы показать, что масштабирование на самом деле может вызвать большое изменение результата. В обоих примерах мы берем $\beta = 10$, проводим действия с тремя цифрами и плавающей запятой и используем частичный выбор главных элементов.

(11.10) Пример. Берем пример (10.1) и умножаем первое уравнение на 10^5:

$$\begin{cases} 10{,}0x_1 + 100\,000x_2 = 100\,000, \\ 1{,}00x_1 + 1{,}00 \quad x_2 = 2{,}00. \end{cases}$$

Так как $10{,}0 > 1{,}00$, метод частичного выбора главного элемента не требует перестановки уравнений. Таким образом, исключение дает

$$\begin{cases} 10{,}0x_1 + 100\,000x_2 = 100\,000, \\ \qquad\quad - \ 10\,000x_2 = -10\,000. \end{cases}$$

Решая, получаем

$$x_2 = 1{,}00,$$
$$x_1 = 0{,}00 \text{ (ужасно!)}.$$

Таким образом, масштабирование первой строки приводит к выбору главного элемента из первого уравнения. Процесс исключения точно такой же, как в примере (10.1) без применения перестановок, которые, как мы видели, играют такую большую роль в этом примере. Сам этот пример служит также иллюстрацией к теореме (11.9), что читатель увидит, проведя сравнение с первым решением (10.1).

Так как масштабирование столбцов само не влияет на выбор главного элемента в первом столбце, мы можем переписать пример (11.10) двумя способами без существенного изменения результата:

$$\begin{cases} 10{,}0x_1 + \quad 1{,}00x_2 = 1{,}00_{10}5, \\ 1{,}00x_1 + 1{,}00_{10} - 5x_2 = 2{,}00 \end{cases}$$

или

$$\begin{cases} 1{,}00x_1 + \quad 1{,}00x_2 = 1{,}00_{10}5, \\ 0{,}10x_1 + 1{,}00_{10} - 5x_2 = 2{,}00. \end{cases}$$

(Здесь и ниже мы используем обозначения АЛГОЛа-60 для записи чисел с основанием 10. Таким образом, $1{,}00_{10}$—5 есть $1{,}00 \cdot 10^{-5}$.)

Несмотря на то что первоначальное решение, полученное в примере (11.10), очень плохое, уточнение с помощью итераций (рассматриваемое в разд. 13) дает правильный результат.

Во втором примере порядок равен трем.

(11.11) Пример. Рассмотрим систему

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 6, \\ x_1 - x_2 + x_3 = 1, \\ 2_{10} - 4x_1 + x_2 + x_3 = 2. \end{cases} \tag{11.12}$$

При правильном округлении решение есть

$$(1{,}0010,\ 1{,}0004,\ 0{,}9994).$$

Решая систему с помощью гауссовского исключения с частичным выбором главных элементов и проводя дей-

ствия с тремя знаками при плавающей запятой, мы получаем после прямого исключения

$$(11.13)\qquad \begin{cases} x_1+2x_2+3x_3=6, \\ \quad -3x_2-2x_3=-5, \\ \qquad\quad 0{,}332x_3=0{,}333. \end{cases}$$

Проводя обратную подстановку, мы получаем

$$x^{(0)}=(1{,}00,\ 1{,}00,\ 1{,}00).$$

Подставляя в (11.12), находим невязку $b-Ax^{(0)}=\theta$. Таким образом, при правильном округлении решение с тремя знаками получается легко.

Пусть A — матрица системы (11.12):

$$(11.14)\qquad A=\begin{bmatrix} 1 & 2 & 3 \\ 1 & -1 & 1 \\ 2_{10}-4 & 1 & 1 \end{bmatrix},\quad b=\begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix}.$$

Довольно громоздкие вычисления дают для обусловленности A сравнительно малое число — 27,4. Пусть

$$(11.15)\qquad D_1^{-1}=\begin{bmatrix} 1 & & \\ & 1 & \\ & & {}_{10}4 \end{bmatrix},\quad D_2=\begin{bmatrix} 1 & & \\ & {}_{10}-4 & \\ & & {}_{10}-4 \end{bmatrix}.$$

После масштабирования с помощью (11.7) и (11.15) система (11.12) приводится к виду

$$(11.16)\qquad \begin{cases} x_1+2_{10}-4x_2+3_{10}-4x_3=6, \\ x_1-{}_{10}-4x_2+{}_{10}-4x_3=1, \\ 2x_1+\qquad x_2+\qquad x_3=2_{10}4. \end{cases}$$

Решая ее с помощью частичного или полного выбора главных элементов, мы выбираем 2 в качестве первого главного элемента, исключаем x_1 и получаем

$$\begin{cases} 2x_1+\qquad x_2+\qquad x_3=\ 2_{10}4, \\ \quad -0{,}500x_2-0{,}500x_3=-{}_{10}4, \\ \quad -0{,}500x_2-0{,}500x_3=-{}_{10}4. \end{cases}$$

Исключение x_2 приводит к системе

$$(11.17)\qquad \begin{cases} 2x_1 + \quad x_2 + \quad x_3 = 2_{10}4, \\ \quad -0{,}500x_2 - 0{,}500x_3 = -_{10}4, \\ \qquad\qquad\qquad 0 = 0, \end{cases}$$

которая является вырожденной.

Таким образом, в этом примере масштабирование делает обычные способы выбора главных элементов неадекватными. По теореме Бауэра (11.9) можно удовлетворительно решить систему (11.16) с a_{11} в качестве первого главного элемента, так как это будет эквивалентно тому, что делалось при получении (11.13). Так как (11.17) — вырожденная система, мы не представляем себе, как получить решение для (11.16) из системы (11.17) с помощью итерационного уточнения (описанного в разд. 13). В этом смысле (11.11) представляет собой худший пример по сравнению с (11.10) и доказывает, что выбор нужного масштабирования очень сложен, если хотим получить программу для решения как можно большего числа систем линейных уравнений.

При изучении (разд. 8) влияния возмущения A и b на решение линейной системы решающим параметром была обусловленность матрицы A. В доказательстве (разд. 22) сходимости метода итерационного уточнения решения линейных систем основное предположение — это что число обусловленности достаточно мало.

Следовательно, теоретически наилучший способ масштабирования — это тот, который делает число обусловленности преобразованной матрицы наименьшим. Мы можем поставить тогда задачи:

(11.18) Дана матрица A; как выбрать диагональные матрицы D_1 и D_2, чтобы обусловленность $D_1^{-1}AD_2$ была минимальной?

(11.19) Дана матрица A. Можем ли мы определить оптимальные D_1 и D_2 из (11.18) или приемлемое приближение к ним с помощью достаточно быстрого вычислительного алгоритма?

Задача (11.18), оказывается, полностью решается для некоторых матричных норм. (Напомним, что $\operatorname{cond}(A) = \|A\| \cdot \|A^{-1}\|$, поэтому результат зависит от выбора нормы.) Задача не

решена для евклидовой нормы, которая главным образом используется в этой книге за исключением двух частных случаев: (i) когда $n=2$ для любой матрицы A, (ii) когда и A, и A^{-1} имеют определенное распределение знаков, называемое *шахматным* (определение этого понятия см. Бауэр [4]). Оптимальные D_1 и D_2 зависят существенно от A^{-1}. Так как цель нашего масштабирования матрицы A состоит в облегчении нахождения $A^{-1}b$, то мы, конечно, не знаем A^{-1}. Следовательно, мы не можем надеяться на использование найденного решения задачи (11.18) при решении задачи (11.19).

Мы сейчас приведем решение Бауэра [4] задачи (11.18) для максимум-нормы. Вспомним, что если $\|x\|_\infty$ определяется по формуле (2.15), то соответствующая матричная норма $\|A\|_\infty$ есть

$$\|A\|_\infty = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|. \tag{11.20}$$

Пусть для любой матрицы $B=(b_{ij})$ через $|B|$ обозначена матрица с неотрицательными элементами $|b_{ij}|$.

Пусть дана неразложимая матрица A (см. (6.4)). Положим $C=|A|\cdot|A^{-1}|$. Так как C имеет неотрицательные элементы, то одно из ее максимальных по модулю собственных значений есть простое вещественное положительное число (см. Варга [7]); обозначим его через ρ. Матрица C имеет также собственный вектор $u=(u_1, \ldots, u_n)^T$, соответствующий ρ, так что $Cu=\rho u$. Пусть $d_i^{(1)}=u_i$ для $i=1, \ldots, n$. Матрица D_1 имеет вид (11.3) с указанными $d_i^{(1)}$. Аналогично, для матрицы $|A^{-1}|\cdot|A|$ ρ является единственным максимальным по модулю простым положительным собственным значением и соответствующий собственный вектор есть $v=(v_1, \ldots, v_n)^T$. Положим $d_j^{(2)}=v_j$ для $j=1, \ldots, n$. Тогда матрица D_2 дается формулой (11.1) с указанными $d_j^{(2)}$. Определенные сейчас матрицы D_1 и D_2 минимизируют обусловленность $D_1^{-1}AD_2$. (Так как $\operatorname{cond}(B)$ не изменяется от умножения B на скаляр, масштабирование векторов u и v не имеет отношения к задаче. То есть D_1 и D_2 определены однозначно только с точностью до постоянного множителя.) Это завершает решение Бауэра задачи (11.18) в норме $\|x\|_\infty$.

Решение в евклидовой норме $\|x\|$ (для специальных матриц A) также содержит матрицы $|A|\cdot|A^{-1}|$ и $|A^{-1}|\cdot|A|$ и их собственные векторы, но является более сложным. Масштабирование, хорошее для нормы $\|x\|_\infty$, должно быть удовлетвори-

тельным для нормы $\|x\|$, так как можно показать, что для матрицы порядка n

$$\frac{1}{n} \leqslant \frac{\operatorname{cond}_\infty(A)}{\operatorname{cond}(A)} \leqslant n. \tag{11.21}$$

Здесь $\operatorname{cond}(A) = \|A\| \cdot \|A^{-1}\|$ для нормы $\|x\|$, а $\operatorname{cond}_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty$. Но в любом случае мы не знаем удовлетворительного практического решения задачи масштабирования (11.19), которое работало бы для любых матриц A в произвольной норме.

(11.22) Упражнение. Доказать (11.21).

Из любопытства наш коллега Варах провел экспериментальное определение D_1 и D_2, таких, что $\operatorname{cond}(D_1^{-1}AD_2)$ минимизируется в евклидовой норме для матрицы A из (11.14). Он нашел, что

$$D_1^{-1} \simeq \begin{bmatrix} 1 & & 0 \\ & 1{,}7318 & \\ 0 & & 2{,}9998 \end{bmatrix},$$

$$D_2 \simeq \begin{bmatrix} 1 & & 0 \\ & 0{,}5 & \\ 0 & & 0{,}4083 \end{bmatrix}.$$

Минимальным значением обусловленности было $\operatorname{cond}(D_1^{-1}AD_2) \simeq 13{,}9427$, в то время как $\operatorname{cond}(A) \simeq 27{,}4$.

Подход к проблеме масштабирования, избранный Уилкинсоном [32], состоит в том, чтобы сделать матрицу *равновесной* перед началом решения. Грубо говоря, матрица называется равновесной, если все ее строки и столбцы имеют примерно одинаковую длину в некоторой норме. Анализ округления, приведенный в разд. 21 для гауссовского исключения, дает наиболее эффективные результаты в том случае, когда матрица равновесна, так как в этом случае небольшое возмущение одной строки (или столбца) матрицы будет того же порядка, что и любой другой строки (или столбца).

(11.23) Определение. Матрица A называется *равновесной по строкам* (по норме $\|x\|_\infty$), если для каждого номера

строки i

$$\beta^{-1} \leqslant \max_{1 \leqslant j \leqslant n} |a_{ij}| \leqslant 1,$$

где β есть основание системы с плавающей запятой.

(11.24) Определение. Матрица A называется *равновесной по столбцам (относительно нормы* $\|x\|_\infty$), если для каждого номера столбца j

$$\beta^{-1} \leqslant \max_{1 \leqslant i \leqslant n} |a_{ij}| \leqslant 1.$$

(11.25) Определение. Матрица A называется *равновесной*, если она является равновесной и по строкам, и по столбцам.

Использование β в определениях позволяет переходить к равновесной матрице, изменяя только показатель степени.

Прием Уилкинсона состоит в том, что перед решением системы уравнений матрица делается приблизительно равновесной. К сожалению, не существует единственной равновесной формы матрицы. Различные алгоритмы будут приводить к самым различным равновесным матрицам. Например, пусть $\beta = 10$ и

$$A = \begin{bmatrix} 1 & 1 & 2_{10}9 \\ 2 & -1 & {}_{10}9 \\ 1 & 2 & 0 \end{bmatrix}.$$

Если мы сначала уравновесим столбцы A, то получим матрицу

$$B = \begin{bmatrix} 0{,}1 & 0{,}1 & 0{,}2 \\ 0{,}2 & -0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0 \end{bmatrix},$$

которая является также равновесной по строкам. Но если мы сначала уравновесим строки A, то получим матрицу

$$C = \begin{bmatrix} {}_{10}-10 & {}_{10}-10 & 0{,}2 \\ 2_{10}-10 & -{}_{10}-10 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0 \end{bmatrix},$$

которая является также равновесной по столбцам. Однако матрицы B и C заметно отличаются обусловленностью, характером масштабирования и выбором главных элементов для

гауссовского исключения[1]). Матрица B является более удовлетворительно масштабированной формой матрицы A.

Хотя, несомненно, с помощью проверки всегда можно найти приемлемое масштабирование матрицы третьего порядка, однако совершенно не ясно, как писать программу масштабирования для матрицы общего вида. Известные нам алгоритмы, подобные предложенному Мак-Киманом [18], не дают универсального решения проблемы.

В приведенных в этой книге алгоритмах для решения системы линейных уравнений мы используем только масштабирование строк. То есть мы отыскиваем матрицу D_1, такую, что в матрице D_1A все строки имеют одинаковую длину в норме $\| \ \|_\infty$. Мы не претендуем на то, что это есть оптимальное решение проблемы масштабирования (она, как мы пытались показать, гораздо сложнее). В нашем описании масштабирование строк производится неявно путем запоминания соответствующих коэффициентов, а не с помощью действительного изменения матричных элементов (см. разд. 16).

---

[1]) Мы благодарны Хеммингу за то, что он привлек наше внимание к примеру такого типа.

12. МОДИФИКАЦИИ КРАУТА И ДУЛИТЛА

Метод Краута без перестановок, предназначенный специально для настольных вычислительных машин, исключает выписывание промежуточных величин, подобных $a_{44}^{(2)}$ в разд. 9. Вместо этого числа m_{ij} и u_{ij} вычисляются одно за другим в процессе непрерывного накопления суммы произведений. (В действительности это метод Дулитла. Более известное сейчас имя Краута связано, собственно, с разложением $A=LU$, в котором $u_{ii}=1$, а диагональные элементы L обычно отличны от 1. Различие, правда, невелико.)

Пусть $b_i=a_{i5}$ $(i=1,\ldots,4)$. Алгоритм Краута для системы уравнений 4-го порядка таков:

for $j:=1,\ 2,\ 3,\ 4,\ 5$ **do** $u_{1j}:=a_{1j}$;

for $i:=2,\ 3,\ 4$ **do** $m_{i1}:=a_{i1}/u_{11}$;

for $j:=2,\ 3,\ 4,\ 5$ **do** $u_{2j}:=a_{2j}-m_{21}\times u_{1j}$;

for $i:=3,\ 4$ **do** $m_{i2}:=(a_{i2}-m_{i1}\times u_{12})/u_{22}$;

for $j:=3,\ 4,\ 5$ **do** $u_{3j}:=a_{3j}-m_{31}\times u_{1j}-m_{32}\times u_{2j}$;

$m_{43}:=(a_{43}-m_{41}\times u_{13}-m_{42}\times u_{23})/u_{33}$;

for $j:=4,\ 5$ **do** $u_{4j}:=a_{4j}-m_{41}\times u_{1j}-m_{42}\times u_{2j}-m_{43}\times u_{3j}$.

Мы получим непосредственно числа u_{ij} и m_{ij} метода исключения Гаусса без каких-либо промежуточных записей. Заметим, что метод Краута по существу состоит в конструктивном вычислении матриц L и U из LU-теоремы в порядке, несколько отличном от того, который дан в нашем индуктивном доказательстве (9.2). Количество арифметических операций в методе Краута и в методе исключения Гаусса одинаково, поскольку операции выполняются те же самые, хотя и в другом порядке.

Для симметричной матрицы A арифметические операции можно сократить вдвое за счет вычисления G в разложении $A=GG^T$; см. (9.3). Перестановки нарушают симметрию кроме того случая, когда столбцы и строки переставляются одновременно. Уилкинсон [31] доказал, что если матрица симметрична и положительно определена, то перестановки можно опустить без серьезного увеличения ошибок округления, а это упрощает программирование для этих очень часто встречающихся матриц.

Предпочтительнее ли метод Краута метода исключения Гаусса? При использовании настольных вычислительных устройств очень важно избегать выписывания промежуточных

результатов, но это менее существенно для автоматических вычислительных машин. Перестановки трудно осуществить с помощью карандаша и бумаги в методе Краута и легче в методе исключения Гаусса. Однако трудности с перестановками в методе Краута не возникают, если иметь дело с автоматическими вычислительными машинами. Алгоритм Краута приводит к меньшим ошибкам округления при условии, что вычислительная машина (типа IBM 7090) может легко запомнить накапливаемую сумму произведений на регистре двойной длины и округлять ее с обычной точностью только при запоминании окончательного результата. Если для вычислительной машины (типа Берроуз B 5500, например) такое запоминание затруднительно, то использование метода Краута не дает преимуществ.

13. ИТЕРАЦИОННОЕ УТОЧНЕНИЕ

Несмотря на то что ошибки округления в методе исключения Гаусса с перестановками для хорошо масштабированной матрицы очень малы, иногда возникает необходимость получить бо́льшую точность. Например, при изменении координат с помощью преобразования подобия для вычисления собственных значений необходимо иметь уверенность в том, что S^{-1} вычислена настолько точно, насколько это возможно, когда обратная матрица вычисляется с обычной точностью. В других случаях требовательный заказчик может настаивать на вычислении $A^{-1}b$ с большей точностью, чем даже гарантируется исходными данными! Поэтому важно иметь в виду, что решение x очень высокой точности может быть получено для большинства линейных систем $Ax=b$ и что это может быть сделано при скромном (около 25%) увеличении времени, требующегося для получения первоначального решения.

В этом параграфе мы используем обозначения x_1, x_2, r_1 и т. д. для обозначения векторов, а не компонент вектора.

Ключом для улучшения точности первого решения x_1 является вычисление с двойной точностью его *невязки* $r_1 = b - Ax_1$. Зная r_1, мы затем решаем систему

$$Az_1 = r_1.$$

Если мы знаем z_1 точно, то $x_2 = x_1 + z_1$ будет точным решением системы $Ax=b$, потому что

$$Ax_2 = A(x_1 + z_1) = Ax_1 + Az_1 = Ax_1 + r_1 = b.$$

Если же мы знаем z_1 только до нескольких разрядов, то это обычно дает возможность вычислить x_2, который будет более точным, чем x_1. Вычисляя $r_2 = b - Ax_2$, мы можем затем вычислить z_2 и x_3. Векторы $x_1, x_2, \ldots$, вычисленные с простой точностью, обычно образуют последовательность, быстро сходящуюся к постоянному вектору, который с простой точностью является решением системы $Ax=b$. Заметим, что каждая система $Az_k = r_k$ имеет ту же самую матрицу A, так что мы можем использовать m_{ij}, уже хранящиеся после первого решения (см. разд. 9). Такой путь итерационного уточнения первоначального решения приводит к умеренному увеличению времени счета.

Исключительно важно, чтобы вычисление невязок r_k выполнялось с более высокой точностью, чем остальные операции. Это основной принцип для всех способов решения уравнения: вычисление невязки является критической операцией и дол-

жно быть выполнено с наибольшей точностью. Изредка необходимо достигать такой высокой точности во всех других частях алгоритма. Другой пример этого принципа: при использовании метода Ньютона для вычисления изолированного нуля вещественной функции $f(x)$ можно обычно довольно грубо вычислять производную $f'(x_i)$, но невязка $f(x_i)$ должна быть точной.

Заметим, что ошибка e_1 решения x_1 связана с r_1 простым соотношением

$$e_1 = x_1 - A^{-1}b = A^{-1}(Ax_1 - b) = -A^{-1}r_1.$$

Таким образом,

$$\|e_1\| \leqslant \|A^{-1}\| \cdot \|r_1\|. \tag{13.1}$$

Детальный анализ ошибок округления метода исключения Гаусса дан в разд. 21, а в разд. 22 дано доказательство сходимости итерационного уточнения. Цель следующего ниже изложения — дать читателю некоторое правдоподобное представление о природе алгоритма. В частности, мы хотим показать, что величина $\operatorname{cond}(A)$ приближенно определяет быстроту сходимости последовательности x_k к своему предельному вектору. Обратно, последовательность x_k может быть использована для грубой оценки $\operatorname{cond}(A)$.

Предположим для определенности, что мы работаем на t-разрядной вычислительной машине с основанием β в режиме с плавающей запятой (см. разд. 20). Тогда ошибка округления одной арифметической операции для любого числа y равна $\beta^{-t}|y|$. Полагаем

$$x_1 = (x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$$

и

$$r_1 = (r_1^{(1)}, r_2^{(1)}, \ldots, r_n^{(1)}).$$

Так как $r_1 = b - Ax_1$, то

$$r_i^{(1)} = b_i - a_{i1}x_1^{(1)} - a_{i2}x_2^{(1)} - \ldots - a_{in}x_n^{(1)}. \tag{13.2}$$

Метод исключения Гаусса определяет с обычной точностью компоненты $x_j^{(1)}$ вектора x_1, так что компоненты $r_i^{(1)}$ вектора r_1 в (13.2) малы. Большинство разрядов членов $a_{ij}x_j^{(1)}$ в i-й сумме (13.2) уничтожаются друг другом и некоторыми ведущими разрядами b_i. В результате этого i-я компонента $r_i^{(1)}$ имеет порядок наименьшего значащего разряда наибольшего (по абсолютной величине) члена $a_{ij}x_j^{(1)}$, т. е. $\beta^{-t} \cdot \max_j |a_{ij}| \cdot |x_j^{(1)}|$.

Следовательно, грубо говоря,

$$\|r_1\| \simeq \beta^{-t}\|A\|\cdot\|x_1\|.$$

Так как (13.1) является обычно приближенным равенством, то

$$\|e_1\| \simeq \|A^{-1}\|\cdot\|A\|\cdot\|x_1\|\beta^{-t} = \operatorname{cond}(A)\|x_1\|\beta^{-t}.$$

Следовательно, при $\operatorname{cond}(A) \simeq \beta^p$

$$\|e_1\| \simeq \|x_1\|\beta^{p-t}.$$

Если $p>t$, то ошибка $\|e_1\|$ будет больше, чем $\|x_1\|$ и мы ничего не потеряем, полагая $x_1=\theta$. С другой стороны, предположим, что $p \leqslant t-1$, и пусть $q=t-p$. Тогда

$$\|e_1\|/\|x_1\| \simeq \beta^{-q},$$

т. е. у x_1 приблизительно q первых разрядов верны, а остальные в пределах погрешности.

Теперь r_1 вычислено и, поскольку оно вычислено с повышенной точностью, мы можем предположить, что невязка имеет t верных разрядов, см. (20.21). Когда мы решаем систему $Az_1=r_1$, то получаем z_1, которое имеет примерно q верных разрядов. Эти разряды должны выправить q первых ошибочных разрядов в x_1 и, следовательно, сумма $x_2=x_1+z_1$ будет иметь $2q$ верных разрядов. Вообще говоря, утверждение, что в режиме с плавающей запятой сумма двух чисел, имеющих q верных разрядов, будет иметь $2q$ верных разрядов, неверно, но в этой ситуации x_1 и z_1 связаны специальным образом. С каждой итерацией точность увеличивается примерно на q разрядов, и процесс состоит из k итераций: он продолжается до тех пор, пока не выполнится условие $kq \geqslant t$. После этого дальнейшее уточнение невозможно, если сложение z_k и x_k само не будет выполняться с двойной точностью.

Так как z_1 совпадает с $-e_1$ в q разрядах, мы имеем

$$\frac{\|z_1\|}{\|x_1\|} \simeq \frac{\|e_1\|}{\|x_1\|} \simeq \beta^{-q} = \beta^{p-t}.$$

Таким образом,

$$\operatorname{cond}(A) \simeq \beta^t\|z_1\|/\|x_1\|,$$

и мы можем оценить $\operatorname{cond}(A)$ из двух первых итераций. В то же время мы можем оценить количество необходимых итераций, т. е.

$$t/\log_\beta(\|x_1\|/\|z_1\|).$$

При $p \geqslant t$ матрица настолько плохо обусловлена, что итерационное уточнение невозможно. Система в этом случае не

может быть решена без привлечения повышенной точности. Для таких систем (и только для них) весь процесс исключения должен вестись с двойной или более высокой точностью.

Чтобы сделать проведенный выше анализ более наглядным, приведем пример с $\beta=10$ и $t=5$. Пусть

$$A=\begin{bmatrix} 1{,}0303 & 0{,}99030 \\ 0{,}99030 & 0{,}95285 \end{bmatrix}, \quad b=\begin{bmatrix} 2{,}4944 \\ 2{,}3988 \end{bmatrix}.$$

При округлении до восьми разрядов $A^{-1}b=(1{,}22402691,\ 1{,}24536512)^T$. Пусть $e_0=b\simeq(2{,}49,\ 2{,}40)^T$. Предположим, что мы работаем в режиме с плавающей запятой с пятью десятичными разрядами. Процесс исключения с $m_{21}=0{,}96118$ дает матрицу

$$\left[\begin{array}{cc|c} 1{,}0303 & 0{,}99030 & 2{,}4944 \\ 0 & 0{,}00099 & 0{,}0012 \end{array}\right].$$

Мы получаем

$$x_1=(1{,}2560,\ 1{,}2121)^T, \text{ так что } e_1=(0{,}0320,\ -0{,}0333)^T$$

и

$$r_1=(0{,}00000057,\ 0{,}000033715)^T.$$

Решая систему $Az_1=r_1$, получаем

$$z_1=(-0{,}03221,\ 0{,}033502)^T.$$

Далее,

$$x_2=(1{,}2238,\ 1{,}2456)^T;\ e_2=(-0{,}000272,\ 0{,}000235)^T;$$
$$r_1=(0{,}00000118,\ 0{,}0000009)^T;$$
$$z_2=(0{,}0002285,\ -0{,}0002365)^T.$$

$$x_3=(1{,}2240,\ 1{,}2454)^T;\ e_3=(-0{,}0000272,\ 0{,}0000351)^T;$$
$$r_3=(-0{,}00000682,\ -0{,}00000659)^T;$$
$$z_3=(0{,}00002717,\ -0{,}00003515)^T.$$

Теперь z_3 настолько мало, что x_3+z_3 совпадает с x_3 в пяти первых десятичных разрядах. Обычно следует остановиться на этом, но процесс можно продолжить, если для x_k использовать большее число разрядов. Мы сделаем еще один шаг:

$x_4=(1{,}22402717,\ 1{,}24536485)^T$ (используем теперь больше разрядов); $e_4=(0{,}00000026,\ -0{,}00000027)^T$;

$$r_4=(-0{,}000000004206,\ -0{,}000000003774)^T.$$

Заметим, что погрешность e_k уменьшается примерно в 100 раз после каждой итерации. Единственное исключение состоит в

том, что $\|e_3\|/\|e_2\| \simeq {}^1/_{10}$, и это из-за того, что мы сохранили в x_3 всего четыре десятичных разряда после запятой.

В данном случае $\|A\| \simeq 2$, тогда как $\|A^{-1}\| \simeq 2000$. Следовательно, $\operatorname{cond}(A) \simeq 4000 \simeq 10^{3,6}$. Согласно нашей приближенной теории, изложенной выше, $p \simeq 3,6$ и $q \simeq 5-3,6=1,4$. Таким образом, теория предсказывает улучшение примерно 1,4 десятичных разрядов за итерацию, тогда как в действительности исправляется около 2 разрядов.

Отметим, что $10^5\|z_1\|/\|x_1\| = 10^5(0,046)/1,73 \simeq 2700$ является достаточно хорошей, но заниженной оценкой $\operatorname{cond}(A)$. Это типично. Отметим также, что невязка r_k уменьшается вначале значительно медленнее, чем ошибка e_k:

$$\|r_1\| = 0,000034,$$
$$\|r_2\| = 0,0000015,$$
$$\|r_3\| = 0,0000095.$$

Даже несмотря на то что $\|e_3\| \simeq (1/10)\|e_2\|$, мы имеем $\|r_3\| \simeq \simeq 6\|r_2\|$ (!). Такое поведение типично для плохо обусловленных систем. Это существенно связано с тем фактом, что r_2 настолько мало, насколько позволяют пять значащих разрядов x. Даже r_1 меньше половины единицы в последних значащих разрядах b. Таким образом, основная особенность итерации состоит в том, что $\|r_k\|$ остается почти постоянной, тогда как $\|e_k\|$ убывает с относительно большого значения $\|e_1\|$ до нуля, в смысле обычной точности.

Наконец, мы должны отметить, что x_1 уже настолько близко к решению, насколько позволяют данные. Если правую часть изменить на половину единицы последнего значащего десятичного разряда, то $b^* = (2,49445,\ 2,39885)$ и истинное решение будет близко к

$$x^* = A^{-1}b^* = (1,318605,\ 1,147017).$$

В окрестности решения $A^{-1}b$ множество векторов в десятичном представлении ($\beta=10$, $t=5$) образуют квадратную решетку L в плоскости x с шагом сетки 10^{-4}. Ясно, что точка x решетки, ближайшая к $A^{-1}b$, находится округлением $A^{-1}b$ до четвертой значащей цифры. Преобразование A отображает эту квадратную решетку в решетку AL из параллелограммов. Так как $\operatorname{cond}(A) \simeq 4000$, длина параллелограмма может примерно в 4000 раз превосходить его ширину. Таким образом, вполне вероятно, что точка x решетки L, ближайшая к $A^{-1}b$, будет отображена в точку Ax решетки AL, более удаленную от b, чем другие точки AL. Действительно, наши вычисления показывают, что именно это и происходит. Таким образом,

минимумы $\|Ax-b\|$ и $\|x-A^{-1}b\|$ не достигаются для одного и того же x. Хотя это и частный пример, но здесь мы сталкиваемся с явлением, характерным для линейных систем.

(13.3) Упражнение. Используя вычислительные средства (или аналитические, если возможно), найти вектор x из класса векторов в десятичном представлении с плавающей запятой с пятью значащими разрядами ($\beta=10$, $t=5$), который минимизирует $\|r_2\|=\|b-Ax_2\|$. Является ли этим вектором x_3?

(13.4) Упражнение. Пусть x_1 — приближенное решение линейной системы $Ax=b$, где $b\neq\theta$. Пусть $e_1=x_1-A^{-1}b$ и $r_1=b-Ax_1$. Определим *относительную ошибку* x_1:

$$\rho_x=\|e_1\|/\|A^{-1}b\|$$

и соответствующую *относительную невязку*

$$\rho_r=\|r_1\|/\|b\|.$$

Доказать, что

$$\frac{1}{\operatorname{cond}(A)}\leqslant\frac{\rho_x}{\rho_r}\leqslant\operatorname{cond}(A), \tag{13.5}$$

и, более того, что для произвольной матрицы A при точных вычислениях любое неравенство в (13.5) может быть превращено в равенство надлежащим выбором b и x_1.

14. ВЫЧИСЛЕНИЕ ОПРЕДЕЛИТЕЛЯ

Напомним, что из доказательства LU-теоремы (9.2) следует, что

$$\det(A) = u_{11} \cdot u_{22} \ldots u_{nn}. \tag{14.1}$$

Находить определитель матрицы A требуется довольно часто, так что полная программа решения линейной системы уравнений должна это предусмотреть. Существует одна, возможно неожиданная, программистская проблема при перемножении u_{ii}, а именно вероятность переполнения или появления машинного нуля. Для $n \geqslant 20$, например, величина определителя может быть очень маленькой, даже если элементы матрицы порядка 1. Поэтому требуется специальная подпрограмма перемножения, которая допускала бы изменение показателей степени в значительно большем диапазоне, чем при обычных арифметических операциях с плавающей запятой. В Стэнфордском университете в различных программах мы использовали специальный вид двупараметрического накапливающегося произведения. Один параметр является числом в десятичном представлении с плавающей запятой, абсолютная величина которого относительно близка к единице — например, находится в области $[10^{-10}, 10^{10}]$ или даже $[10^{-1}, 1]$. Второй целочисленный параметр есть показатель степени, связанный с произведением. Перед каждым перемножением производится проверка возможности переполнения или потери всех разрядов и при необходимости изменяется показатель степени. Окончательное произведение преобразовывается, если это возможно, к нормальному представлению числа с плавающей запятой. Если некоторый сомножитель равен нулю, можно, конечно, вычисление окончить раньше.

Другое решение проблемы переполнения заключается в том, чтобы вычислять

$$\log|\det(A)| = \sum_{i=1}^{n} \log u_{ii} \tag{14.2}$$

и делать дополнительный анализ знака и возможности обращения в нуль u_{ii}. Однако использование логарифмов может привести к некоторой потере точности из-за взаимного уничтожения слагаемых в сумме (14.2), тогда как обобщенное произведение может быть вычислено с очень маленькой ошибкой округления, что можно показать, используя результаты разд. 20. Потерю точности можно предотвратить вычислением

суммы (14.2) с двойной точностью. Если двойная точность логарифмической функции доступна, ее имеет смысл использовать, но накопление суммы (14.2) с двойной точностью, вероятно, полезно, даже если логарифмы вычислены только с обычной точностью.

Как было указано Каханом [14], хорошая вычислительная система с плавающей запятой должна следить за возможностью переполнения или потери всех разрядов и информировать основную программу об этом без прерывания и без потери ведущих разрядов. Такая система позволит избежать необходимости в специальных подпрограммах для вычисления определителей. К сожалению, нет уверенности, что такие системы будут широко доступны в ближайшем будущем.

15. ПОЧТИ ВЫРОЖДЕННЫЕ МАТРИЦЫ

Если матрица A системы линейных уравнений $Ax=b$ является вырожденной, мы теряем математическую основу для обычных методов решения. Такая система имеет решение только для некоторых правых частей b, а в тех случаях, когда решение существует, можно найти другие решения, прибавляя к первому любое решение u однородной системы $Au=\theta$.

Мы не знаем, что именно вычислительные программы должны делать с вырожденной матрицей A. Игнорируем временно вопросы округления и рассмотрим чисто математическую проблему.

Если рассматривать матрицу A как множество вектор-столбцов $c_1, \ldots, c_n$, где c_j является j-м столбцом A, решение линейной системы $Ax=b$ можно интерпретировать как нахождение n действительных чисел $x_1, \ldots, x_n$, таких, что

15 1) $$b = x_1c_1 + \ldots + x_nc_n.$$

Если матрица A вырожденна, столбцы c_j являются линейно зависимыми. С этой точки зрения, может быть, программа должна найти ранг r множества вектор-столбцов. Затем, может быть, она должна найти среди n столбцов подмножество, состоящее из r столбцов, которые в некотором смысле образуют «хорошо обусловленный» базис для пространства вектор-столбцов. Наконец, может быть, она должна установить, принадлежит ли вектор b этому пространству вектор-столбцов и, если принадлежит, найти представление b в виде линейной комбинации r базисных векторов:

$$b = x_1c_1 + \ldots + x_rc_r,$$

где мы предположили для удобства обозначений, что первые r столбцов образуют базис. Программа также может представить каждый столбец c_j, не принадлежащий базису, как линейную комбинацию столбцов базиса.

С другой стороны, матрицу A можно рассматривать как множество вектор-строк $r_1, \ldots, r_n$, где r_j является i-й строкой A. Тогда мы можем интепретировать решение системы $Ax=b$ как нахождение вектор-столбца x, такого, что

(15.2) $$r_1x = b_1, \ \ldots, \ r_nx = b_n.$$

Если матрица A вырожденна, строки $r_1, \ldots, r_n$ линейно зависимы. С точки зрения такого подхода, может быть, программа должна найти ранг r матрицы и подмножество r строк,

которые образуют «хорошо обусловленный» базис. Затем, может быть, она должна установить, существует ли вектор x, такой, что выполняется (15.2) и, если да, то вычислить его. Программа может также выразить каждую другую вектор-строку A как линейную комбинацию строк базиса.

Причина, по которой мы так много использовали выражение «может быть» в приведенном выше обсуждении, состоит в том, что, по-видимому, не существует общепринятой последовательности действий для вырожденной матрицы. Мы не знаем, что действительно нужно тем, кто пользуется программами решения линейных уравнений. Более того, не ясно, существует ли какой-нибудь обоснованный способ определения ранга данной матрицы в процессе прямого исключения. Чрезвычайно усложняет ситуацию тот факт, что резкое математическое различие между вырожденными и невырожденными матрицами существует только в математически идеальном мире вещественных чисел. Поскольку действия над матрицами производятся с округлением, это различие становится неопределенным. Таким образом, некоторая невырожденная матрица в результате округлений может стать вырожденной. Очень вероятно, что вырожденная в действительности матрица будет за счет округления превращена в близкую, но невырожденную. Это означает, что в вычислительной практике имеется, очевидно, значительное количество вырожденных матриц, которые потребуют специального рассмотрения в программе решения произвольных линейных уравнений.

В наших алгоритмах решения линейных уравнений, разд. 16 и 17, появление вырожденной или почти вырожденной матрицы A будет обнаруживаться одним из двух способов:

(15.3) на некоторой стадии исключения все элементы ведущего столбца точно обращаются в нуль

или

(15.4) итерационное уточнение, примененное в нашем алгоритме, не будет сходиться.

Если вырожденность матрицы A не очевидна, то условие (15.3) маловероятно, поскольку обычно округление делает элементы ведущего столбца отличными от нуля. Условие (15.4) может иметь и имеет место для некоторых матриц, и мы должны предусмотреть определенные рекомендации, как поступать дальше. Существующие программы предусматривают простое обращение к процедуре ошибки *SINGULAR* и предполагают, что пользователь программ предусматривает свою собственную

восстанавливающую подпрограмму, если он знает, что он хочет делать.

Важной проблемой является разработка надежной программы, которая будет преодолевать любое последствие близости к вырождению. По нашему мнению, такая программа должна использовать метод полного упорядочивания для отыскания достаточно хорошо обусловленного $(r \times r)$-минора M матрицы A, где r должно приближенно равняться рангу матрицы A. Строки минора M должны образовывать базис для строк матрицы A, а столбцы — базис для столбцов матрицы A. Мы остерегаемся сказать, как должна работать дальше программа. Возможно, следует использовать некоторые аспекты метода наименьших квадратов для несовместных уравнений, включая некоторые аспекты задачи вычисления матрицы, псевдообратной для матрицы A. См. Голуб [11], Бузингер н Голуб [6] и Голуб и Кахан [12].

Обсуждение вырожденности и почти вырожденности матрицы в связи с другими проблемами численного анализа можно найти у Форсайта [37], где показано, что почти сингулярные задачи приводят к бо́льшим вычислительным трудностям, чем истинно сингулярные задачи, и что почти сингулярные задачи лучше исследуются методами, которые в некотором смысле близки методам сингулярных задач.

16. ПРОГРАММИРОВАНИЕ НА АЛГОЛе-60

Вычислительные программы, использующие метод исключения Гаусса или один из его вариантов, были написаны на многих программных языках и использовались на разных вычислительных машинах. Часть этих программ использует также некоторые формы итерационного уточнения. Вместе с Мак-Киманом мы разрабатывали на языке АЛГОЛ-60 систему из четырех процедур, которая приводится здесь как программа (16.1). Ранние варианты этой программы содержатся у Форсайта [38] и у Мак-Кимана [18]. Введение в язык АЛГОЛ-60 и его определение см. Бауман и др. [1] и Наур и др. [22]. После программы следует несколько страниц пояснительного текста.

(16.1)

Программа на языке АЛГОЛ-60 для решения линейных систем

```
begin
  comment Блок линейных систем, вариант
                                АЛГОЛ-60;
  integer array ps[1 : 100]; comment Глобальный мас-
                                сив индексов
                                главных элемен-
                                тов. Мы предпо-
                                лагаем, что n ≤
                                ≤ 100;
  procedure DECOMPOSE(n, A, LU);
    value n; integer n;
    real array A, LU; comment A, LU[1 : n, 1 : n];
    comment Используется глобальный массив ps;
    comment Вычисляются треугольные матрицы L
            и U и матрица перестановок P, та-
            кие, что LU = PA. Запоминается L−I
            и U в LU. Массив ps содержит ин-
            дексы переставленных строк.
    comment DECOMPOSE(n, A, A) записывает LU
            на место A;
  begin
    real array scales[1 : n];
    integer i, j, k, pivotindex;
    real normrow, pivot, size, biggest, mult;
```

```
comment Присвоение начальных значений мас-
        сивам ps, LU и scales;
for i : = 1 step 1 until n do
begin
  ps[i] : = i;
  normrow : = 0;
  for j : = 1 step 1 until n do
  begin
    LU[i, j] : = A[i, j];
    if normrow < abs(LU[i, j]) then norm-
       row : = abs (LU[i, j]);
  end;
  if normrow ≠ 0 then scales [i] : = 1/normrow
    else begin scales [i] : = 0; SINGULAR(0) end;
end;
comment Метод исключения Гаусса с частич-
        ным упорядочиванием;
for k : = 1 step 1 until n − 1 do
begin
  biggest : = 0;
  for i : = k step 1 until n do
  begin
    size : = abs(LU[ps[i], k]) × scales [ps[i]];
    if biggest < size then
      begin biggest : = size; pivotindex : = i end;
  end;
  if biggest = 0 then
    begin SINGULAR (1); go to endkloop end;
  if pivotindex ≠ k then
    begin
      j : = ps[k]; ps[k] : = ps[pivotindex]; ps[pi-
      votindex] : = j
    end;
  pivot : = LU[ps[k], k];
  for i : = k + 1 step 1 until n do
  begin
    LU[ps[i], k] : = mult : = LU [ps[i], k]/pivot;
    if mult ≠ 0 then
    for j : = k + 1 step 1 until n do
      LU[ps[i], j] : = LU[ps[i], j] − mult ×
      × LU[ps[i], j];
```

```
            comment  Внутренний цикл. Меняется
                     только индекс столбца. Для
                     большей эффективности мо-
                     жет использоваться машинный
                     код.
      end
    endkloop:
    end;
    if LU[ps[n], n] = 0 then SINGULAR(1);
  end DECOMPOSE;
```

```
  procedure SOLVE(n, LU, b, x);
    value n; integer n;
    real array LU, b, x; comment LU[1 : n, 1 : n],
                                 b, x[1 : n];
    comment Использует глобальный целый мас-
            сив ps;
    comment Решает Ax = b, используя LU из
            DECOMPOSE;
  begin
    integer i, j;
    real dot;
    for i : = 1 step 1 until n do
    begin
      dot : = 0;
      for j : = 1 step 1 until i — 1 do
        dot : = dot + LU[ps[i], j] × x[j];
      x[i] : = b[ps[i]] — dot;
    end;
    for i : = n step —1 until 1 do
    begin
      dot : = 0
      for j : = i + 1 step 1 until n do
        dot : = dot + LU[ps[i], j] × x[j];
      x[i] : = (x[i] — dot)/LU[ps[i], i];
    end;
    comment Как и в DECOMPOSE, внутренние
            циклы меняют только индекс столбца
            LU и могут быть для эффективно-
            сти представлены в машинном коде.
  end SOLVE;
```

```
procedure IMPROVE (n, A, LU, b, x, digits);
  value n; integer n;
  real array A, LU, b, x; comment A, LU[1 : n, 1 : n],
                                  b, x[1 : n];
  real digits;
  comment A — исходная матрица, LU из DECOM-
            POSE, b — правая часть, x — ре-
            шение, полученное из SOLVE.
            Уточняет x до машинной точно-
            сти, а переменную digits полагает
            равной числу разрядов x, которые
            не меняются при итерации.
  comment   Величины, зависящие от типа ма-
            шины, обозначены через 0-0;
begin
  real array r, dx[1 : n];
  integer iter, itmax, i;
  real t, normdx, eps, normx;
  real procedure log (x); value x; real x;
    log : = 0,4342944819 × ln (x);
  real procedure accumdotprod(n, A, i, x, extra-
                                        term);
    value n, i, extraterm; integer n, i; real extra-
                                        term;
    real array A, x;
    comment Эта процедура должна вычислять
            скалярное произведение i-й строки
            массива A и вектора x, затем при-
            бавлять к результату extraterm.
            Умножение A[i, j] × x[j] и все сло-
            жения должны быть выполнены
            с двойной точностью. Тело про-
            цедуры не может быть написано
            на языке АЛГОЛ-60;
    comment Тело процедуры accumdotprod с ис-
            пользованием процедуры innerprod
            (см. Мартин, Питерс и Уилкинсон
            [19], стр. 206) может быть написано
            следующим образом:
    begin
      real d1, d2
      integer k
```

```
        innerprod (1, 1, n, extraterm, 0, A [i, k], x[k],
                                        k, d1, d2)
        accumdotprod : = d1
      end;
    begin
      comment ⟨в кодах⟩;
      accumdotprod : =0-0; comment 0-0 обозначает
                                    результат ма-
                                    шинного кода;
    end accumdotprod;
    eps : = 0-0; comment Зависящий от машины
                         порядок округления;
    itmax : = 0-0; comment Используется приблизительно 2×log(1/eps);
    normx : =0;
    for i : = 1 step 1 until n do
      if normx < abs (x [i]) then normx : = abs (x[i]);
    if normx = 0 then
      begin digits : = log(eps); go to converged end;
    for iter : = 1 step 1 until itmax do
    begin
      for i : = 1 step 1 until n do
        r[i] : = − accumdotprod (n, A, i, x, − b[i]);
      SOLVE (n, LU, r, dx);
      normdx : = 0;
      for i : = 1 step 1 until n do
      begin
        t : = x[i];
        x[i] : = x[i] + dx[i];
        if normdx < abs(x[i] − t) then normdx : =
                                        abs(x[i] − t);
      end;
      if iter = 1 then
        digits : = − log(if normdx ≠ 0 then
                          normdx/normx else eps);
      if normdx ≤ eps × normx then go to conver-
        ged;
    end iter;
    comment Итерации не сходятся;
    SINGULAR (2);
    converged:
  end IMPROVE;
```

```
procedure SINCULAR (why);
  value why; integer why;
  comment Печатается причина ошибки для процедур DECOMPOSE и IMPROVE;
  comment outstring Означает выдачу на печать;
begin
  if why = 0 then
    outstring („Матрица с нулевой строкой в DECOMPOSE“);
  if why = 1 then
    outstring („Вырожденная матрица в DECOMPOSE. SOLVE будет делить на нуль“);
  if why = 2 then
    outstring („Нет сходимости в IMPROVE. Матрица почти вырожденна“);
end SINGULAR;
end Блока линейных систем на языке АЛГОЛ-60
```

Замечания к программе на языке АЛГОЛ-60: процедура *DECOMPOSE*(n, A, LU) использует метод исключения для нахождения треугольных $(n \times n)$-матриц L и U, таких, что $LU = PA$, где PA отличается от матрицы A перестановкой строк. Информация о перестановке хранится в глобальном массиве *ps*, а матрицы $L - I$ и U хранятся в массиве LU.

Процедура *SOLVE*(n, LU, b, x) использует LU-разложение из процедуры *DECOMPOSE* для нахождения приближенного решения заданной системы уравнений $Ax = b$.

Процедура *IMPROVE* $(n, A, LU, b, x, digits)$ использует исходную матрицу A, ее LU-разложение, правую часть b и приближенное решение x, полученное процедурой *SOLVE*. Она выполняет процесс итерационного уточнения, если возможно, до тех пор, пока x не будет точным в пределах машинных разрядов. Она также дает оценку *digits* — точности первого приближения. Значение *digits* представляет собой, грубо говоря, число десятичных знаков, которые не меняются при итерациях. Это число является мерой обусловленности A.

Процедура *SINGULAR*(*why*) используется другими процедурами для указания наличия условий для появления ошибок.

На практике эти процедуры используются другими процедурами или управляющими программами, написанными для

решения специфического класса задач. Например, мы включили в разд. 18 процедуру, которая обращает матрицу.

Процедура *DECOMPOSE* использует метод исключения в основном в форме, описанной в разд. 9. Временно игнорируя масштабирование и выбор главного элемента, мы можем выразить основные вычисления метода исключения следующим образом:

(16.2)
$$\textbf{for } j := k+1 \textbf{ step } 1 \textbf{ until } n \textbf{ do}$$
$$a_{ij} := a_{ij} - (a_{ik}/a_{kk}) \times a_{kj}.$$

Эта операция выполняется во внутреннем цикле. Коэффициенты, т. е. множители a_{ik}/a_{kk}, сохраняются, поскольку они образуют нижнюю треугольную матрицу L.

Важность надлежащего масштабирования, или уравновешивания, была отмечена в разд. 11. Там было показано, что единственно возможным эффектом масштабирования является изменение выбора главного элемента. Поэтому в процедуре *DECOMPOSE* мы находим максимальный по модулю элемент в каждой строке матрицы и записываем его обратную величину в массив *scales*, но в действительности никакого масштабирования не производим. Вместо этого мы используем масштабные множители для выбора главного элемента и только для этого. Эта методика имеет две положительные стороны: нам нет необходимости вычислять точно степени основания вычислительной машины для масштабирования и не нужно умножать правые части на масштабные множители.

Те же самые соображения применяются при выборе главного элемента. Глобальному массиву *ps* присваивается начальное значение $ps[i]=i$. В процессе исключения наибольший элемент в столбце выбирается в качестве главного элемента, но строки (уравнения) в действительности не переставляются. Вместо этого переставляются соответствующие элементы *ps*. В этом случае мы пишем $A[ps[i], j]$ вместо $A[i, j]$. Это не приводит к большой потере времени, поскольку все «внутренние циклы» организуются для фиксированного столбца с индексом j. Мы экономим время, которое потребовалось бы для выполнения перестановок.

Схему выбора главного элемента можно интерпретировать двумя способами. Для того чтобы это проиллюстрировать, предположим, что $n=5$ и что после процедуры *DECOMPOSE* вектор $ps = (3, 5, 1, 2, 4)$.

Информация, хранящаяся в массиве LU, может быть разделена на две части, которые мы представим схематически

следующим образом:

$$\begin{bmatrix} m & m & u & u & u \\ m & m & m & u & u \\ u & u & u & u & u \\ m & m & m & m & u \\ m & u & u & u & u \end{bmatrix}.$$

Эта таблица представляет две матрицы

$$L_* = \begin{bmatrix} m & m & 1 & 0 & 0 \\ m & m & m & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ m & m & m & m & 1 \\ m & 1 & 0 & 0 & 0 \end{bmatrix} \quad \text{и} \quad U_* = \begin{bmatrix} 0 & 0 & u & u & u \\ 0 & 0 & 0 & u & u \\ u & u & u & u & u \\ 0 & 0 & 0 & 0 & u \\ 0 & u & u & u & u \end{bmatrix}.$$

При этом

$$L_* U_* = A.$$

Конечно, матрицы L_* и U_* не являются треугольными, но они имеют такую форму, что уравнения $L_* y = b$ и $U_* x = y$ могут быть легко решены.

С другой стороны, пусть P — матрица перестановок,

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

и пусть треугольные матрицы L и U отличаются от матриц L_* и U_* только перестановкой строк. Тогда

$$LU = PA.$$

Иными словами, матрицы L и U являются сомножителями некоторой матрицы, полученной из A перестановкой строк. В общем случае см. теорему (10.3).

Поскольку мы решили при выборе главного элемента вместо действительной перестановки строк обходиться схемой двойной индексации, то на многих вычислительных машинах оказалось более эффективным использовать метод исключения Гаусса, нежели метод Краута. Дело в том, что алгоритм Краута использует переменные внутренние циклы по индексам как строк, так и столбцов. В общем случае не ясно,

который из этих методов лучше. Это может зависеть от конструктивных особенностей вычислительной машины.

Если масштабный множитель или главный элемент обращается в нуль, то A должна быть вырожденна. В этом случае в процедуру *DECOMPOSE* можно включить в качестве еще одного параметра выходную метку для вырожденных матриц. Мы избегали этого, потому что такие метки часто мешают неискушенному потребителю. Вместо этого мы обращались к процедуре *SINGULAR* для вывода на печать соответствующей информации и возвращали управление в процедуру *DECOMPOSE*. Потребитель может видоизменить процедуру *SINGULAR*, если он хочет ввести в нее еще какие-либо операции. Если в процедуре *SOLVE* используется LU-разложение, как это обычно бывает, то может произойти деление на нуль. (Относительно вырожденных систем см. разд. 15.)

Процедура *DECOMPOSE* может быть использована двумя способами. При обращении к

$$DECOMPOSE(n,\ A,\ LU)$$

сомножители матрицы A заносятся в массив LU. Массив A при этом не меняется и сохраняется для использования в последующих вычислениях, например, при итерационном уточнении. Однако хранение массива A требует два $(n\times n)$-массива — A и LU. При обращении к

$$DECOMPOSE(n,\ A,\ A)$$

сомножители матриц заносятся непосредственно в массив A, тем самым уничтожая исходную матрицу. Это экономит память, если матрица A больше не нужна.

Используя LU-разложение, можно относительно просто вычислять определители. Следует только быть осторожным, чтобы избежать переполнения или потери всех разрядов, а при анализе знака определителя учитывать характер перестановок (см. разд. 14 и упражнение (16.5)).

Процедура *SOLVE* выполняет небольшой объем вычислений, поскольку почти все операции выполняются в процедуре *DECOMPOSE*. Процедура *SOLVE* состоит из двух циклов. В первом решается система уравнений с нижней треугольной матрицей $Ly=b$. Во втором, обратном, цикле решается система уравнений $Ux=y$ с верхней треугольной матрицей. Промежуточный вектор y хранится в массиве x, а правая часть b не меняется. Так как в процедуре *SOLVE* требуется информация о перестановках строк, массив *ps* описан на том же самом уровне, что и основные процедуры, и является по отношению к ним глобальным.

В процедуре *IMPROVE* делается попытка уточнить решение, полученное процедурой *SOLVE*. Невязка r вычисляется с помощью вещественной процедуры *accumdotprod*. Затем с помощью процедуры *SOLVE* находится соответствующее dx и прибавляется к x. Этот процесс продолжается примерно до тех пор, пока изменение вектора x не окажется меньше точности вычислительной машины или пока не будет достигнут верхний предел числа итераций.

Критерий для прекращения итераций, как и в данном случае, установить трудно. Мы выбрали следующий. Вычисляется норма первого решения, называемая *normx*. Норму $\max|x_i|$ использовать предпочтительнее, чем евклидову норму (см. разд. 2), потому что она быстрее вычисляется, а изменяется по существу так же, как и евклидова. Норма *normdx* поправки dx к решению x также вычисляется на каждой итерации. Когда относительное изменение x, т. е. *normdx*/*normx*, становится меньше некоторой малой величины, считается, что итерации сошлись. Малая величина *eps* является единицей округления вычислительной машины; она будет определена в (20.9). По существу, это наибольшее число, для которого равенство

$$1{,}0 + eps = 1{,}0$$

справедливо в машинной арифметике с плавающей запятой. Его величина зависит от машины и должна задаваться во всех частных случаях.

Число *itmax* — максимальное число итераций — берется равным удвоенному числу десятичных разрядов в машинном представлении числа, хотя такой выбор в общем-то является произвольным. Эта величина также зависит от типа машины. Если этот максимум достигается, то матрица A должна быть так плохо обусловлена, что первоначальное решение x и все поправки dx имеют, по-видимому, меньше «половины разряда» верных знаков.

Если решение x достаточно получить с меньшей точностью, чем машинная, то значение *eps* можно взять большим. В некоторых приложениях даже желательно сделать *eps* параметром процедуры *IMPROVE* так, чтобы можно было контролировать точность различных решений.

Параметр *digits*, который определяется величиной $-\log_{10}(normdx/normx)$ для первой поправки, указывает число верных знаков (десятичных) в решении x, полученном процедурой *SOLVE*. Если x является точным решением, то $dx=\theta$ и *digits* должно равняться бесконечности. Если x имеет ошибку порядка единицы последнего разряда, то dx имеет порядок этой ошибки и *digits* приближенно равно числу разрядов в

машинном представлении числа. С другой стороны, если A так плохо обусловлена, что dx имеет тот же порядок, что и x, то *digits* может быть близким к нулю или даже отрицательным. Ход рассуждений, набросанный в разд. 13, показывает, что обусловленность $\mathrm{cond}(A)$ можно грубо оценить так:

$$\mathrm{cond}(A) = \frac{1}{eps}\,\frac{normdx}{normx} = \frac{1}{eps}\cdot 10^{-digits}. \tag{16.3}$$

Существенно, чтобы невязки были точны; i-я компонента невязки

$$r_i = b_i - \sum_{j=1}^{n} a_{ij}x_j$$

вычисляется вещественной процедурой *accumdotprod*. Мы ожидаем, что сумма

$$\sum_{j=1}^{n} a_{ij}x_j$$

будет очень близка к величине b_i, так что их разность будет очень малой величиной. Если сумма вычисляется с обычной машинной точностью, то при последнем вычитании будут потеряны почти все разряды. Наилучшим способом вычисления невязок является использование *накапливающегося скалярного произведения*. Обычное перемножение двух чисел a_{ij} и x_j, заданных с простой точностью, дает число с удвоенным числом разрядов. Сумму этих членов, имеющих двойную точность, следует вычислять, используя сложение с удвоенным числом разрядов. Наконец, эту сумму следует вычесть из b_i, используя также вычитание с удвоенным числом разрядов. Все эти операции могут быть выполнены очень эффективно на некоторых вычислительных машинах и требуют сложного и длительного программирования на других. В любом случае вычисления невозможно полностью выразить на языке АЛГОЛ-60, поэтому в теле процедуры *accumdotprod* необходимо использовать некоторое расширение АЛГОЛа или машинные коды. Мы более подробно рассмотрим накапливающееся скалярное произведение в разд. 20, а для некоторых специфических вычислительных машин — в следующем разделе.

Как в процедуре *DECOMPOSE*, так и в процедуре *SOLVE* большинство арифметических операций выполняется во внутренних циклах по j. Если имеющийся алгольный транслятор недостаточно эффективен, желательно воспользоваться рас-

ширением языка или машинным кодом для получения наибольшей эффективности этих циклов.

Представленные здесь процедуры были выбраны из нескольких возможных вариантов. При этом мы пытались дать такие программы, которые легко могли быть поняты читателями и которые были бы достаточно эффективны для большинства вычислительных машин. Например, если для вычисления матриц L и U применить метод Краута с использованием накапливающегося скалярного произведения, то процедура *DECOMPOSE* потребует больше времени, но процедура *IMPROVE* может потребовать меньше. Общее различие во времени будет зависеть от конкретной решаемой задачи и способа вычисления скалярного произведения. Кроме того, выбор между двойной индексацией строк или действительными перестановками будет зависеть от возможностей используемого транслятора.

Ралстон [24] также рассматривал реализацию различных методов. Программы на языке АЛГОЛ приводят Боудлер, Мартин, Питерс и Уилкинсон [5].

(16.4) Упражнение. Используйте и испытайте эти процедуры на вычислительной машине, внося необходимые изменения для конкретной алгольной системы, имеющейся в вашем распоряжении. В частности, напишите тело процедуры *accumdotprod* и оцените значения *eps* и *itmax*.

(16.5) Упражнение. Напишите процедуру *DETERMINANT* (*n*, *LU*, *logdet*, *signdet*)[1]), которая использует *LU*-разложение из *DECOMPOSE* для вычисления логарифма и знака определителя матрицы (см. разд. 14). Для удобства можете модифицировать *DECOMPOSE* так, чтобы следить за изменениями знака, обусловленными перестановками, или можете получить информацию о знаке из массива *ps*.

(16.6) Упражнение. Сделайте необходимые изменения в процедуре *IMPROVE* так, чтобы итерации немедленно прекращались, если *digits* окажется меньше некоторой заранее определенной величины. Какова должна быть эта величина и как она связана с *eps* и *itmax*. Можно ли здесь использовать процедуру *SINGULAR*?

---

[1]) *logdet* означает логарифм определителя; *signdet* — знак определителя. — *Прим. ред.*

17. ПРОГРАММЫ НА ФОРТРАНе, РАСШИРЕННОМ АЛГОЛе И НА PL/I

Процедуры из предыдущего раздела почти полностью могут быть непосредственно транслированы на другие алгоритмические машинные языки. Мы это проводим здесь для принятого стандарта ФОРТРАНа, для частной реализации расширенного АЛГОЛа и для предварительно специализированного языка PL/I. Каждая программа сопровождается примечаниями о самих процедурах, о языках и используемых машинах. (Мы предлагаем читателю ознакомиться с материалами по языкам или машинам, с которыми он не знаком.)

Под ФОРТРАНом, который мы используем, мы подразумеваем язык, описанный в работе [43] и обычно называемый ASA ФОРТРАН. Он включает, насколько это возможно, общие черты трех диалектов ФОРТРАНа: ФОРТРАНа IV для IBM 7090/94, ФОРТРАНа-63 для CDC 1604 и основное программное обеспечение ФОРТРАНа для IBM System/360 (см. соответственно [47], [46], [48]). Мы избегали характерных особенностей, вроде описаний типов, условных выражений, помеченной общей памяти и размерностей массивов, которые были бы полезны, но имеют разные формы или же отсутствуют в одной или нескольких системах. Однако некоторые незначительные отличия все же имеются: описания представления с двойной точностью в IMPRUV, написание названий вещественных функций ABC, AMAX 1 и ALOG 10 в DECOMP и IMPRUV, а также вывод единичного числа в SING. С учетом возможных изменений в этих пунктах подпрограммы должны также работать в других системах ФОРТРАНа.

(17.1)

Программа на ФОРТРАНе для решения линейной системы [1])

```
      SUBROUTINE DECOMP (NN, A, UL)
      DIMENSION A(30,30), UL(30,30), SCALES(30), IPS(30)
      COMMON IPS
      N = NN
C
C     INITIALIZE IPS, UL AND SCALES
      DO 5 I = 1,N
         IPS(I) = I
         ROWNRM = 0.0
         DO 2 J = 1,N
```

[1]) Программы на языках ФОРТРАН и PL/I воспроизведены в точности по английскому оригиналу. — *Прим. ред.*

```
            UL(I,J) = A(I,J)
            IF(ROWNRM-ABS(UL(I,J))) 1,2,2
    1       ROWNRM = ABS(UL(I,J))
    2    CONTINUE
         IF (ROWNRM) 3,4,3
    3    SCALES(I) = 1.0/ROWNRM
         GO TO 5
    4    CALL SING(1)
         SCALES(I) = 0.0
    5 CONTINUE
C
C      GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
       NM1 = N-1
       DO 17 K = 1,NM1
          BIG = 0.0
          DO 11 I = K,N
             IP = IPS(I)
             SIZE = ABS(UL(IP,K))*SCALES(IP)
             IF (SIZE-BIG) 11,11,10
   10           BIG = SIZE
                IDXPIV = I
   11     CONTINUE
          IF (BIG) 13,12,13
   12        CALL SING(2)
             GO TO 17
   13     IF (IDXPIV-K) 14,15,14
   14        J = IPS(K)
             IPS(K) = IPS(IDXPIV)
             IPS(IDXPIV) = J
   15     KP = IPS(K)
          PIVOT = UL(KP,K)
          KP1 = K+1
          DO 16 I = KP1,N
             IP = IPS(I)
             EM = -UL(IP,K)/PIVOT
             UL(IP,K) = -EM
             DO 16 J = KP1,N
                UL(IP,J) = UL(IP,J) + EM*UL(KP,J)
C            INNER LOOP.  USE MACHINE LANGUAGE CODING IF COMPILER
C            DOES NOT PRODUCE EFFICIENT CODE.
   16     CONTINUE
   17 CONTINUE
       KP = IPS(N)
       IF (UL(KP,N)) 19,18,19
   18 CALL SING(2)
   19 RETURN
       END
```

```
       SUBROUTINE SOLVE (NN, UL, B, X)
       DIMENSION UL(30,30), B(30), X(30), IPS(30)
       COMMON IPS
       N = NN
       NP1 = N+1
C
       IP = IPS(1)
       X(1) = B(IP)
       DO 2 I = 2,N
          IP = IPS(I)
          IM1 = I-1
          SUM = 0.0
          DO 1 J = 1,IM1
    1        SUM = SUM + UL(IP,J)*X(J)
    2  X(I) = B(IP) - SUM
```

```
C
      IP = IPS(N)
      X(N) = X(N)/UL(IP,N)
      DO 4 IBACK = 2,N
      I = NP1-IBACK
C        I GOES (N-1),...,1
         IP = IPS(I)
         IP1 = I+1
         SUM = 0.0
         DO 3 J = IP1,N
    3       SUM = SUM + UL(IP,J)*X(J)
    4 X(I) = (X(I)-SUM)/UL(IP,I)
      RETURN
      END
```

```
      SUBROUTINE IMPRUV (NN, A, UL, B, X, DIGITS)
      DIMENSION A(30,30), UL(30,30), B(30), X(30), R(30), DX(30)
C     USES ABS(), AMAX1(), ALOG10()
      DOUBLE PRECISION SUM
      N = NN
C
      EPS = 1.0E-8
      ITMAX = 16
C     ***  EPS AND ITMAX ARE MACHINE DEPENDENT. ***
C
      XNORM = 0.0
      DO 1 I = 1,N
    1    XNORM = AMAX1(XNORM,ABS(X(I)))
      IF (XNORM) 3,2,3
    2    DIGITS = -ALOG10(EPS)
         GO TO 10
C
    3 DO 9 ITER = 1,ITMAX
         DO 5 I = 1,N
            SUM = 0.0
            DO 4 J = 1,N
    4          SUM = SUM + A(I,J)*X(J)
            SUM = B(I) - SUM
    5    R(I) = SUM
C        *** IT IS ESSENTIAL THAT A(I,J)*X(J) YIELD A DOUBLE PRECISION
C            RESULT AND THAT THE ABOVE + AND - BE DOUBLE PRECISION.  ***
         CALL SOLVE (N,UL,R,DX)
         DXNORM = 0.0
         DO 6 I = 1,N
            X = X(I)
            X(I) = X(I) + DX(I)
            DXNORM = AMAX1(DXNORM,ABS(X(I)-X))
    6    CONTINUE
         IF (ITER-1) 8,7,8
    7       DIGITS = -ALOG10(AMAX1(DXNORM/XNORM,EPS))
    8    IF (DXNORM-EPS*XNORM) 10,10,9
    9 CONTINUE
C     ITERATION DID NOT CONVERGE
      CALL SING(3)
   10 RETURN
      END
```

```
      SUBROUTINE SING (IWHY)
   11 FORMAT(54HOMATRIX WITH ZERO ROW IN DECOMPOSE.                    )
   12 FORMAT(54HOSINGULAR MATRIX IN DECOMPOSE.  ZERO DIVIDE IN SOLVE.  )
   13 FORMAT(54HONO CONVERGENCE IN IMPRUV. MATRIX IS NEARLY SINGULAR.  )
      NOUT = 3
C        NOUT = STANDARD OUTPUT UNIT
      GO TO (1,2,3),IWHY
    1 WRITE (NOUT,11)
      GO TO 10
    2 WRITE (NOUT,12)
      GO TO 10
    3 WRITE (NOUT,13)
   10 RETURN
      END
```

Заметим, что *LU* из алгольной программы в нашей программе на ФОРТРАНе выражается как *UL*.

Возможно, что потребуется несколько выражений на ФОРТРАНе для представления простого алгольного выражения, особенно при наличии индексов и логических или булевых конструкций. Но транслятор может в некотором смысле компенсировать этот недостаток более эффективным использованием машинных кодов. Это, в частности, наблюдается в нашем случае. На ФОРТРАНе внутренний цикл из DECOMP имеет вид

```
      DO 16 J = KP 1, N
(17.2)
   16 UL(IP, J) = UL(IP, J) + EM*UL(KP, J)
```

На IBM 7090 получаемые машинные коды состоят только из семи команд: выборка, умножение, сложение, запоминание и три индексные команды. Все операции, которые не зависят от J, выполняются вне цикла. Внутренние циклы в SOLVE формируются аналогично.

Во всех трех подпрограммах трансляторы машин IBM 7090 и CDC 1604 могли формировать почти совершенные машинные коды. Это обязано частично детализации, с которой необходимо описывать алгоритмы на ФОРТРАНе, а частично довольно ухищренным приемам, которые использовали эти трансляторы при формировании кодов. Можно спросить, оправдывается ли то большое время, которое затрачивается программистом и транслятором. В случае подпрограмм (таких, как эти), которые используются много раз без изменения, мы считаем, что это так.

Специальная процедура *accumdotprod* не нужна в описании процедуры, аналогичной *IMPROVE*, на ФОРТРАНе. Накапливающееся скалярное произведение получается просто из

того, что SUM описывается как переменная с двойной точностью. Оператор

```
(17.3)   DOUBLE PRECISION SUM
         SUM = SUM + A(I, J)*X(J)
```

получит произведение с удвоенной длиной слова и прибавит его с двойной точностью к SUM. Однако это выполняется различными способами на разных машинах.

На машинах IBM 7090 и IBM 7094 команда умножения в режиме с плавающей запятой дает результат удвоенной длины на двух регистрах (сумматор и регистр умножения и деления). Операция сложения с двойной точностью прибавляет такое число к другому числу удвоенной длины, хранящемуся в двух ячейках. Таким образом, оператор (17.3) производит и умножение, и сложение, и запоминание с двойной точностью. Это именно то, что мы подразумеваем под накапливающимся скалярным произведением. (Операция сложения с двойной точностью на машине IBM 7094 представляет сложную команду, а на IBM 7090 выполняется с помощью четырех простых сложений.)

Машина CDC 1604 автоматически производит округление после каждой операции с плавающей запятой. Это хорошо во многих отношениях, кроме одного. Для того чтобы получить произведение с удвоенным числом знаков и выполнить сложение с двойной точностью, необходимо использовать специальные подпрограммы, которые «распаковывают» числа с плавающей запятой и выполняют операции в режиме с фиксированной запятой. Это означает, что итерационное уточнение оказывается довольно трудоемким по сравнению с треугольным разложением. Однако ограничиться вычислениями с двойной точностью только для относительно небольшого количества арифметических операций, необходимых в итерационном уточнении, конечно, гораздо лучше, чем проводить полный расчет с удвоенным числом разрядов, чтобы получить, возможно, ту же окончательную точность.

Наш опыт с IBM System/360 хотя и не полон, но обнаруживает еще и другие стороны. Машина 360 имеет числа с плавающей запятой двух длин — «короткую форму» и «длинную форму», точность которой более чем вдвое превышает точность короткой формы. Результат умножения чисел с плавающей запятой в короткой форме является числом в длинной форме. Если затем следует сложение в длинной форме, то мы получаем накапливающееся скалярное произведение. Однако первоначальный вариант транслятора, который мы использовали, перед выполнением сложения урезал произведение до

короткой формы. В этом случае необходимо было заменить (17.3) на

```
DOUBLE PRECISION SUM, AIJ, XJ
AIJ = A(I, J)
XJ = X(J)
SUM = SUM + AIJ*XJ
```

Последующие модификации транслятора могут дать улучшение для этого случая.

Мы входим здесь в детали по нескольким причинам. Они иллюстрируют, как особенности различных машин и транслирующих систем могут воздействовать на точность и эффективность конкретных алгоритмов. Они также указывают на необходимость стандартизации — участок программы (17.3) пригоден для всех трех трансляторов, хотя в каждом случае дает различные результаты. Наконец, к сожалению, каждый, кто захочет использовать эти программы на какой-либо другой машине, встретится с теми же или с аналогичными проблемами.

Что касается других программ решения линейных систем на языке ФОРТРАН IV, то мы можем рекомендовать прекрасные программы Кахана [15]. Оии полностью используют модификации из университета Торонто для ФОРТРАНа, позволяющие избежать переполнения или появления машинного нуля.

Мы использовали также вариант АЛГОЛа, а именно расширенный АЛГОЛ для Берроуз B 5500 (см. [45]). Мы не воспроизводим полностью программы, упомянутые здесь, так как они почти идентичны программам на языке АЛГОЛ-60 из разд. 16. (Левая стрелка ← ставится вместо оператора присвоения :=; должны быть описаны нижние границы индексов для массивов параметров; также должны быть описаны метки. Описание процедуры *SINGULAR* должно быть помещено впереди описания *DECOMPOSE*. Нижние границы всех массивов в описании следует положить равными нулю, так как это способствует получению более быстродействующей программы.) Однако мы хотим описать два полезных расширения АЛГОЛа, допускаемых в системе Берроуз: переменная «массив-строка» и «двойной» оператор.

Переменная «массив-строка» используется для повышения эффективности почти во всех процедурах, кроме *SINGULAR*. При фиксированном индексе i и переменном j совокупность $[a_{ij}]$ аналогична одномерному массиву. Эта одномерная строка обозначается как A[I, *] и может использоваться в качестве действительного параметра в любой процедуре, в которой со-

ответствующий формальный параметр описан как одномерный массив. При выполнении этой процедуры такая индексация оказывается очень эффективной. Например, оператор процедуры *DECOMPOSE* в АЛГОЛе

```
if mult ≠ 0 then
for j := k + 1 step 1 until n do
    LU[ps[i], j] := LU[ps[i], j] − mult × LU[ps[k], j]
```

заменится оператором

```
IF MULT ≠ 0 THEN
   ELIM(K + 1, N, MULT, LU[PS[I], *] LU[PS[K], *])
```

в варианте расширенного АЛГОЛа для Берроуз. Описание процедуры ELIM следующее:

```
PROCEDURE ELIM (J1, J2, MULT, AI, AK);
   VALUE J1, J2, MULT; INTEGER J1, J2; REAL MULT;
   REAL ARRAY AI, AK[0];
BEGIN INTEGER J;
   FOR J ← J1 STEP 1 UNTIL J2 DO
      AI[J] ← AI[J] − MULT × AK[J];
END ELIM;
```

Аналогичные изменения проведены для операции цикла по j в процедуре *SOLVE*. Обе процедуры работают и без этих изменений; изменения только увеличивают скорость.

Двойные операторы требуются при вычислении вектора невязки. Двойной оператор состоит из последовательности операторов двойной точности и аргументов в так называемой обратной польской нотации. Мы не собираемся входить здесь в более детальные объяснения. Достаточно сказать, что оператор

```
DOUBLE(X[J], 0, Y[J], 0, ×, SUMHI, SUMLO,
                                  +, ←, SUMHI, SUMLO)
```

в варианте с двойной точностью переходит в

```
SUM ← SUM + X[J] × Y[J].
```

Этот оператор, как и массив-строка, используется при вычислении невязок. Приводим соответствующую часть программы:

```
FOR I ← 1 STEP 1 UNTIL N DO
   R[I] ← − DOUBLEDOTPROD (1, N, A[I, *], X, −B[I]);
```

и

```
REAL PROCEDURE DOUBLEDOTPROD (J1, J2, X, Y, C);
   VALUE J1, J2, C; INTEGER J1, J2; REAL C,
                                   REAL ARRAY X, Y[0];
BEGIN INTEGER J; REAL SUMHI, SUMLO;
   SUMHI ← SUMLO ← 0;
   FOR J ← J1 STEP 1 UNTIL J2 DO
      DOUBLE (X[J], 0, Y[J], 0, ×, SUMHI, SUMLO,
                                   +, ←, SUMHI, SUMLO);
   DOUBLE (SUMHI, SUMLO, C, 0, +, ←, SUMHI, SUMLO);
   DOUBLEDOTPROD ← SUMHI;
END DOUBLEDOTPROD.
```

Заметим, что операцию умножения с двойной точностью необходимо выполнять так же хорошо, как и операцию сложения с двойной точностью.

К моменту написания книги еще не были ясны некоторые черты языка PL/I. Однако, так как PL/I может стать важным машинным языком, мы включаем в этот раздел варианты нашей программы на языке PL/I. Эти программы идут успешно, но для согласования языковых особенностей, возможно, потребуются изменения. Программы даются в терминах языка, определенного в работе [49].

(17.4) Упражнение. Проделайте упражнения (16.4), (16.5) и (16.6), используя доступные вам язык и вычислительную машину.

(17.5) Программы PL/I для решения линейной системы

```
DECOMPOSE: PROCEDURE (N, A, LU) ;
  DECLARE A(*,*)  /* MATRIX TO BE DECOMPOSED */,
          LU(*,*) FLOAT  /* RESULTS */ ;
  DECLARE PS(100) FIXED BINARY EXTERNAL  /* PIVOT VECTOR, N<=100 */,
          SCALES (N), (MULT, NORMROW, BIGGEST, SIZE, PIVOT) FLOAT,
          (I, J, K, PIVIDX) FIXED BINARY ;

  /* INITIALIZE PS, LU AND SCALES */
  DO I = 1 TO N ;
     PS(I) = I ;
     NORMROW = 0 ,
     DO J = 1 TO N ;
        LU(I,J) = A(I,J) ;
        NORMROW = MAX(NORMROW, ABS(LU(I,J))) ;
     END ;
     IF NORMROW  = 0 THEN SCALES(I) = 1.0/NORMROW ;
     ELSE DO;  SCALES(I) = 0;  CALL SINGULAR('ROW');   END ;
```

```
END ;
/* GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING */
DO K = 1 TO N-1 ;
   BIGGEST = 0 ;
   DO I = K TO N ;
      SIZE = ABS(LU(PS(I),K))*SCALES(PS(I)) ;
      IF SIZE > BIGGEST THEN
         DO;  BIGGEST = SIZE;  PIVIDX = I;  END ;
   END ;
   IF BIGGEST = 0 THEN
      DO;  CALL SINGULAR('PIV');  GO TO ENDKLOOP;  END ;
   IF PIVIDX  = K THEN
      DO;  J = PS(K);  PS(K) = PS(PIVIDX);  PS(PIVIDX) = J;  END ;
   PIVOT = LU(PS(K),K) ;
   DO I = K+1 TO N ;
      LU(PS(I),K) = MULT = LU(PS(I),K)/PIVOT ;
      IF MULT  = 0 THEN
         DO J = K+1 TO N ;
            LU(PS(I),J) = LU(PS(I),J) - MULT*LU(PS(K),J) ;
            /* INNER LOOP.  ROW SUBSCRIPTS DON'T VARY */
         END ;
   END ;
 ENDKLOOP:
END ;
IF LU(PS(N),N) = 0 THEN CALL SINGULAR('PIV') ;
END DECOMPOSE ;
```

```
SOLVE: PROCEDURE (N, LU, B, X) ;
  DECLARE LU(*,*) FLOAT  /* DECOMPOSITION OF A */ ,
          B(*)  /* RIGHT HAND SIDE */ ,
          X(*)  /* SOLUTION */ ;
  DECLARE PS(100) FIXED BINARY EXTERNAL  /* PIVOT VECTOR, N<=100 */,
          DOT FLOAT, (I, J) FIXED BINARY ;

  DO I = 1 TO N ;
     DOT = 0 ;
     DO J = 1 TO I-1 ;
        DOT = DOT + LU(PS(I),J)*X(J) ;
     END ;
     X(I) = B(PS(I)) - DOT ;
  END ;

  DO I = N TO 1 BY -1 ;
     DOT = 0 ;
     DO J = I+1 TO N ;
        DOT = DOT + LU(PS(I),J)*X(J) ;
     END ;
     X(I) = (X(I) - DOT)/LU(PS(I),I) ;
  END ;
  END SOLVE ;
```

```
IMPROVE: PROCEDURE (N, A, LU, B, X, DIGITS) ;
  DECLARE  A(*,*)  /* ORIGINAL MATRIX */ ,
           LU(*,*) FLOAT  /* DECOMPOSITION OF A */ ,
           B(*)  /* RIGHT HAND SIDE */ ,
           X(*)  /* APPROXIMATE SOLUTION TO BE IMPROVED */ ,
           DIGITS  /* WILL BE SET TO ACCURACY OF INPUT X */ ;

  DECLARE  (R,DX) (N), (NORMX, NORMDX, T) FLOAT,
           (I, J, ITER) FIXED BINARY,
           EPS INITIAL (1.E-6)  /* MACHINE DEPENDENT ROUNDOFF LEVEL */,
           ITMAX INITIAL (12)  /* USE 2*LOG10(1/EPS) APPROXIMATELY */ ;

  DECLARE DPSUM FLOAT (12)
  /* IT IS ESSENTIAL THAT PRECISION OF DPSUM AND ARGUMENT OF MULTIPLY
     USED BELOW BE TWICE DEFAULT PRECISION.  DEFAULT PRECISION OF 6
     ASSUMED HERE. */ ;

  NORMX = 0 ;
  DO I = 1 TO N ;
     NORMX = MAX(NORMX,ABS(X(I))) ;
  END ;
  IF NORMX = 0 THEN
     DO;  DIGITS = -LOG10(EPS);  GO TO CONVERGED;  END ;

  DO ITER = 1 TO ITMAX ;
     DO I = 1 TO N ;
        DPSUM = 0 ;
        DO J = 1 TO N ;
           DPSUM = DPSUM + MULTIPLY(A(I,J), X(J), 12) ;
        END ;
        DPSUM = B(I) - DPSUM ;
        R(I) = DPSUM ;
     END ;
     CALL SOLVE(N,LU,R,DX) ;
     NORMDX = 0 ;
     DO I = 1 TO N ;
        T = X(I) ;
        X(I) = X(I) + DX(I) ;
        NORMDX = MAX(NORMDX,ABS(X(I)-T)) ,
     END ;
     IF ITER = 1 THEN DIGITS = -LOG10(MAX(NORMDX/NORMX,EPS)) ;
     IF NORMDX <= EPS*NORMX THEN GO TO CONVERGED ;
  END ;
  CALL SINGULAR('CON') ;
  CONVERGED;
  END IMPROVE ;
```

```
SINGULAR: PROCEDURE (WHY) ;
  DECLARE WHY CHARACTER(3) ;
  IF WHY='ROW' THEN PUT SKIP(2) LIST
     ('ZERO ROW IN DECOMPOSE') ;
  IF WHY='PIV' THEN PUT SKIP(2) LIST
     ('SINGULAR MATRIX IN DECOMPOSE. SOLVE WILL DIVIDE BY ZERO.') ;
  IF WHY='CON' THEN PUT SKIP(2) LIST
     ('NO CONVERGENCE IN IMPROVE. MATRIX IS NEARLY SINGULAR.') ;
  END SINGULAR ,
```

18. ОБРАЩЕНИЕ МАТРИЦ

Один из способов решения системы линейных уравнений $Ax=b$ состоит в вычислении матрицы, обратной к A, и последующем умножении A^{-1} на b. Этот метод может оказаться особенно выгодным, если заданы несколько правых частей b_k, так как обратную матрицу необходимо вычислить только один раз. Однако комплекс процедур решения линейных уравнений, таких, как *DECOMPOSE* и *SOLVE*, может выполнить эту задачу с меньшим числом операций и с большей точностью. Если матрицы L и U вычислены, то решение системы $LUx=b$ требует n^2-n умножений и n делений, т. е. всего n^2 мультипликативных операций (см. упражнение (9.5)). Если же A^{-1} вычислена, то определение $A^{-1}b$ требует также n^2 мультипликативных операций. С этой точки зрения оба эти метода требуют приблизительно одинакового числа операций. Однако предшествующие вычисления LU и A^{-1} требуют соответственно $^1/_3n^3$ и n^3 мультипликативных операций. Таким образом, лучше использовать LU, нежели A^{-1}, даже если имеется много различных правых частей. Более того, из-за меньшего числа операций и, следовательно, меньших ошибок округления можно ожидать, что в большинстве случаев использование LU даст более точный результат.

Однако имеются приложения, в которых необходимость вычисления A^{-1} не связана с решением систем линейных уравнений. В этих случаях обращение матрицы можно провести с помощью процедур для решения линейных уравнений. Столбцы A^{-1} являются решениями различных линейных систем

$$Ax_1=e_1,\ Ax_2=e_2,\ \ldots,\ Ax_n=e_n.$$

Здесь e_i есть i-й единичный вектор, у которого i-я координата есть единица, а остальные нули. Следовательно, для обращения матрицы достаточно решить n раз линейные системы с правыми частями $e_1, e_2, \ldots, e_n$. При этом в случае надобности может быть выполнено также итерационное уточнение для каждого столбца. Программа (18.1) иллюстрирует, как это легко сделать с помощью процедур *DECOMPOSE*, *SOLVE* и *IMPROVE*.

Иногда обращение матрицы требуется выполнить, не запоминая одновременно много больше, чем n^2 элементов. В этих случаях от матрицы A можно перейти к A^{-1} с использованием только n^2 ячеек, отведенных под A. Идеи метода таковы. Единичную матрицу хранить вообще не требуется. При гауссовском исключении элементы A затираются параллельно с появлением новых членов в правой части. Таким образом, новые

элементы преобразуемой единичной матрицы можно хранить на месте затертых элементов A. Это часто делается в компактном методе, который довольно труден для понимания. Однако при этом страдают многие процедуры из-за невозможности поиска главных элементов. Более того, так как матрица A уничтожается во время обращения, невозможно провести итерационное уточнение.

(18.1) Процедура обращения матрицы на языке АЛГОЛ-60

```
procedure INVERT (n, A, AINV);
value n; integer n; real array A, AINV;
comment Пример использования DECOMPOSE,
        SOLVE и IMPROVE для обращения ма-
        трицы;
begin
  real array LU[1 : n, 1 : n], b, x[1 : n];
  integer i, j; real digits;
  DECOMPOSE (n, A, LU);
  for j : = 1 step 1 until n do
  begin
    for i : = 1 step 1 until n do
      b[i] : = if i = j then 1 else 0;
  SOLVE (n, LU, b, x);
  IMPROVE (n, A, LU, b, x, digits);
      comment digits пренебрегается
    for i : = 1 step 1 until n do AINV [i, j] : = x[i];
  end;
end INVERT;
```

Если специально хранить A для итерационного уточнения, то теряется главное преимущество компактного метода — экономия памяти.

При обращении матрицы с выбором главных элементов или без него требуется примерно n^3 умножений для получения первого приближения к обратной матрице. Эта матрица, например X_0, может быть уточнена с помощью приема, аналогичного рассмотренному для линейных систем. Пусть $R_k = I - AX_k$, $k=0, 1, 2, \ldots$, будет матрицей невязки для A^{-1}. Тогда, если X_k есть хорошее приближение для A^{-1}, мы предполагаем, что X_kR_k близка к $A^{-1}R_k = A^{-1} - X_k$. Последнее выра-

жение — это то, которое надо прибавить к X_k для получения A^{-1}. Следовательно, мы можем использовать $X_k+X_kR_k$ в качестве следующего приближения к A^{-1}, которое обозначим X_{k+1}.

В итоге итерационный процесс

$$X_{k+1} := X_k + X_k(I - AX_k)$$

должен давать быстрое уточнение обратной матрицы. Для доказательства (без учета округления) заметим, что

$$I - AX_{k+1} = (I - AX_k)^2.$$

Следовательно, если

$$\| I - AX_0 \| = \varepsilon < 1,$$

то мы видим, что

$$\| I - AX_1 \| \leqslant \varepsilon^2$$

и что

$$\| I - AX_k \| \leqslant \varepsilon^{2^k}.$$

Так как выражение стремится к нулю как ε^{2^k} при $k \to \infty$, то при точных вычислениях итерационный процесс быстро сходится. Он часто называется ньютоновским процессом, так как его скалярный аналог

$$x_{k+1} := x_k + x_k(1 - ax_k)$$

есть в точности ньютоновский процесс для решения уравнения

$$f(x) = a - \frac{1}{x} = 0.$$

Ньютоновский матричный итерационный процесс часто записывается в форме

$$X_{k+1} := X_k(2I - AX_k).$$

Из обсуждения итерационного уточнения в разд. 13 читатель может вынести сомнение в том, что итерационный процесс в рассмотренной форме при простой точности может привести к точной обратной матрице, если матрица A плохо обусловлена. Аналогично итерационному уточнению, этот ньютоновский матричный итерационный процесс дает высокую точность только в том случае, когда невязка $R_k = I - AX_k$ вычисляется с двойной точностью, а поправка $X_k(I - AX_k)$ вычисляется отдельно и затем прибавляется к X_k.

Следует заметить, что аналогия с ньютоновским процессом не применима к итерационному уточнению для отдельной си-

стемы линейных уравнений, так как в последнем случае мы имеем

$$x_{k+1} := x_k + (LU)^{-1}(b - Ax_k).$$

В отличие от ньютоновского процесса, здесь одно и то же приближение обратной матрицы используется во всех итерациях. Как мы позже увидим, в этом случае ошибка ведет себя как ε^k, а не как ε^{2^k}.

Если читателю не требуется явного выражения для элементов A^{-1}, мы настоятельно рекомендуем их не вычислять. Как мы уже говорили, почти все, что можно сделать с помощью матрицы A^{-1}, может быть выполнено и без нее.

19. ПРИМЕР: МАТРИЦЫ ГИЛЬБЕРТА

В матричных вычислениях в качестве примеров часто используются — и иногда неправильно — матрицы Гильберта (Гильберт [10]). Приведем один из случаев, в которых они появляются.

Пусть на интервале $0 \leqslant x \leqslant 1$ дана непрерывная функция $f(x)$ и нам требуется аппроксимировать $f(x)$ полиномом от x степени $n-1$. Мы записываем полином в форме

$$\sum_{i=1}^{n} c_i x^{i-1}$$

и определяем ошибку аппроксимации как

$$E = \int_0^1 \left[\sum_{i=1}^{n} c_i x^{i-1} - f(x) \right]^2 dx.$$

Коэффициенты c_i должны быть определены из условия минимальности E. Так как ошибка является дифференцируемой функцией неизвестных c_i, то минимум достигается при

$$\frac{\partial E}{\partial c_i} = 0 \qquad (i = 1, \ldots, n).$$

Расписывая эти производные, приходим к условиям

$$\frac{\partial E}{\partial c_i} = 2 \int_0^1 \left[\sum_{j=1}^{n} c_j x^{j-1} - f(x) \right] x^{i-1}\, dx = 0 \qquad (i = 1, \ldots, n).$$

Меняя порядок суммирования и интегрирования, мы получаем

$$\sum_{j=1}^{n} \left(\int_0^1 x^{i+j-2}\, dx \right) c_j = \int_0^1 f(x)\, x^{i-1}\, dx \qquad (i = 1, \ldots, n). \tag{19.1}$$

Этим n уравнениям удовлетворяют n неизвестных c_i. Если мы положим

$$h_{ij} = \int_0^1 x^{i+j-2}\, dx = \frac{1}{i+j-1}$$

и

$$b_i = \int_0^1 f(x)\, x^{i-1}\, dx \qquad (i = 1, \ldots, n),$$

то уравнения (19.1) могут быть записаны в виде

$$\sum_{j=1}^{n} h_{ij}c_j = b_i \quad (i=1,\ 2,\ \ldots,\ n).$$

Таким образом, столбец из коэффициентов $c=(c_1, \ldots, c_n)^T$ можно найти, решая систему из n уравнений с n неизвестными:

$$H_n c = b,$$

где матрица H_n имеет элементы

$$h_{ij} = \frac{1}{i+j-1} \quad (i,\ j=1,\ \ldots,\ n), \tag{19.2}$$

а вектор $b=(b_1, \ldots, b_n)^T$ определяется по данной функции $f(x)$.

Матрица H_n, определенная формулой (19.2), называется *$(n\times n)$-матрицей Гильберта*. Мы обозначим через T_n обратную к ней матрицу,

$$T_n = H_n^{-1}. \tag{19.3}$$

Матрицы Гильберта интересуют нас главным образом потому, что они очень плохо обусловлены даже для небольших значений n и числа обусловленности этих матриц, как видно из табл. (19.4), быстро растут при увеличении n.

(19.4) Некоторая информация о матрицах Гильберта

n	$\|H_n\|$	$\|T_n\|$	cond (H_n)	наибольший элемент T_n
2	1,27	$1{,}52_{10}$ 1	$1{,}93_{10}$ 1	$1{,}20_{10}$ 1
3	1,41	$3{,}72_{10}$ 2	$5{,}24_{10}$ 2	$1{,}92_{10}$ 2
4	1,50	$1{,}03_{10}$ 4	$1{,}55_{10}$ 4	$6{,}48_{10}$ 3
5	1,57	$3{,}04_{10}$ 5	$4{,}77_{10}$ 5	$1{,}79_{10}$ 5
6	1,62	$9{,}24_{10}$ 6	$1{,}50_{10}$ 7	$4{,}41_{10}$ 6
7	1,66	$2{,}86_{10}$ 8	$4{,}75_{10}$ 8	$1{,}33_{10}$ 8
8	1,70	$9{,}00_{10}$ 9	$1{,}53_{10}$10	$4{,}25_{10}$ 9
9	1,73	$2{,}86_{10}$11	$4{,}93_{10}$11	$1{,}22_{10}$11
10	1,75	$9{,}15_{10}$12	$1{,}60_{10}$13	$3{,}48_{10}$12

Из-за такой плохой обусловленности матрицы Гильберта часто используются в качестве примеров и тестов для вычислительных матричных процедур. Но, опять же по причине

плохой обусловленности, при этом необходимо соблюдать осторожность. Рассмотрим характерный пример.

В таблицах с (19.5) по (19.10) мы имеем шесть квадратных матриц шестого порядка. Все они вычислены на машине IBM 7090, и для нашего случая мы можем предполагать, что ее относительная точность равна 10^{-8}. (То есть значение *eps* в процедуре *IMPROVE* равно приблизительно 10^{-8}.)

(19.5) Таблица матрицы T, обратной к (6×6)-матрице Гильберта H

36,00	—630,00	3360,00	—7560,00	7560,00	—2772,00
—630,00	14700,00	—88200,00	211680,00	—220500,00	83160,00
3360,00	—88200,00	564480,00	—1411200,00	1512000,00	—582120,00
—7560,00	211680,00	—1411200,00	3628800,00	—3969000,00	1552320,00
7560,00	—220500,00	1512000,00	—3969000,00	4410000,00	—1746360,00
—2772,00	83160,00	—582120,00	1552320,00	—1746360,00	698544,00

(19.6) Таблица матрицы, обратной к T, вычисленной без итерационного уточнения

1,00018226	0,50015343	0,33346176	0,25010935	0,20009571	0,16675133
0,50019167	0,33349262	0,25013416	0,20011510	0,16676722	0,14294624
0,33351852	0,25015344	0,20012929	0,16677780	0,14295436	0,12508627
0,25017456	0,20014443	0,16678839	0,14296186	0,12509169	0,11119255
0,20016330	0,16680165	0,14297093	0,12509795	0,11119693	0,10007627
0,16681924	0,14298318	0,12510627	0,11120262	0,10008021	0,09098041

Первая матрица, (19.5), есть $T_6 = H_6^{-1}$. Она вычислена с помощью процедуры, приведенной в (19.11), и является точной, т. е. не содержит ошибок округления. Все ее элементы — целые числа. Мы обозначили ее просто через T, опустив индекс «6».

Вторая матрица, (19.6), получена обращением T без итерационного уточнения, т. е. с использованием только процедур *DECOMPOSE* и *SOLVE*. Любой другой обычный способ исключения даст при такой точности приблизительно ту же матрицу. При отсутствии ошибок округления мы должны были бы получить матрицу H. Однако вычисленные действительные значения благодаря округлению совпадают с приведенными в таблице только до третьего или четвертого знака.

Третья матрица, (19.7), получена с помощью итерационного уточнения матрицы (19.6). Итерационное уточнение дает сходимость к матрице, в которой многие элементы вообще не имеют ошибки! Максимальная относительная ошибка равна

(19.7) Таблица матрицы, обратной к T, вычисленной с использованием итерационного уточнения

1,00000000	0,49999999	0,33333333	0,24999999	0,20000000	0,16666666
0,50000000	0,33333333	0,25000000	0,20000000	0,16666666	0,14285714
0,33333333	0,24999999	0,20000000	0,16666666	0,14285714	0,12500000
0,25000000	0,20000000	0,16666666	0,14285714	0,12500000	0,11111110
0,20000000	0,16666666	0,14285714	0,12499999	0,11111110	0,09999999
0,16666666	0,14285714	0,12500000	0,11111110	0,09999999	0,09090909

(19,8) Таблица матрицы H_* — машинного представления матрицы H

1,00000000	0,50000000	0,33333333	0,25000000	0,20000000	0,16666666
0,50000000	0,33333333	0,25000000	0,20000000	0,16666666	0,14285714
0,33333333	0,25000000	0,20000000	0,16666666	0,14285714	0,12500000
0,25000000	0,20000000	0,16666666	0,14285714	0,12500000	0,11111110
0,20000000	0,16666666	0,14285714	0,12500000	0,11111110	0,09999999
0,16666666	0,14285714	0,12500000	0,11111110	0,09999999	0,09090909

(19.9) Таблица матрицы T_0, обратной к H_*, вычисленной без уточнения

36,05	—631,40	3369,20	—7583,47	7585,57	—2781,95
—631,45	14739,81	—88462,34	212349,10	—221227,39	83443,11
3369,79	—88468,85	566250,87	—1415714,73	1516905,83	—584028,87
—7585,42	212377,81	—1415794,56	3640509,72	—3981720,78	1557268,50
7588,01	—221268,61	1517059,27	—3981890,84	4424001,00	—1751805,56
—2783,02	83462,27	—584109,19	1557387,41	—1751862,88	700684,00

(19.10) Таблица матрицы T_*, обратной к H_*, вычисленной с использованием уточнения

36,09	—632,59	3377,15	—7603,97	7607,99	—2790,75
—632,59	14771,47	—88673,18	212891,85	—221821,99	83676,11
3377,15	—88673,18	567610,55	—1419212,98	1520737,50	—585529,96
—7603,97	212891,85	—1419212,98	3649301,66	—3991348,19	1561039,56
7607,99	—221821,99	1520737,50	—3991348,19	4434355,06	—1755860,73
—2790,75	83676,11	—585529,96	1561039,56	—1755860,73	702249,59

примерно 10^{-3}, т. е. находится в пределах машинного округления. Конечно, если процедура *IMPROVE* прерывается только после достижения максимально возможной точности, то ошибка и должна быть столь малой. (В действительности элементы

матрицы имеют двоичное представление, а не десятичное. Любая ошибка в (19.7) не превышает величины последнего двоичного разряда числа с плавающей запятой.) При вычислении (19.7) значение для *digits*, получаемое из *IMPROVE*, находится в пределах от 3,72 для первого столбца до 3,27 для последнего. Это является хорошей оценкой точности соответствующих столбцов в (19.6).

Четвертая матрица, (19.8), получена непосредственно по следующему алгоритму, описанному на ФОРТРАНе, при вычислениях с блокировкой округления:

```
      DO 1 I=1, N
      DO 1 J=1, N
    1 H(I, J)=1, 0/FLOAT(I+J-1).
```

Таким образом, (19.8) не является точно матрицей Гильберта. Она имеет ошибки в некоторых элементах, например $h_{13}=1/3$ не может быть точно представлен десятичной или двоичной дробью. Мы называем в этом случае H_* машинным представлением для H.

Пятая матрица, (19.9), получена обращением H_* без использования итерационного уточнения, а шестая, (19.10), получена применением уточнения к (19.9). Обозначим эти две матрицы соответственно через T_0 и T_*.

Заметим сразу же, что T_* сильно отличается от T. Однако (19.10) есть точная обратная матрица к (19.8), т. е.

$$T_*=H_*^{-1}$$

во всех приведенных десятичных знаках. Доказательство сходимости будет дано в разд. 22; другие вычисления также подтверждают, что это справедливо. Таким образом, все различие между T и T_* вызвано различием между H и H_*. Если мы хотим пронаблюдать эффект ошибки округления, вносимой в процессе исключения до итерационного уточнения, то мы должны сравнить T_0 с T_*, а не с T. Если этим соображением пренебречь, то использование матриц Гильберта в качестве тестовых будет некорректным.

Мы видим, что различие между T_0 и T_* в действительности меньше, чем между T_* и T. То есть ошибка округления, накапливаемая в процессе исключения при получении T_0, производит меньший эффект, чем начальная погрешность, вносимая при подготовке исходных данных. Это соображение будет

основой для рассмотрения и анализа ошибки округления в разд. 20 и 21.

В отличие от H_n машинное представление матрицы T_n, по крайней мере при небольших n, не приводит к изменению элементов, так как они являются целыми. Таким образом, матрицы, обратные к гильбертовым, являются лучшим примером при изучении погрешности округления, чем сами матрицы Гильберта. Они могут быть получены с помощью процедуры в программе (19.11) или же взяты из таблиц Саважа и Лукаса и введены в машину в качестве данных.

Однако и H_n, и T_n являются положительно определенными матрицами, для которых, как можно показать, выбор главных элементов не обязателен. (То есть размеры ошибки будут приемлемы и при гауссовском исключении без выбора главных элементов. Поэтому нельзя ожидать, что изменение стратегии выбора главных элементов повлияет на точность.) Следовательно, матрицы Гильберта не следует употреблять для проверки методов выбора главных элементов, разработанных для матриц общего вида. Это второй пример иногда встречающегося неправильного использования гильбертовых матриц.

Наши шесть матриц иллюстрируют также важный для итерационного уточнения момент. Если начальные данные, например H_n, содержат ошибку или подвергаются возмущению, то результат, получаемый по гауссовскому исключению без уточнения, обычно имеет ту точность, которую гарантируют данные. Если, наоборот, данные, как в T_n, точны, то возникает законный вопрос, будет ли решение точным или приближенным. В этом последнем случае применение итерационного уточнения оправданно и полезно.

Относительно дальнейшего изучения матриц Гильберта см. Тодд [28], [29].

Плохая обусловленность матриц Гильберта может быть связана с проблемой аппроксимации, которую мы использовали для введения этих матриц. На интервале $0 \leqslant x \leqslant 1$ функции x^i $(i=0, \ldots, n-1)$ почти линейно зависимы. Это означает, что строки гильбертовой матрицы почти линейно зависимы, т. е. матрица почти вырожденна. Как мы видели, в таких случаях небольшие возмущения в данных вызывают огромные изменения в результате. В исходной задаче небольшие ошибки в значениях функции $f(x)$ или ошибки округления при ее вычислении могут привести к большим изменениям в коэффициентах c_i. Короче говоря, задача аппроксимации не является «хорошо поставленной», если она приводит к матрицам, подобным гильбертовым.

(19.11)
Алгольная программа обращения гильбертовых матриц

```
procedure Inverse Hilbert (n, T);
value n; integer n; array T;
comment образование матрицы, обратной к (n×n)-
        матрице Гильберта
comment Используются действия с целыми числами.
        Результат точен, если n достаточно мало,
        так что не происходит переполнения.
        Деление целых ÷ может быть заменено
        на /, если при этом не вводится ошибки
        округления;
begin
  integer p, r, i, j;
  p := n;
  for i := 1 step 1 until n do
  begin
    if i≠1 then
    p := ((n − i + 1) × p × (n + i − 1)) ÷ (i − 1)↑2;
    r := p↑2;
    T[i, i] := r ÷ (2 × i − 1);
    for j := i + 1 step 1 until n do
    begin
      r := −((n − j + 1)×r×(n + j − 1)) ÷ (j − 1)↑2;
      T[j, i] := T[i, j] := r ÷ (i + j − 1);
    end;
  end;
end Inverse Hilbert;
```

(19.12) Упражнение. Упражнение (8.18) показывает, что

$$\frac{\|T_* - T\|}{\|T_*\|} \leqslant \operatorname{cond}(H) \frac{\|H_* - H\|}{\|H\|}.$$

Используя упражнение (2.19) и табл. (19.4), найти оценки для величин, входящих в это неравенство. Является ли неравенство неулучшаемым? Почему?

(19.13) Упражнение. Провести обращение матриц T_n и $(H_n)_*$ как с применением итерационного уточнения, так и без него. Использовать в качестве значений n целые

числа 2, 3, ..., пока элементы T_n не станут настолько большими, что не смогут быть точно представлены, или итерационное уточнение не перестанет сходиться. Если возможно, провести вычисления с длиной машинного слова, отличной от той, которая использована в наших примерах. (В нашем случае использовалась машина IBM 7090, которая имеет 27 двоичных разрядов с плавающей запятой. При этом максимальная обращаемая матрица Гильберта была $(H_7)_*$.)

(19.14) Упражнение. Доказать, что равенство нулю $\frac{\partial E}{\partial c_l}$ действительно обеспечивает минимум E, как это утверждалось в начале разд. 19. Доказать также, что вектор решения c есть единственный вектор, минимизирующий E.

20. АНАЛИЗ ОШИБОК ОКРУГЛЕНИЯ В СИСТЕМЕ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Мы прервем исследование матриц, чтобы изложить основы анализа округлений при вычислениях с плавающей запятой, следуя теории Уилкинсона [32]. Этот материал будет использован при изучении ошибок округления, возникающих при решении систем линейных уравнений методом исключения.

Мы опишем так называемую *нормализованную* вычислительную систему с плавающей запятой, хотя существуют и другие. Какое-то целое число β выбирается в качестве основания, и t разрядов d_i с основанием β составляют *значащую часть* s числа (также называемую *дробной частью* или *мантиссой*). Мы рассматриваем системы с $\beta=2$, 8, 10 и 16; соответствующие значения t могут сильно меняться. Задается также некоторая область для целых показателей степени e, например, $-m \leqslant e \leqslant M$. Наконец, число имеет знак. Таким образом, ненулевое число в системе с плавающей запятой имеет вид

$$\pm\, 0{,}d_1 d_2 \ldots d_t \times \beta^e,$$

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

$$\begin{aligned} &1 \leqslant d_1 \leqslant \beta - 1, \\ &0 \leqslant d_i \leqslant \beta - 1 \quad \text{для} \quad i = 2, 3, \ldots, t, \\ &-m \leqslant e \leqslant M. \end{aligned}$$

Условие $d_1 \neq 0$ является характерным признаком *нормализованного числа* в системе с плавающей запятой.

Если $s = \pm 0{,}d_1 d_2 \ldots d_t$, то найдется целое N, такое, что

$$|s| = N\beta^{-t} \text{ и } \beta^{t-1} \leqslant N < \beta^t.$$

Представление нуля в системе с плавающей запятой может быть разным: обычно $s=+0$ и $e=-m$.

На вещественной числовой оси числа в системе с плавающей запятой образуют некоторое ограниченное множество $F=F(\beta, t, m, M)$ отнюдь не равноотстоящих точек. На рис. (20.1) схематически представлены 33 точки F для $\beta=2$, $t=3$, $m=1$ и $M=2$.

(20.1)

Числа в системе с плавающей запятой для $\beta=2$, $t=3$, $m=1$, $M=2$

Любое вещественное число x, которое мы хотим представить в машине, аппроксимируется числом из только что описанного множества F — чаще всего ближайшим числом из F с некоторым выбором в случае неоднозначности. Для данного x мы обозначаем через x_R («x округленное») ближайшее число из F; в случае неоднозначности мы выбираем в качестве x_R, то из двух ближайших чисел из F, у которого больше абсолютная величина. Если $x=0$, то $x_R=0$. Если $x\neq 0$, то выбираем s и e так, что

(20.2) $$|x| = s\times\beta^e, \text{ где } \beta^{-1}\left(1-\frac{1}{2}\beta^{-t}\right)\leqslant s<1-\frac{1}{2}\beta^{-t}.$$

Если e лежит вне интервала $-m\leqslant e\leqslant M$, то мы не имеем приемлемого представления для x; мы просто будем говорить, что x находится вне области чисел в системе с плавающей запятой, не останавливаясь на этом подробно. Если e находится внутри допустимой области, мы берем бесконечное представление числа $s+{}^1/_2\beta^{-t}$ с основанием β (с нулями на конце в случае конечного представления) и отбрасываем все разряды после первых t вправо от запятой. Тогда x_R состоит из знака x, только что полученных t разрядов и e, определенного в (20.2).

(20.3) Упражнение. Показать, что если $\beta^{e-1}\left(1-\frac{1}{2}\beta^{-t}\right)\leqslant|x|<$ $<\beta^e\left(1-\frac{1}{2}\beta^{-t}\right)$, то

$$x_R = \operatorname{sign}(x)\cdot\operatorname{entier}\left(\beta^{t-e}|x|+\frac{1}{2}\right)\cdot\beta^{e-t},$$

где $\operatorname{entier}(y)$ обозначает наибольшее целое, не превосходящее y. (Математики также используют обозначения $[y]$ и $E(y)$ для той же самой функции.)

Следующая теорема является основной для теории ошибок округления в системах с плавающей запятой; она дает удобное выражение для относительной ошибки, возникающей при замене x на x_R.

(20.4) Теорема. *Если x — вещественное число из области чисел в системе с плавающей запятой, то*

(20.5) $$x_R = x(1+\delta), \text{ где } |\delta|\leqslant\frac{1}{2}\beta^{1-t}.$$

Для доказательства (20.4) предположим, что $x>0$, так как для $x<0$ все выкладки аналогичны, а случай $x=x_R=0$

тривиален. Пусть e является тем целым числом, для которого

$$\beta^{e-1} \leqslant x \leqslant \beta^{e}.$$

В области $[\beta^{e-1}, \beta^{e}]$ в системе с плавающей запятой числа распределены равномерно с интервалом β^{e-t}. Ближайшее к x число есть x_R и оно должно находиться от x не дальше, чем на расстоянии $^1/_2\beta^{e-t}$, т. е.

$$|x_R - x| \leqslant \frac{1}{2}\beta^{e-t}.$$

Так как $\beta^{e-1} \leqslant x$, то

$$\frac{|x_R - x|}{|x|} \leqslant \frac{\frac{1}{2}\beta^{e-t}}{\beta^{e-1}} = \frac{1}{2}\beta^{1-t}.$$

Тем самым, поскольку

$$\delta = (x_R - x)/x,$$

теорема доказана.

Пусть, для примера, $\beta = 10$, $t = 4$ и $x = \pi$. Десятичное представление π имеет вид

$$3{,}14159265\ldots = +0{,}314159265\ldots \times 10^1.$$

Таким образом, $e = 1$ и $x_R = 0{,}3142 \times 10^1$. Следовательно,

$$\delta = (x_R - x)/x = -0{,}00012966\ldots.$$

Так как $^1/_2\beta^{1-t} = {}^1/_2(10^{-3}) = 0{,}0005$, то очевидно, что δ удовлетворяет неравенству $|\delta| \leqslant {}^1/_2\beta^{1-t}$.

Чтобы показать, что $^1/_2\beta^{1-t}$ является почти точной границей для $|\delta|$, рассмотрим $x = \beta^a(1 + {}^1/_2\beta^{1-t})$ для некоторого a. Тогда

$$\frac{|x_R - x|}{|x|} = \frac{\frac{1}{2}\beta^{1-t}\beta^a}{\beta^a\left(1 + \frac{1}{2}\beta^{1-t}\right)} = \frac{\frac{1}{2}\beta^{1-t}}{1 + \frac{1}{2}\beta^{1-t}},$$

что очень близко к $^1/_2\beta^{1-t}$ для практических значений t.

Вторым способом представления вещественных чисел в системе с плавающей запятой является *урезание* числа. Для данного вещественного числа, лежащего в области чисел в системе с плавающей запятой, мы выбираем x_C («урезанное x») как единственное число в F, ближайшее к x и удовлетворяющее условию $|x_C| \leqslant |x|$; x_C можно получить, если взять бесконечное представление числа x с основанием β (с нулями на конце в случае конечного представления) и отбросить значащие разряды после t старших.

(20.6) Упражнение. Доказать, что если $\beta^{a-1} \leqslant |x| \leqslant \beta^a$, то

$$x_C = \operatorname{sign}(x) \cdot \operatorname{entier}\left(\beta^{t-a}|x|\right)\beta^{a-t}.$$

(20.7) Упражнение. Доказать, что если x находится в области чисел в системе с плавающей запятой, то

$$x_C = x(1+\delta), \quad |\delta| < \beta^{1-t}.$$

Мы хотим теперь обсудить проблему ошибок округления в арифметических операциях. Мы не можем рассмотреть все вычислительные машины, так как между ними существует большое различие в выполнении операций. Однако границы ошибок для различных машин меняются незначительно. Мы ограничимся двумя типами арифметических операций — с *округлением* и с *блокировкой округления*.

Для двух данных чисел x, y в системе с плавающей запятой через

$$\mathrm{fl}(x+y),\ \mathrm{fl}(x-y),\ \mathrm{fl}(x+y),\ \mathrm{fl}(x/y)$$

мы будем обозначать соответственно результаты сложения, вычитания, умножения и деления при вычислениях с плавающей запятой независимо от того, получен ли результат с округлением или с блокировкой округления. Мы будем предполагать, что система с плавающей запятой ведет себя следующим образом.

Операция с округлением: для любых двух чисел x и y в F находится точно число $x+y$, $x-y$, $x \times y$ или x/y и затем округляется. (При делении предполагаем, что $y \neq 0$.) Таким образом,

$$\mathrm{fl}(x+y) = (x+y)_R;\ \mathrm{fl}(x-y) = (x-y)_R;$$
$$\mathrm{fl}(x \times y) = (x \times y)_R;\ \mathrm{fl}(x/y) = (x/y)_R.$$

Операция с блокировкой округления: для любых двух чисел x и y в F существуют

$$\mathrm{fl}(x \pm y) = (x \pm y)_C;\quad \mathrm{fl}(x \times y) = (x \times y)_C;\quad \mathrm{fl}(x/y) = (x/y)_C.$$

Из (20.4) и (20.7) получаем следующую теорему:

(20.8) Теорема. *Пусть $*$ означает любую из операций $+$, $-$, $\times$, $/$. Тогда*

$$\mathrm{fl}(x * y) = (x * y)(1+\delta),$$

где
$$\begin{cases} |\delta| \leqslant \frac{1}{2}\beta^{1-t} & \text{(операции с округлением)}, \\ |\delta| < \beta^{1-t} & \text{(операции с блокировкой округления)}. \end{cases}$$

В большинстве случаев операции с округлением предпочтительнее операций с блокировкой округления. Для последних граница ошибки вдвое больше, чем для операций с округлением. Более важное различие заключается в следующем: при сложении с блокировкой округления большого числа n положительных чисел ошибка после n шагов может быть очень большой, так как погрешность урезания числа имеет в этом случае одинаковый знак, а это вносит систематическую ошибку. В операциях с округлением встречаются как положительные, так и отрицательные ошибки, поэтому взаимное уничтожение ошибок разного знака обычно приводит к много меньшей величине суммарной ошибки после n сложений, чем при операциях с блокировкой округления. Действительно, если все ошибки имеют одинаковую величину, а их знаки независимы и среди них одинаково часто встречаются $+$ и $-$, то из предельной теоремы теории вероятностей следует, что вероятная ошибка суммы n положительных членов с блокировкой округления будет приблизительно в $\sqrt{n}$ раз больше ошибки суммы с округлением.

Тем не менее, как это ни удивительно, вычисления с блокировкой округления широко распространены в системах с плавающей запятой и используются, например, в языках ФОРТРАН II и ФОРТРАН IV для машины IBM 7090. Вычисления с блокировкой округления имеют то существенное преимущество, что знак ошибки известен.

Для удобства мы введем следующее сокращение:

(20.9) Определение. Пусть u означает *единичную ошибку округления*, измеренную в относительных единицах. Тогда

$$u=\frac{1}{2}\beta^{1-t} \quad \text{(операции с округлением)},$$

$$u=\beta^{1-t} \quad \text{(операции с блокировкой округления)}.$$

С помощью полученных выше фундаментальных оценок для операций с плавающей запятой мы можем построить другие. Например, определив $\mathrm{fl}(x+y+z)$ как

$$\mathrm{fl}(\mathrm{fl}(x+y)+z),$$

получим

$$\begin{aligned}(20.10)\quad \mathrm{fl}(x+y+z) &= \mathrm{fl}((x+y)(1+\delta_1)+z) = \\ &= ((x+y)(1+\delta_1)+z)(1+\delta_2) = \\ &= (x+y)(1+\delta_1)(1+\delta_2)+z(1+\delta_2), \quad |\delta_i|\leqslant u.\end{aligned}$$

Таким же образом, складывая по очереди слагаемые от $i=1$ до $i=4$, получаем

$$\mathrm{fl}(x_i \times y_i) = x_i y_i (1+\delta_i), \quad |\delta_i| \leqslant u,$$

так что

$$\begin{aligned}(20.11)\quad \mathrm{fl}\left(\sum_{i=1}^{4} x_i \times y_i\right) &= (((x_1y_1(1+\delta_1)+x_2y_2(1+\delta_2))(1+\delta_5)+\\ &+x_3y_3(1+\delta_3))(1+\delta_6)+x_4y_4(1+\delta_4))(1+\delta_7)=\\ &=x_1y_1(1+\delta_1)(1+\delta_5)(1+\delta_6)(1+\delta_7)+\\ &+x_2y_2(1+\delta_2)(1+\delta_5)(1+\delta_6)(1+\delta_7)+\\ &+x_3y_3(1+\delta_3)(1+\delta_6)(1+\delta_7)+\\ &+x_4y_4(1+\delta_4)(1+\delta_7),\end{aligned}$$

где $|\delta_i| \leqslant u$ для всех i.

Поскольку часто встречается сомножитель $1+\delta_i$, желательно получить для него некоторые приемлемые оценки. Этой цели служит следующая лемма:

(20.12) Лемма. *Если* $0 \leqslant u \leqslant 1$ *и если* $n=1, 2, 3, \ldots,$ *то*

$$1-nu \leqslant (1-u)^n.$$

Доказательство. Пусть $f(u)=(1-u)^n$. Разложение в ряд Тейлора дает

$$\begin{aligned}f(u) &= f(0)+uf'(0)+\frac{f''(\theta u)}{2}u^2=\\ &=1-nu+\frac{n(n-1)}{2}(1-\theta u)^{n-2}u^2, \quad 0<\theta<1.\end{aligned}$$

Так как остаточный член неотрицателен, мы имеем $f(u) \geqslant 1-nu$, что доказывает (20.12).

(20.13) Лемма. *Если* $n=1, 2, \ldots$ *и если* $0 \leqslant nu \leqslant 0{,}01$, *то*

$$(1+u)^n \leqslant 1+1{,}01nu.$$

Доказательство. Можно показать, что

$$(20.14)\qquad 1+x \leqslant e^x \quad \text{для всех} \quad x \geqslant 0;$$

$$(20.15)\qquad e^x \leqslant 1+1{,}01x \quad \text{для всех} \quad 0 \leqslant x \leqslant 0{,}01.$$

(Оставляем доказательство этих неравенств читателю.) Тогда

$$(1+u)^n \leqslant (e^u)^n = e^{nu} \leqslant 1+1{,}01nu \quad \text{по (20.14) и (20.15).}$$

В приложениях, для которых предназначена эта лемма, n есть порядок матрицы, а u — единичная ошибка округления. Следовательно, предположение $nu \leqslant 0{,}01$ выполняется во всех практических задачах.

(20.16) Лемма. *Если* $|\delta_i| \leqslant u$ *для* $i=1, \ldots, n$ *и если* $nu \leqslant 0{,}01$, *то*

$$1 - nu \leqslant \prod_1^n (1+\delta_i) \leqslant 1 + 1{,}01nu. \tag{20.17}$$

Доказательство. Поскольку

$$(1-u)^n \leqslant \prod_1^n (1+\delta_i) \leqslant (1+u)^n,$$

доказательство следует из (20.12) и (20.13).

Замечание. Неравенства (20.17) удобно представить в форме

$$\prod_1^n (1+\delta_i) = 1 + 1{,}01n\theta u, \quad |\theta| \leqslant 1.$$

Используя (20.16), при $4u \leqslant 0{,}01$ можно следующим образом переписать полученный ранее результат (20.11):

$$\mathrm{fl}\left(\sum_{i=1}^4 x_i \times y_i\right) = x_1y_1(1+4{,}04\theta_1 u) + x_2y_2(1+4{,}04\theta_2 u) + \\ + x_3y_3(1+3{,}03\theta_3 u) + x_4y_4(1+2{,}02\theta_4 u), \quad |\theta_i| \leqslant 1.$$

Аналогично можно доказать следующую теорему:

(20.18) Теорема. *Если* $nu \leqslant 0{,}01$, *то*

$$\mathrm{fl}\left(\sum_{i=1}^4 x_i \times y_i\right) = \sum_{i=1}^n x_iy_i[1 + 1{,}01(n+2-i)\theta_i u], \quad |\theta_i| \leqslant 1. \tag{20.19}$$

При выводе (20.19) мы для простоты ослабили оценку для $i=1$ с $1{,}04n\theta_1 u$ до $1{,}01(n+1)\theta_1 u$.

Эти оценки для операций с плавающей запятой позволяют нам определить погрешность в самых разнообразных вычислениях. Одной из наиболее общих операций в матричных вычислениях является образование скалярного произведения двух векторов. Например, скалярное произведение векторов x и y есть $\sum_{i=1}^n x_iy_i$. При перемножении двух матриц эту опера-

цию требуется выполнить n^2 раз. Некоторые цифровые вычислительные машины (например, большинство настольных вычислительных машин) имеют устройство для накопления промежуточных величин $\sum_{i=1}^{k} x_i y_i$. Это является преимуществом, позволяющим исключить некоторые пересылки и хитроумные операции, а также эффективно использовать индексные регистры на некоторых машинах. (Такие преимущества были основными факторами развития алгоритма Краута.) При матричных вычислениях может быть достигнуто также значительное уменьшение ошибок округления, если регистр накопления (как в настольных вычислительных машинах) имеет в своей мантиссе существенно больше разрядов, чем число разрядов x_i и y_i. Так как зачастую ошибки округления требуется уменьшить до минимума, мы приведем соответствующие оценки для накопления с двойной точностью. Они могут быть сравнены с оценками для накопления с обычной точностью, представленными в (20.19).

Для большей ясности предположим, что x_i и y_i хранятся как обычные числа в системе с плавающей запятой с t разрядами в мантиссе, но что накопление суммы выполняется в регистре с плавающей запятой с мантиссой, имеющей $2t$ разрядов. После того как скалярное произведение накоплено полностью, оно округляется до t разрядов и запоминается. Мы обозначим результат, округленный до t разрядов, через

$$\mathrm{fl}_2\left(\sum_{i=1}^{n} x_i \times y_i\right).$$

Как представить ошибку округления для вычисленного скалярного произведения? Вплоть до окончательного округления до t разрядов, накопление внутреннего произведения происходит почти как в обычных вычислениях с плавающей запятой с $2t$ разрядами, за исключением того, что умножение выполняется без ошибок. Если бы накопление происходило в точности таким же образом, то, используя формулу (20.19), скалярное произведение можно было бы представить посредством

$$\sum_{i=1}^{n} x_i y_i \left\{1 + 1{,}01\,(n+1-i)\,\theta_i F \frac{1}{2}\beta^{1-2t}\right\}, \quad |\theta_i| \leqslant 1, \tag{20.20}$$

где $F=1$. Однако сложение чисел с удвоенным числом разрядов обычно происходит с большей относительной ошибкой, чем сложение с обычной точностью, и поэтому представление

(20.20), как правило, не справедливо. Мы приводим без доказательства утверждение, что представление (20.20) справедливо для обычного накопления с двойной точностью с множителем $F=1+1/\beta$. Подробные объяснения для $\beta=2$ или 10 см. Уилкинсон [32].

Для основного результата (20.21) различие между $F=1$ и $F=1+1/\beta$ практически несущественно, потому что после окончательного округления до t разрядов имеем

$$(20.21)\quad \mathrm{fl}_2\left(\sum_{i=1}^{n} x_i \times y_i\right) = \left(1+\theta_0 \frac{1}{2}\beta^{1-t}\right)\sum_{i=1}^{n} x_i y_i \left\{1+1{,}01\,(n+1-i)\,\theta_i F \frac{1}{2}\beta^{1-2t}\right\},$$

где $F=1+1/\beta$ и все $|\theta_i|\leqslant 1$. На обычных машинах для достижимых значений n величина $1{,}01nF\frac{1}{2}\beta^{1-2t}$ на много порядков меньше, чем $u=\frac{1}{2}\beta^{1-t}$.

При отсутствии большой потери значащих цифр, как показывает равенство (20.21),

$$\mathrm{fl}_2\left(\sum_{i=1}^{n} x_i \times y_i\right)$$

отличается от истинного значения $\sum_{i=1}^{n} x_i y_i$ на величину, лишь немногим превосходящую единичную ошибку округления.

(20.22) Упражнение. С помощью примера с $\beta=10$, $t=3$ показать, что $\mathrm{fl}_2\left(\sum_{i=1}^{n} x_i \times y_i\right)$ может отличаться от $\sum_{i=1}^{n} x_i y_i$ при наличии потери значащих цифр на большую относительную величину.

(20.23) Упражнение. Доказать с помощью (20.21), что если nu достаточно мало, то

$$\left|\mathrm{fl}_2\left(\sum_{i=1}^{n} x_i \times y_i\right) - \sum_{i=1}^{n} x_i y_i\right| \leqslant 1{,}01u \cdot \|x_2\| \cdot \|y_2\|.$$

Полученные выше оценки ошибок можно трактовать как выражение результатов точных вычислений в системе с плавающей запятой при слегка измененных данных. Таким обра-

зом, равенство

$$\mathrm{fl}(x \times y) = xy(1+\delta)$$

можно интерпретировать как точное произведение двух неизвестных чисел $x'=x$ и $y'=y(1+\delta)$, относительно мало отличающихся от x и y соответственно. Еще один из многих возможных способов: положим $x'=x(1+\delta)^{1/2}$ и $y'=y(1+\delta)^{1/2}$. Такой подход называется *обратным анализом округлений*, так как ошибки приписываются данным.

В *прямом анализе округлений* умножение в системе с плавающей запятой (например) рассматривается как операция, аппроксимирующая точное умножение. Затем дается оценка разности между произведением в системе с плавающей запятой $\mathrm{fl}(x \times y)$ и точным произведением xy. Трудность здесь та, что произведение в системе с плавающей запятой не является ассоциативной операцией, и поэтому обычный анализ не может быть использован. Требуется построить совершенно новый принцип анализа, что очень обременительно и скучно. (См., например, Хаусхолдер [41], гл. 1.) Более того, сложение в системе с плавающей запятой не ассоциативно, и закон дистрибутивности также не выполняется.

Так как в обратном анализе округлений результат операций в системе с плавающей запятой интерпретируется как результат обычных вычислений, мы без труда можем использовать правила алгебры. Такой метод поэтому легче в использовании и менее чувствителен к ошибкам программиста.

Обратный анализ округлений был введен Гивенсом [9]. В других разделах теории ошибок эта идея была использована Ланцошем (τ-метод); см., например, его работу [17].

Иногда можно ограничиться просто обратным анализом ошибок, как в двух следующих примерах. В первом предполагаем, что требуется решить линейную систему $Ax \doteq b$. Обратный анализ ошибок, изложенный в разд. 21, утверждает, что вычисленное решение x является точным решением линейной системы $A'x=b$, где A' — некоторая неизвестная матрица с тем свойством, что $\|A'-A\| < \varepsilon$ (ε — некоторое определенное число). Далее, если мы знаем заранее, что матрица A сама неопределенна или известна с ошибкой, много большей ε, то мы будем удовлетворены, если вычисленное x является настолько хорошим решением, насколько можно ожидать. Едва ли необходимо знать, насколько близко x к «истинному» решению $A^{-1}b$.

Во втором примере мы просим читателя вообразить, что вектор x характеризует положение ракеты, соответствующее различным моментам времени при перелете с Земли на Марс.

Предположим также, что Ax представляет собой производную по времени от количества движения ракеты в те же самые моменты времени. Наконец, предположим, что b представляет силу, действующую на ракету в эти же моменты времени, включая как внешие силы, так и силу тяги ракеты. Тогда система уравнений $Ax=b$ представляет баланс сил, действующих на ракету. Для любого заданного b истинное решение x представляет траекторию ракеты. Мы можем также сказать, что вычисленное x точно удовлетворяет системе $Ax=b+\delta b$. Другими словами, ракета может удерживаться на траектории x с помощью дополнительных сил δb. Если известно, что $\|\delta b\|<\varepsilon$, мы имеем оценку величины необходимых корректирующих сил. Если ε достаточно мало по сравнению с величиной корректирующих сил, имеющихся в распоряжении ракеты в виде резервной тяги, мы можем быть уверены, что вычисленная траектория достижима. То есть любые ошибки округления, которые мы можем сделать при вычислении x, легко компенсируются изменением b. Таким образом, может быть более целесообразно исследовать величину δb, нежели задаваться вопросом, насколько неверно x при фиксированной правой части b.

Несмотря на приведенные выше примеры, во многих случаях при постановке физической задачи действительно нужно знать, насколько вычисленное решение x системы $Ax=b$ близко к истинному решению $A^{-1}b$. Однако обратный анализ ошибок может быть очень полезным промежуточным шагом в решении этого вопроса, так как если известно, что $Ax=b+\delta b$, где $\|\delta b\|<\varepsilon$, то

$$\begin{aligned}\|x-A^{-1}b\| &= \|A^{-1}(Ax-b)\| = \\ &= \|A^{-1}(\delta b)\| \leqslant \\ &\leqslant \|A^{-1}\|\cdot\|\delta b\| < \\ &< \|A^{-1}\|\cdot\varepsilon.\end{aligned}$$

Следовательно, если мы имеем некоторую верхнюю оценку для $\|A^{-1}\|$, мы можем от оценки $\|\delta b\|$ перейти к оценке $\|x-A^{-1}b\|$. Этот подход к получению оценки $\|x-A^{-1}b\|$ может оказаться много легче фронтальной атаки посредством прямого анализа ошибок.

(20.24) Упражнение. Пусть $Q(t)$ есть результат вычисления значения полинома

$$P(t)=\sum_{k=0}^{n} a_k t^k$$

по следующему алгоритму:

$$q := a_n;$$
$$\textbf{for } k := n-1 \textbf{ step } -1 \textbf{ until } 0 \textbf{ do}$$
$$q := \mathrm{fl}(t \times q + a_k).$$

Доказать, что

$$Q(t) = \sum_{k=0}^{n} [1 + 1{,}01\,(2k+1)\,\theta_k u]\, a_k t^k,$$

где все $|\theta_k| \leqslant 1$. (Предполагается, что t, a_k и все промежуточные значения являются числами в системе с плавающей запятой.)

(20.25) Задача для исследования. Исследовать ошибку округления при использовании (14.2) в качестве метода избежания переполнения или исчезновения разрядов при вычислении $\det(A)$. Рассмотреть два случая: когда вычисление логарифмов с двойной точностью возможно и когда нет.

(20.26) Упражнение. Доказать (20.14) и (20.15).

(20.27) Упражнение. В некоторых приложениях, как показал Кахан, другая форма теоремы (20.8) приводит к несколько более простому выражению. Доказать теорему

(20.28) Теорема. *Пусть* $*$ *означает* $+$, $-$, $\times$, $/$. *Тогда*

$$\mathrm{fl}(x * y) = \frac{x*y}{1+\delta},$$

$$\text{где} \begin{cases} |\delta| \leqslant \frac{1}{2}\beta^{1-t} & (\textit{операции с округлением}), \\ |\delta| \leqslant \beta^{1-t} & (\textit{операции с блокировкой округления}) \end{cases}$$

21. ОШИБКИ ОКРУГЛЕНИЯ В ГАУССОВСКОМ МЕТОДЕ ИСКЛЮЧЕНИЯ

В двух следующих разделах мы объединим математический аппарат, введенный в предыдущих разделах, — нормы векторов и матриц, гауссовское исключение, итерационное уточнение и анализ ошибок округления в системе с плавающей запятой — для изучения поведения наших алгоритмов на вычислительной машине.

Пусть A, b и x — соответственно заданная матрица, заданная правая часть и решение, вычисленное по методу исключения Гаусса с выбором главного элемента. (Элементы матрицы A и векторов b и x должны быть числами в системе с плавающей запятой.) В этом разделе мы покажем, что вычисленное решение x точно удовлетворяет возмущенному уравнению

$$(A+\delta A)\,x=b, \tag{21.1}$$

где δA представляет собой некоторую «малую» матрицу. Мы свяжем верхнюю границу малости δA с нормой A и единичной ошибкой округления u. Заметим, что выражение (21.1) является примером обратного анализа ошибок, описанного в разд. 20.

Большинство используемых нами идей были развиты в последние годы Уилкинсоном [31]—[33]. Мы просто применяем его результаты в нашем конкретном случае. См. также Фокс [36] и Вендрофф [8].

Мы находим более удобным использовать для векторов и матриц *максимум-нормы*, нежели евклидовы нормы, использованные ранее. Эти нормы, приведенные в разд. 2 и 11, определяются как

$$\|x\|_\infty=\max_{1\leqslant i\leqslant n}|x_i|, \tag{21.2}$$

если $x=(x_1,\ldots,x_n)^T$ есть вектор, и

$$\|A\|_\infty=\max_{1\leqslant i\leqslant n}\sum_{j=1}^{n}|a_{ij}|, \tag{21.3}$$

если $A=(a_{ij})$ — матрица. Легко видеть, что если вектор x удовлетворяет неравенству $|x_i|\leqslant c$, $i=1,\ldots,n$, то

$$\|x\|_\infty\leqslant c, \tag{21.4}$$

и если для матриц A и B выполняются неравенства $|a_{ij}| \leqslant |b_{ij}|$, $i, j=1, \ldots, n$, то

$$\|A\|_\infty \leqslant \|B\|_\infty. \tag{21.5}$$

Евклидова норма (2.6) уже не обладает свойством (21.5).

Читатель может проверить (см. упражнение (2.18)), что

$$\|A\|_\infty = \max_{\|x\|_\infty=1} \|Ax\|_\infty. \tag{21.6}$$

Как мы видели, первый и самый трудоемкий шаг в гауссовском исключении есть разложение матрицы A в произведение двух треугольных матриц L и U. Предположим, что матрица A заранее масштабирована по строкам и строки ее упорядочены так, что не требуют выбора главного элемента. На практике, конечно, этого может и не быть. Однако выбор главного элемента сводится лишь к перестановкам индексов строк и не влияет на анализ ошибок, так что мы в дальнейшем просто будем им пренебрегать.

Разложение заключается в вычислении последовательности матриц $A^{(1)}=A, A^{(2)}, \ldots, A^{(n)}$, причем элементы матрицы $A^{(k)}$ ниже диагонали в первых $k-1$ столбцах равны нулю. Матрица $A^{(k+1)}$ получается из матрицы $A^{(k)}$ вычитанием k-й строки, умноженной на соответствующий коэффициент, из каждой строки, расположенной ниже нее; остальные строки $A^{(k)}$ не меняются. Коэффициенты подбираются так, чтобы при отсутствии округлений элементы матрицы $A^{(k+1)}$ ниже диагонали в k-м столбце были бы нулями. Мы не вычисляем эти элементы, но берем их равными нулю по определению. (В наших программах эти «дыры» в A используются для хранения коэффициентов, однако это не влияет на анализ ошибок.)

Более строго, если $A^{(k)}$ имеет элементы $a_{ij}^{(k)}$, то, полагая

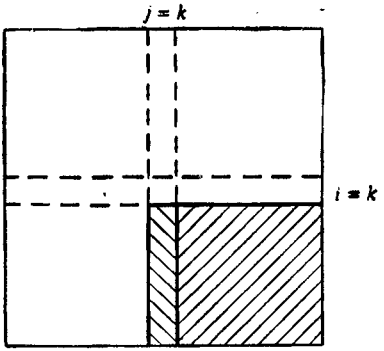
$$m_{ik} = \mathrm{fl}\left(a_{ik}^{(k)}/a_{kk}^{(k)}\right), \qquad i \geqslant k+1, \tag{21.7}$$

получаем

$$a_{ij}^{(k+1)} = \begin{cases} 0 & \text{для } i \geqslant k+1,\ j=k, \\ \mathrm{fl}\left(a_{ij}^{(k)} - m_{ik} \times a_{kj}^{(k)}\right) & \text{для } i \geqslant k+1,\ j \geqslant k+1, \\ a_{ij}^{(k)} & \text{для других } i,\ j. \end{cases} \tag{21.8}$$

Рис. (21.9) иллюстрирует куски матрицы, используемые в этих трех случаях. Такие шаги выполняются для $k=1, \ldots, n-1$.

(21.9)



Наконец, пусть

(21.10) $$U = A^{(n)},$$

(21.11) $$L = \begin{bmatrix} 1 & & & & 0 \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & \cdot & & \\ \vdots & \vdots & & \ddots & \\ m_{n1} & m_{n2} & \cdot & \cdots & 1 \end{bmatrix}.$$

Ясно, что L и U есть соответственно нижняя и верхняя треугольные матрицы. Мы хотим доказать, что

$$LU = A + E,$$

где E — матрица с малыми элементами, учитывающая ошибки округления.

Ошибка при вычислении коэффициента m_{ik} выражается из формулы

(21.12) $$m_{ik} = \left(a_{ik}^{(k)}/a_{kk}^{(k)}\right)(1 + \delta_{ik}), \quad \text{где } |\delta_{ik}| \leqslant u,$$

или

(21.13) $$0 = a_{ik}^{(k)} - m_{ik}a_{kk}^{(k)} + a_{ik}^{(k)}\delta_{ik},$$

т. е.

(21.14) $$\varepsilon_{ik}^{(k)} = a_{ik}^{(k)}\delta_{ik} \quad \text{для } i \geqslant k + 1$$

является ошибкой, возникающей в результате приравнивания $a_{ik}^{(k+1)}$ нулю. Для второго случая из (21.8) получаем (используя (20.8) и (20.28))

$$(21.15)\qquad a_{ij}^{(k+1)} = \mathrm{fl}\left(a_{ij}^{(k)}\right) - \mathrm{fl}\left(m_{ik} \times a_{kj}^{(k)}\right) = \\ = \left(a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}\left(1+\delta_{ij}\right)\right)/\left(1+\delta'_{ij}\right), \text{ где } |\delta_{ij}| \leqslant u, \\ |\delta'_{ij}| \leqslant u,$$

или

$$(21.16)\qquad a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)} - m_{ik}a_{kj}^{(k)}\delta_{ij} - a_{ij}^{(k+1)}\delta'_{ij}.$$

Таким образом,

$$(21.17)\qquad \varepsilon_{ij}^{(k)} = -m_{ik}a_{kj}^{(k)}\delta_{ij} - a_{ij}^{(k+1)}\delta'_{ij} \quad \text{для } i \geqslant k+1,\ j \geqslant k+1$$

есть ошибка при вычислении $a_{ij}^{(k+1)}$ для «новой» части матрицы $A^{(k+1)}$. Остальная часть $A^{(k+1)}$ берется прямо из $A^{(k)}$ без ошибок.

Таким образом,

$$(21.18)\qquad \varepsilon_{ij}^{(k)} = \begin{cases} a_{ik}^{(k)}\delta_{ik} & \text{для } i \geqslant k+1,\ j = k, \\ -m_{ik}a_{kj}^{(k)}\delta_{ij} - a_{ij}^{(k+1)}\delta'_{ij} & \text{для } i \geqslant k+1,\ j \geqslant k+1, \\ 0 & \text{для других } i,\ j. \end{cases}$$

Следовательно, если матрица $E^{(k)}$ имеет элементы $\varepsilon_{ij}^{(k)}$ и

$$(21.19)\qquad L^{(k)} = \begin{bmatrix} & 0 & \\ & 0 & \\ & \vdots & \\ 0 & m_{k+1,\,k} & 0 \\ & m_{k+2,\,k} & \\ & \vdots & \\ & m_{nk} & \end{bmatrix},$$

то уравнение

$$(21.20)\qquad A^{(k+1)} = A^{(k)} - L^{(k)}A^{(k)} + E^{(k)}$$

полностью описывает один шаг разложения с учетом погрешности округления. Складывая эти уравнения для $k=1, \ldots$

$\ldots, n-1$, получаем

$$L^{(1)}A^{(1)}+L^{(2)}A^{(2)}+\ldots+L^{(n-1)}A^{(n-1)}+A^{(n)}=$$
$$=A^{(1)}+E^{(1)}+E^{(2)}+\ldots+E^{(n-1)}.$$

Матрица $L^{(k)}A^{(k)}$ зависит только от k-й строки $A^{(k)}$, а эта строка равна k-й строке матрицы $A^{(n)}$. Таким образом, мы имеем

$$\left(L^{(1)}+\ldots+L^{(n-1)}+I\right)A^{(n)}=A^{(1)}+E^{(1)}+\ldots+E^{(n-1)},$$

т. е.

$$LU=A+E, \tag{21.21}$$

где L и U определены в (21.10) и (21.11) и где

$$E=E^{(1)}+E^{(2)}+\ldots+E^{(n-1)} \tag{21.22}$$

есть сумма ошибок отдельных шагов. Мы получили первый результат — доказательство формулы (21.21).

Наша следующая задача — нахождение оценки для матрицы E. Для этого нам требуется оценить границы чисел m_{ik} и $a_{ij}^{(k)}$, содержащихся в $\varepsilon_{ij}^{(k)}$. Стратегия упорядочивания применяется именно для того, чтобы эти границы были достаточно малы. Мы предположим, что строки матрицы A упорядочены так, что диагональный элемент $a_{kk}^{(k)}$ является максимальным по абсолютной величине среди элементов $a_{ik}^{(k)}$ для $i=k, \ldots, n$. Так как $m_{ik}=\mathrm{fl}\left(a_{ik}^{(k)}/a_{kk}^{(k)}\right)$, то

$$|m_{ik}|\leqslant 1 \text{ для всех } i,\ k. \tag{21.23}$$

Оценка для $a_{ij}^{(k)}$ более сложна. Зададимся вопросом: как велики могут быть числа, возникающие в процессе гауссовского исключения? Относительно легко показать для частичного упорядочивания, что если $|a_{ij}|\leqslant 1$, то $|a_{ij}^{(k)}|\leqslant 2^{k-1}$, и Уилкинсон [31], [32] приводит пример, в котором эта граница действительно достигается. Однако он указывает, что «на практике такой рост является очень редким». Действительно, он наблюдал, что обычно $|a_{ij}^{(k)}|\leqslant 8$ для всех k.

Определим

$$\rho=\max_{i,\,j,\,k}|a_{ij}^{(k)}|/\|A\|_\infty. \tag{21.24}$$

Хотя мы и не имеем хорошей априорной оценки для ρ, она относительно легко может быть вычислена для любой заданной матрицы в процессе треугольного разложения. С по-

мощью этого определения получаем, что

$$|a_{ij}^{(k)}| \leqslant \rho \| A \|_\infty.$$

Это вместе с (21.18) и (21.23) дает

$$(21.25) \qquad |\varepsilon_{ij}^{(k)}| \leqslant \rho \| A \|_\infty \cdot \begin{cases} u & \text{для } i \geqslant k+1,\ j=k, \\ 2u & \text{для } i \geqslant k+1,\ j \geqslant k+1, \\ 0 & \text{для других } i,\ j. \end{cases}$$

Следуя (21.22), объединим $\varepsilon_{ij}^{(k)}$ для получения E. Тогда

$$(21.26) \qquad E \leqslant \rho \| A \|_\infty u \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ 1 & 2 & 2 & \dots & 2 & 2 \\ 1 & 3 & 4 & \dots & 4 & 4 \\ . & . & . & \dots & . & . \\ 1 & 3 & 5 & \dots & 2n-4 & 2n-4 \\ 1 & 3 & 5 & \dots & 2n-3 & 2n-2 \end{bmatrix},$$

где неравенство выполняется по-элементно. Матрица в правой части легко восстанавливается, поскольку ее элементы представляют собой просто число арифметических операций, необходимых для вычисления соответствующего элемента в таблице LU.

Наконец, используя определение (21.3) для $\|E\|_\infty$, мы находим, что

$$\| E \|_\infty \leqslant \rho \| A \|_\infty \left(\sum_{j=1}^{n} (2j-1) - 1 \right) u \leqslant n^2 \rho \| A \|_\infty u.$$

Вот один из основных результатов данного раздела:

(21.27) Теорема. *Матрицы L и U, вычисленные методом исключения Гаусса с выбором главного элемента в системе с плавающей запятой с единичной ошибкой округления u, удовлетворяют равенству*

$$LU = A + E,$$

где

$$(21.28) \qquad \|E\|_\infty \leqslant n^2 \rho \| A \|_\infty u.$$

Другими словами, L и U образуют *точное* разложение некоторой слегка возмущенной матрицы. Если ρ не очень

велико, возмущения имеют порядок ошибок округления элементов A.

Имея L и U, мы находим решение системы $Ax=b$ последовательным решением двух треугольных систем $Ly=b$ и $Ux=y$. Мы поэтому хотим оценить ошибку округления, возникающую при решении общей треугольной системы, например

$$Rx=b.$$

Если R — нижняя треугольная матрица, мы вычисляем компоненты x по порядку следующим образом:

$$(21.29)\qquad \begin{aligned} x_1 &:= \mathrm{fl}\,(b_1/r_{11}),\\ x_i &:= \mathrm{fl}\left(\frac{-r_{i1}x_1-r_{i2}x_2-\ldots-r_{i,\,i-1}x_{i-1}+b_i}{r_{ii}}\right),\ i=2,\ldots,n. \end{aligned}$$

Мы можем использовать накапливающееся скалярное произведение и получить очень точный результат. Но, поскольку мы уже имеем некоторую ошибку в L и U, увеличение машинного времени едва ли оправдано. Таким образом, слегка обобщая теорему (20.18) для ошибки округления в обычном скалярном произведении, мы получаем, что

$$(21.30)\qquad \begin{aligned} x_1 &= \frac{b_1}{r_{11}(1+\delta_{11})},\\ x_i &= \frac{-r_{i1}(1+\delta_{i1})x_1-\ldots-r_{i,\,i-1}(1+\delta_{i,\,i-1})x_{i-1}+b_i}{r_{ii}(1+\delta_{ii})(1+\delta'_{ii})}\\ &\qquad\qquad (i=2,\ldots,n), \end{aligned}$$

где

$$(21.31)\qquad \begin{aligned} &|\delta_{ii}|\leqslant u,\ |\delta'_{ii}|\leqslant u, && i=1,\ldots,n,\\ &|\delta_{i1}|\leqslant(i-1)\cdot 1{,}01u, && i=2,\ldots,n,\\ &|\delta_{ij}|\leqslant(i+1-j)\cdot 1{,}01u, && i=2,\ldots,n,\ j=2,\ldots,i-1. \end{aligned}$$

Заметим, что как (20.8), так и (20.28) используются таким образом, что все множители $1+\delta$ умножаются на элементы R.

(21.32) Упражнение. Провести детальное доказательство (21.30) и (21.31).

Уравнения (21.30) могут быть записаны в виде

$$(21.33)\qquad \begin{aligned} &r_{11}(1+\delta_{11})x_1=b_1,\\ &r_{i1}(1+\delta_{i1})x_1+\ldots+r_{ii}(1+\delta_{ii})(1+\delta'_{ii})x_i=b_i\\ &\qquad\qquad (i=2,\ldots,n), \end{aligned}$$

или

$$(R+\delta R)\,x=b, \tag{21.34}$$

где

$$|\delta R| \leqslant \tag{21.35}$$

$$\leqslant 1{,}01u \begin{bmatrix} |r_{11}| & & & & \\ |r_{21}| & 2|r_{22}| & & & \\ 2|r_{31}| & 2|r_{32}| & 2|r_{33}| & & \\ 3|r_{41}| & 3|r_{42}| & 2|r_{43}| & 2|r_{44}| & \\ \vdots & \vdots & \vdots & \vdots & \\ (n-1)|r_{n1}| & (n-1)|r_{n2}| & (n-2)|r_{n3}| & (n-3)|r_{n4}| & \ldots 2|r_{nn}| \end{bmatrix}.$$

Следовательно,

$$\|\delta R\|_\infty \leqslant \frac{n(n+1)}{2}\cdot 1{,}01u\cdot \max_{i,j}|r_{ij}|. \tag{21.36}$$

В итоге мы получаем:

(21.37) Теорема. *Вектор x, вычисленный с помощью формул* (21.29), *является точным решением возмущенной треугольной системы* $(R+\delta R)x=b$, *где возмущение* δR *удовлетворяет неравенствам* (21.35) *и* (21.36).

Итак, возмущения опять имеют порядок ошибок округления. Однако, сравнивая (21.35) с (21.26), мы видим, что элементы матрицы δR пропорциональны соответствующим элементам матрицы R, но что элементы E и A не имеют такой прямой связи.

Применяя (21.37) к двум треугольным системам $Ly=b$ и $Ux=y$, мы находим, что окончательно вычисленное x удовлетворяет уравнению

$$(L+\delta L)(U+\delta U)\,x=b,$$

или, так как $LU=A+E$,

$$(A+E+(\delta L)\,U+L\delta U+\delta L\,\delta U)\,x=b. \tag{21.38}$$

Матрицы L и U удовлетворяют неравенствам

$$|l_{ij}|\leqslant 1,$$
$$|u_{ij}|\leqslant \rho\|A\|_\infty,$$

где ρ определено в (21.24). Следовательно,

$$(21.39)\qquad \begin{cases} \|L\|_\infty \leqslant n, \\ \|U\|_\infty \leqslant n\rho\|A\|_\infty, \\ \|\delta L\|_\infty \leqslant \frac{n(n+1)}{2}\cdot 1{,}01u, \\ \|\delta U\|_\infty \leqslant \frac{n(n+1)}{2}\cdot 1{,}01\rho\|A\|_\infty u. \end{cases}$$

В теореме (21.27) установлено, что

$$\|E\|_\infty \leqslant n^2\rho\|A\|_\infty u.$$

Так как в любых предполагаемых приложениях $n^2u \ll 1$, мы можем написать

$$\|\delta L\|_\infty \cdot \|\delta U\|_\infty \leqslant n^2\rho\|A\|_\infty u.$$

Наконец, если

$$(21.40)\qquad \delta A = E + (\delta L)U + L\,\delta U + \delta L\,\delta U,$$

то

$$\|\delta A\|_\infty \leqslant \|E\|_\infty + \|\delta L\|_\infty \cdot \|U\|_\infty + \|L\|_\infty \cdot \|\delta U\|_\infty + \|\delta L\|_\infty \cdot \|\delta U\|_\infty$$

и мы уже оценили все слагаемые в правой части. Эти оценки в сочетании с (21.38) приводят нас к теореме:

(21.41) Теорема. *Решение x, вычисленное методом исключения Гаусса с выбором главного элемента, удовлетворяет уравнению*

$$(21.1)\qquad (A+\delta A)x = b,$$

где δA определяется в (21.40). Более того,

$$(21.42)\qquad \|\delta A\|_\infty \leqslant 1{,}01(n^3+3n^2)\rho\|A\|_\infty u.$$

Именно это мы и собирались показать: вектор x, вычисленный методом исключения Гаусса с выбором главного элемента, точно удовлетворяет системе уравнений с возмущенной матрицей $A+\delta A$ и, более того, норма возмущения не превосходит некоторого вычислимого числа, содержащего в качестве сомножителя произведение единичной ошибки округления на норму исходной матрицы.

Заметим, что возмущение δA зависит от b (потому что δU и δL зависят от b), но что оценка для $\|\delta A\|_\infty$ не содержит b. Отметим также, что для того, чтобы в формуле (21.42)

выполнялось хотя бы приближенное равенство, на каждом шаге вычислений должна иметь место «наихудшая» ошибка округления, более того, должны выполняться уравнения, аналогичные $\|L\delta U\|_\infty = \|L\|_\infty \cdot \|\delta U\|_\infty$. Мы не знаем примеров, где $\|\delta A\|_\infty$ хотя бы приближалось к данной оценке. Уилкинсон [32] утверждает (стр. 108), что $\|\delta A\|_\infty$ редко бывает больше чем $nu\|A\|_\infty$.

Теорему (21.41) можно использовать для получения оценок других критериев ошибки, таких, как невязка $b - Ax$ и $x - A^{-1}b$. Мы оставляем это в качестве упражнений.

(21.43) Упражнение. Получить (21.42) из (21.39) и (21.40).

(21.44) Упражнение. Использовать (21.41) для получения верхней оценки относительной невязки

$$\| b - Ax \|_\infty / \| x \|_\infty.$$

Оценка должна зависеть от n, ρ, u, $\|A\|_\infty$.

(21.45) Упражнение. Использовать (21.41) для оценки относительной ошибки

$$\| x - x^* \|_\infty / \| x \|_\infty.$$

Здесь $x^* = A^{-1}b$ есть точное решение задачи. Оценка должна зависеть от n, ρ, u и числа ∞-обусловленности матрицы A, определяемого по формуле $\mathrm{cond}_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty$.

(21.46) Упражнение. Что вы можете сказать, с учетом (21.45), относительно выражения

$$\| x - x^* \|_\infty / \| x^* \|_\infty?$$

(Это более трудная задача.)

(21.47) Упражнение. Невязку $r = b - Ax$ можно эффективно вычислить (используя накапливающееся скалярное произведение). Оцените

$$\| x - x^* \|_\infty / \| x^* \|_\infty$$

в терминах $\|r\|_\infty$, $\|b\|_\infty$ и $\mathrm{cond}_\infty(A)$. Уравнение (21.1) не используется. Это есть апостериорная оценка ошибки, поскольку она требует знания вычисленного решения.

Основные оценки ошибок округления (21.27) для треугольного разложения матрицы A были получены при условии применения стратегии частичного упорядочивания. Однако поскольку неравенства (21.23) выполняются как для частичного, так и для полного упорядочивания, теоремы (21.27) и (21.41) справедливы для любого случая. Величина ρ, конечно, зависит от используемого алгоритма, в том числе от типа упорядочивания.

Предположим, что все $|a_{ij}| \leqslant 1$. Существует трудный и нерешенный вопрос относительно того, как велики могут быть числа $|a_{ij}^{(k)}|$ при исключении с полным упорядочиванием. Пусть

$$g(n) = \max |a_{ij}^{(k)}|, \text{ где все } |a_{ij}| \leqslant 1. \tag{21.48}$$

Уилкинсон [31] доказал, что

$$g(n) \leqslant 1{,}8 n^{(1/4)\ln n} \quad \text{для всех } n. \tag{21.49}$$

Оценка (21.49), конечно, много точнее, чем наилучшая оценка 2^{n-1}, которая может быть дана при частичном упорядочивании. Как мы установили ранее, оценка 2^{n-1} может действительно достигаться при частичном упорядочивании. С другой стороны, наибольшая известная величина $g(n)$ есть только n и она достигается для некоторых матриц произвольно высокого порядка. Разумно предположить, что $g(n) \leqslant n$ для всех вещественных матриц, и эта оценка, конечно, намного точнее, чем (21.49). Торнхейм [30] указал примеры комплексных матриц A, для которых $g(n) \simeq 1{,}05n$ для сколь угодно высокого порядка n.

22. СХОДИМОСТЬ ИТЕРАЦИОННОГО УТОЧНЕНИЯ

Если точность решения, вычисленного методом исключения Гаусса, по некоторым причинам нас не удовлетворяет, то в большинстве случаев очень точное решение при небольшом количестве дополнительной работы может дать алгоритм итерационного уточнения, введенный в разд. 13. В этом разделе мы рассмотрим итерационное уточнение более подробно и выведем достаточные условия его сходимости.

Пусть A и b — заданные матрица и правая часть соответственно. Пусть x_1 — вектор приближенного решения, полученный методом исключения Гаусса. Тогда для $m=1, 2, \ldots$ уточненное решение x_{m+1} получается следующим образом. Сначала вычисляется вектор невязки

$$r_m = b - Ax_m. \tag{22.1}$$

Затем решение системы

$$Ad_m = r_m \tag{22.2}$$

с использованием LU-разложения матрицы A, найденного при вычислении x_1, дает вектор поправки. Наконец, поправка прибавляется к x_m, что дает уточненное решение

$$x_{m+1} = x_m + d_m. \tag{22.3}$$

Эти три шага повторяются до тех пор, пока, как мы надеемся, x_m не достигнет желаемой точности. Полный анализ этого процесса довольно сложен, причем важные аспекты могут потеряться в деталях. Поэтому мы выбираем для исследования идеализированную модель: *мы предполагаем, что ошибка округления возникает только при вычислении поправок по невязкам.* Другими словами, мы предполагаем, что уравнения (22.1) и (22.3) выполняются точно и только (22.2) подвержено влиянию ошибок округления. Это предположение является обоснованным, если для решения уравнения (22.1) используются накапливающиеся скалярные произведения. Полный анализ для вычислений с плавающей запятой без упрощающих предположений был дан Молером [20].

Решение системы (22.2) есть в точности тот процесс, который мы изучили в предыдущей главе. Там мы нашли, что из-за ошибок округления вместо уравнения (22.2) следует рассматривать уравнение

$$(A + \delta A)\, d_m = r_m, \tag{22.4}$$

где матрица δA зависит от r_m, но ее оценка от r_m не зависит. Для того чтобы выявить зависимость от m и упростить последнее выражение, мы перепишем (22.4) в виде

$$A(I+F_m)\,d_m = r_m, \tag{22.5}$$

где

$$F_m = A^{-1}(\delta A). \tag{22.6}$$

Можно ожидать, что матрицы F_m полностью определяют поведение этой идеализированной модели итерационного уточнения.

(22.7) Теорема. *Пусть вектор x_m определен как выше. Пусть x^* — истинное решение линейной системы, т. е. $x^* = A^{-1}b$. Предположим, что для некоторой нормы из разд. 2*

$$\|F_m\| \leqslant \sigma < \frac{1}{2} \quad \text{для всех } m.$$

Тогда приближенное решение сходится к истинному, т. е.

$$\|x_m - x^*\| \to 0 \quad \text{при } m \to \infty.$$

Объединяя формулы (22.1) и (22.5), мы получаем, что

$$A(I+F_m)\,d_m = b - Ax_m.$$

Так как A — невырожденная матрица, то

$$(I+F_m)\,d_m = x^* - x_m.$$

Следовательно, из (22.3) вытекает, что

$$(I+F_m)\,x_{m+1} = F_m x_m + x^*.$$

Вычитая $(I+F_m)x^*$ из обеих частей равенства, мы получаем соотношение

$$(I+F_m)(x_{m+1} - x^*) = F_m(x_m - x^*). \tag{22.8}$$

В дальнейшем нам потребуется следующая лемма:

(22.9) Лемма. *Предположим, что F — такая матрица, что $\|F\| < 1$ для некоторой нормы из разд. 2. Тогда*

(a) $I+F$ *невырожденна*

и

(b) $\|(I+F)^{-1}\| \leqslant 1/(1-\|F\|)$.

Для доказательства п. (a) предположим, что $I+F$ является вырожденной матрицей. Тогда существует некоторый нену-

левой вектор x, такой, что

$$(I+F)x=\theta.$$

Это приводит к равенству

$$\|Fx\|=\|x\|,$$

которое противоречит предположению, что $\|F\|<1$.

Для доказательства (b) положим $G=(I+F)^{-1}$. Тогда $G=I-FG$, так что

$$\|G\|\leqslant\|I\|+\|F\|\cdot\|G\|$$

и, так как $\|I\|=1$,

$$\|G\|\leqslant 1/(1-\|F\|).$$

Это завершает доказательство леммы (22.9).

Возвращаясь к доказательству теоремы (22.7), мы заметим, что предположения $\|F_m\|\leqslant\frac{1}{2}$ более чем достаточно для применения леммы к матрицам F_m. Итак, из соотношения (22.8) следует

$$x_{m+1}-x^*=(I+F_m)^{-1}F_m(x_m-x^*). \tag{22.10}$$

Это уравнение является ключевым в данном разделе. Оно показывает, что ошибка на $(m+1)$-м шаге итерационного процесса есть произведение некоторой (неизвестной) матрицы на ошибку m-го шага. Если не сделать предположения о том, что уравнения (22.1) и (22.3) удовлетворяются точно, то мы имели бы в (22.10) два добавочных члена. (Предположение, что все уравнения (22.1), (22.2) и (22.3) удовлетворяются точно, означало бы, что F_m равно Θ, а x_{m+1} равно x^*.)

Применяя п. (b) леммы (22.9) к (22.10), мы получаем

$$\|x_{m+1}-x^*\|\leqslant\|F_m\|(1+\|F\|)^{-1}\|x_m-x^*\|\leqslant \leqslant\sigma(1-\sigma)^{-1}\|x_m-x^*\|. \tag{22.11}$$

Если $\tau=\sigma/(1-\sigma)$, то по индукции

$$\|x_m-x^*\|\leqslant\tau^{m-1}\|x_1-x^*\|.$$

Мы можем пойти несколько дальше. Полагая $x_0=\theta$, мы можем сказать, что x_1 определяется уравнением (22.3) при $m=0$ Таким образом,

$$\|x_1-x^*\|\leqslant\tau\|x^*\|$$

и следовательно,

$$\|x_m-x^*\|\leqslant\tau^m\|x^*\|. \tag{22.12}$$

Утверждение теоремы теперь следует из неравенства (22.12), поскольку условие $\sigma < 1/2$ обеспечивает $\tau < 1$.

Так как матрицы F_m почти невозможно определить, обычно непрактично проверять, удовлетворяются ли в данной ситуации предположения теоремы (22.7) или нет. Следующий результат является более удобным.

(22.13) Следствие. *Предположим, что*

$$1{,}01\,(n^3+3n^2)\,\rho u\,\|A\|_\infty\,\|A^{-1}\|_\infty < \frac{1}{2}.$$

Тогда $\|x_m - x^*\|_\infty \to 0$ *при* $m \to \infty$.

(Определения ρ, u и $\|\ \|_\infty$ см. в (21.24), (20.9) и (21.3).)

Можно построить примеры, в которых предположения теоремы (22.7) выполняются, тогда как предположения следствия (22.13) — нет.

Доказательство (22.13) следует непосредственно из уравнений (21.42) и (22.6). Величина $\mathrm{cond}_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty$ является обусловленностью матрицы A в максимум-норме. Таким образом, следствие (22.13) является перефразировкой утверждения: «Итерационное уточнение сходится, если матрица не слишком плохо обусловлена».

Неравенство (22.12) дает больше, чем просто доказательство сходимости. Оно дает также оценку скорости сходимости. Ошибка на каждом шаге итерационного процесса не превосходит τ, умноженного на ошибку предыдущего шага. Например, предположим, что мы имеем вычислительную машину типа IBM 7090 с 27 двоичными разрядами и операциями с блокировкой округления. Тогда

$$u = 2^{-26} \simeq 10^{-8}.$$

Предположим, что мы имеем (20×20)-матрицу с числом обусловленности порядка 5×10^5. Для этой машины задача очень плохо обусловлена. Предположим далее, что

$$\|\delta A\|_\infty \simeq nu\,\|A\|_\infty,$$

так что

$$\|F_m\|_\infty \simeq nu\,\mathrm{cond}_\infty(A) \simeq 0{,}1$$

и, следовательно,

$$\tau \simeq \|F\|_\infty/(1-\|F\|_\infty) \simeq 0{,}1.$$

Тогда неравенство

$$\|x_1 - x^*\|_\infty \leqslant \tau\,\|x^*\|_\infty$$

показывает, что первое приближение имеет точность порядка величины одного десятичного знака, в то время как соотношение

$$\|x_8 - x^*\|_\infty \leqslant \tau^8\,\|x^*\|_\infty$$

показывает, что восьмое приближение имеет примерно такую точность, какая обеспечивается машиной.

Если мы разберем этот пример несколько внимательнее, мы обнаружим один существенный дефект нашего идеализированного анализа. Теорема (22.7) утверждает, что, проведя достаточное число итераций, можно сделать ошибку $\|x_m - x^*\|_\infty$ произвольно малой. Ясно, что это невозможно, поскольку компонеты вектора x_m должны быть числами в системе с плавающей запятой, выраженными с простой точностью, а компоненты вектора x^* в общем случае — нет. Легко убедиться, что эта трудность возникает из-за предположения, что уравнение (22.3) удовлетворяется точно. Практически мы делаем ошибки округления в каждой компоненте на этом шаге, так что мы никогда не получим x_m точнее, чем позволяет единичная ошибка округления. Более детальный анализ дает неравенство

$$\|x_m - x^*\|_\infty \leqslant \left(\tau^m + \frac{u}{1-\tau}\right) \cdot \|x^*\|_\infty \tag{22.14}$$

вместо (22.12). Мы видим, что ошибка не стремится к нулю, а только становится малой.

Наконец, мы снова подчеркиваем необходимость точного вычисления невязок. Мы предположили, что уравнение (22.1) является точным. Это предположение допустимо, если мы используем для вычисления невязок скалярные произведения повышенной точности. Однако если используется обычная арифметика с плавающей запятой, то почти вся точность будет потеряна при вычитании $b_i - \sum_{j=1}^{n} a_{ij}x_j$, и вычисленные невязки будут иметь только отдаленную связь с их истинной величиной.

(22.15) Упражнение. Уточнить (22.3), включив член, который учитывает ошибки округления. Пересмотреть доказательство теоремы (22.7), принимая во внимание этот дополнительный член.

(22.16) Упражнение. Предположим, что A является квадратной матрицей. Предположим, что приближенная обратная матрица X имеет свойства

$$\|AX - I\| \leqslant 0{,}01,$$
$$\|X\| = 200.$$

Дать строгую достаточно точную верхнюю оценку для $\|X - A^{-1}\|$. (Ответ: 2,03.)

23. ПОЛОЖИТЕЛЬНО ОПРЕДЕЛЕННЫЕ МАТРИЦЫ; ЛЕНТОЧНЫЕ МАТРИЦЫ

Некоторые специальные типы матриц так часто встречаются во многих задачах, что целесообразно использовать их частные свойства. В этом разделе мы рассмотрим два типа — положительно определенные матрицы и ленточные матрицы.

В разд. 9 мы пришли к заключению, что любая положительно определенная матрица A имеет единственное разложение вида

$$A = GG^T, \tag{23.1}$$

где G является нижней треугольной матрицей с положительными диагональными элементами. В частности, из положительной определенности матрицы следует невырожденность ее главных миноров, необходимая для LU-теоремы. Расписав равенство (23.1) по-элементно, мы видим, что если a_{jj} находится на главной диагонали A, то

$$a_{jj} = g_{j1}^2 + g_{j2}^2 + \ldots g_{jj}^2, \tag{23.2}$$

а если a_{ij} лежит ниже диагонали, то

$$a_{ij} = g_{i1}g_{j1} + g_{i2}g_{j2} + \ldots + g_{ij}g_{jj} \qquad (j < i). \tag{23.3}$$

Эти уравнения, взятые в соответствующем порядке, определяют все элементы матрицы G. Один из типичных порядков таков:

$$g_{11},\ g_{21},\ \ldots,\ g_{n1},\ g_{22},\ g_{32},\ \ldots,\ g_{n2},\ g_{33},\ \ldots,\ g_{nn}.$$

Вначале из уравнения (23.2) находится диагональный элемент, а затем используется (23.3) для нахождения внедиагональных элементов того же самого столбца. Это порождает алгоритм

$$\begin{aligned}
&\textbf{for } j := 1 \textbf{ step } 1 \textbf{ until } n \textbf{ do}\\
&\textbf{begin } g_{jj} := sqrt\left(a_{jj} - \sum_{k=1}^{j-1} g_{jk}^2\right);\\
&\qquad \textbf{for } i := j+1 \textbf{ step } 1 \textbf{ until } n \textbf{ do}\\
&\qquad\qquad g_{ij} := \left(a_{ij} - \sum_{k=1}^{j-1} g_{ik}g_{jk}\right)\Big/ g_{jj};\\
&\textbf{end.}
\end{aligned} \tag{23.4}$$

Отметим, что элементы матрицы A, расположенные выше диагонали, ввиду симметрии не рассматриваются.

Этот алгоритм представляет собой *метод Холесского* или *метод квадратного корня* для разложения положительно определенной матрицы (см. Фаддеев и Фаддеева [34]). Одно из

преимуществ метода состоит в том, что нет необходимости в выборе главного элемента. Для матриц, которые не являются положительно определенными, выбор главного элемента необходим, так как мы делим на эти элементы и должны быть уверены, что элементы образуемой матрицы не слишком велики. Большие элементы привели бы к большим ошибкам округления и потере точности. Однако в методе Холесского для положительно определенных матриц непосредственно из уравнения (23.2) видно, что

$$|g_{ij}| \leqslant \sqrt{a_{ii}} \tag{23.5}$$

для любых i, j. Иными словами, элементы матрицы G ограничены даже при отсутствии выбора главного элемента.

Использование аналогичного метода для матриц, не являющихся положительно определенными, невозможно без усложнения вычислений, поскольку разложение (23.1) в общем случае не имеет места. Даже для симметричной матрицы A алгоритм может прерваться из-за деления на нуль или извлечения квадратного корня из отрицательного числа. (Если бы матрица A была положительно определенной, но очень плохо обусловленной, то ошибки округления могли бы нарушить определенность. Это также привело бы к извлечению квадратного корня из отрицательных чисел и необходимости прерывания алгоритма.)

Вследствие симметрии матрицы A необходимо запоминать только $\frac{1}{2}n(n+1)$, т. е. немного больше половины ее элементов, что дает важную экономию машинной памяти для больших матриц. Однако чтобы этой экономией воспользоваться, обычно требуются относительно сложные индексные схемы, приводящие к потере времени. В разд. 6 мы определили *ленточные матрицы* с шириной ленты $2m+1$ как матрицы, для которых $a_{ij}=0$, если $|i-j|>m$. Например, если $m=1$, то ширина ленты равна трем и матрица имеет следующий вид:

$$\begin{bmatrix} a_{11} & a_{12} & & & & & 0 \\ a_{21} & a_{22} & a_{23} & & & & \\ & a_{32} & a_{33} & a_{34} & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \\ & & & & & \cdot & a_{n,\,n-1} \\ 0 & & & & & a_{n-1,\,n} & a_{nn} \end{bmatrix}. \tag{23.6}$$

Такие матрицы называются также *тридиагональными*.

Понятие ленточной матрицы полезно только тогда, когда m значительно меньше n. Собственно говоря, любая матрица является ленточной с $m=n-1$. (Матрицы со сравнительно малым числом нулевых элементов называются плотными матрицами в отличие от ленточных или редких матриц; см. разд. 6.) Ширина ленты зависит от упорядочивания неизвестных и уравнений в системе. Например, перестановка второй и третьей строк (уравнений) в матрице (23.6) увеличивает ширину ленты с трех до пяти.

Используя преимущество ленточной структуры матрицы при решении линейной системы, можно сэкономить как время, так и память. Память экономится благодаря тому, что только ненулевую часть матрицы требуется хранить в машине. Для этого требуется $(2m+1)n$ ячеек, вместо n^2 в случае полной матрицы. Таким образом, если m меньше, чем $(n-1)/2$, то память экономится. Ненулевую часть ленточной матрицы можно расположить в памяти машины в виде обычного прямоугольного массива, полагая

$$c_{ij}=a_{i,\,i+j},\quad i=1,\ldots,n,\quad j=-m,\ldots,0,\ldots,m. \tag{23.7}$$

Массив C займет тогда $(2m+1)n$ ячеек. (Элементы c_{ij} при $i+j<0$ или $i+j>n$ не определены, но все же занимают ячейки памяти.) Для очень маленьких m может оказаться удобным использовать вместо этого $2m+1$ векторов. Ленту матрицы (23.6) можно запомнить в трех векторах — например d, e и f — следующим образом:

$$\begin{bmatrix} e_1 & f_1 & & & & & & 0 \\ d_2 & e_2 & f_2 & & & & & \\ & d_3 & e_3 & f_3 & & & & \\ & & \cdot & \cdot & \cdot & & & \\ & & & \cdot & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \cdot & \\ & & & & & \cdot & \cdot & f_{n-1} \\ 0 & & & & & & d_n & e_n \end{bmatrix}. \tag{23.8}$$

Компактное хранение матрицы не было бы очень полезным, если бы оно не оставалось таким в процессе решения системы уравнений. К счастью, справедлива следующая теорема:

(23.9) Теорема. *Если ленточная матрица с шириной ленты $2m+1$ имеет LU-разложение, то $L=(l_{ij})$ и $U=(u_{ij})$ яв-*

ляются треугольными ленточными матрицами, т. е.

$$l_{ij} \neq 0 \quad \text{только для} \quad i = j - m, \ldots, j,$$
$$u_{ij} \neq 0 \quad \text{только для} \quad i = j, \ldots, j + m.$$

Доказательство следует из теоремы (9.2), так что мы его опускаем. Когда $m=1$, матрицы L и U имеют вид

$$L = \begin{bmatrix} 1 & & & & & & 0 \\ l_{21} & 1 & & & & & \\ & l_{31} & 1 & & & & \\ & & \cdot & \cdot & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ 0 & & & & & l_{n,n-1} & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} u_{11} & u_{12} & & & & & 0 \\ & u_{22} & u_{23} & & & & \\ & & u_{33} & u_{34} & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & \cdot & \cdot \\ 0 & & & & & & u_{n-1,n} \\ & & & & & & u_{nn} \end{bmatrix}.$$

В теореме (23.9) мы предполагаем, что LU-разложение существует. Конечно, это не всегда так, и, возможно, потребуется изменение порядка уравнений (упорядочивание), чтобы быть уверенными в существовании и допустимой точности L и U. Перестановка уравнений будет обычно увеличивать ширину ленты, но не более, чем вдвое. Матрицы L и U все еще будут иметь ленточную структуру, но с шириной, большей m.

Для некоторых ленточных матриц, встречающихся на практике, упорядочивание не нужно. Матрицы, полученные при конечно-разностной аппроксимации дифференциальных уравнений, например, часто являются неразложимыми (см. определение (6.4)) и имеют диагональное преобладание (см. определение (6.1)). Можно показать, что упорядочивания для таких матриц не требуется.

(23.10) Упражнение. Доказать, что упорядочивание не является обязательным для неразложимых матриц с диагональным преобладанием. См. Вендрофф [8].

Использование ленточной структуры L и U экономит также и время. Цель гауссовского исключения состоит в сокращении числа неизвестных в каждом уравнении до тех пор, пока уравнения можно будет разрешить непосредственно. Для ленточных матриц число неизвестных в каждом уравнении с самого начала мало, и, следовательно, сведение к треугольной форме требует меньше времени. Действительно, разложение требует только mn^2 умножений вместо $n^3/3$ для полной матрицы. Таким образом, (100×100)-матрица с шириной ленты 11 требует менее одного процента времени, необходимого для полной (100×100)-матрицы.

Программа разложения произвольной ленточной матрицы (т. е. матрицы, у которой ширина ленты является параметром) во многом такая же, как и программа для полной матрицы. Отличие состоит только в области изменения индексов. Действительно, процедура *DECOMPOSE* будет для ленточных матриц достаточно эффективной, поскольку перед внутренним циклом проводится проверка на равенство сомножителей нулю. Но она не будет столь эффективна, как программа, которая «знает наперед», что сомножители вне ленты должны быть нулями.

Программа же без упорядочивания, написанная для фиксированной ленты небольшой ширины — три или пять, например, — является чрезвычайно простой. Для ленты ширины три, элементы которой хранятся в соответствии с (23.8), алгоритм имеет вид:

(23.11)
```
u_1 := e_1;
for i := 2 step 1 until n do
begin m_i := d_i/u_{i-1};
    u_i := e_i - m_i × f_{i-1};
end.
```

Матрицы L и U при этом таковы:

$$L=\begin{bmatrix} 1 & & & & & 0 \\ m_2 & 1 & & & & \\ & m_3 & 1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ 0 & & & & \cdot & \cdot \\ & & & & m_n & 1 \end{bmatrix},$$

$$U = \begin{bmatrix} u_1 & f_1 & & & & & 0 \\ & u_2 & f_2 & & & & \\ & & u_3 & f_3 & & & \\ & & & \cdot & \cdot & & \\ & & & & \cdot & \cdot & \\ & & & & & \cdot & \cdot \\ 0 & & & & & & f_{n-1} \\ & & & & & & u_n \end{bmatrix}.$$

Последовательное решение системы $Ax = LUx = b$ также очень просто. Имеем

(23.12)
$$\begin{aligned} & x_1 := b_1; \\ & \textbf{for } i := 2 \textbf{ step } 1 \textbf{ until } n \textbf{ do} \\ & \qquad x_i := b_i - m_i \times x_{i-1}; \\ & x_n := x_n/u_n; \\ & \textbf{for } i := n-1 \textbf{ step } -1 \textbf{ until } 1 \textbf{ do} \\ & \qquad x_i := (x_i - f_i \times x_{i+1})/u_i. \end{aligned}$$

Алгоритмом (23.11) следует, однако, пользоваться с осторожностью, поскольку в нем отсутствует упорядочивание. (Некоторые u_i могут быть нулями.)

Из-за небольшого числа арифметических операций при решении системы уравнений с ленточной матрицей обычно не требуется использования итерационного уточнения. В этом случае можно сэкономить память, так как не нужно хранить матрицу A. В алгоритме (23.11) массивы m и u можно просто располагать на месте d и e.

Если производится упорядочивание, то при удвоении ширины ленты уже только матрица U потребует столько же памяти, сколько занимает матрица A. В этом случае либо должна быть предусмотрена дополнительная память для L, либо правые части нужно обрабатывать одновременно и не запоминать L.

(23.13) Упражнение. Написать программы *CHOLESKY DECOMPOSE* (n, A, G) и *CHOLESKY SOLVE* (n, G, b, x) взамен *DECOMPOSE* и *SOLVE* для положительно определенных матриц. Заметим, что *IMPROVE* не требует изменений.

(23.14) Упражнение. Доказать, что симметричные неразложимые матрицы с диагональным преобладанием и положительными диагональными элементами являются положительно определенными.

(23.15) Упражнение. Показать, что при использовании гауссовского исключения для положительно определенных матриц упорядочивание не необходимо.

(23.16) Упражнение. Написать программу $BAND(n, m, G, b, x)$ для решения системы $Ax = b$ с произвольной ленточной матрицей, хранящейся в соответствии с (23.7). Использовать упорядочивание без запоминания элементов L. Не пользоваться никакими массивами памяти, кроме предусмотренных параметрами.

24. ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ ЛИНЕЙНЫХ СИСТЕМ

Когда практические ситемы линейных уравнений имеют очень большой порядок, их матрицы, как правило, бывают редкими. Например, решение задачи Дирихле для уравнения Лапласа в частных производных можно аппроксимировать конечно-разностными уравнениями на сетке с числом узлов, равным 5000. Соответствующая матрица имеет порядок 5000 и, следовательно, содержит 25 000 000 элементов. Однако только около 25 000 из них отличны от нуля. Даже эти 25 000 ненулевых элементов нет необходимости хранить, так как их значения известны непосредственно из геометрии разностной сетки. Для таких матриц метод исключения Гаусса крайне неудобен, поскольку он многие из нулевых элементов переводит в ненулевые. Более того, он переводит простые ненулевые элементы в сложные, которые должны запоминаться. Таким образом, в приведенном выше примере математик, использующий метод исключения Гаусса, должен примириться с хранением более чем 100 000 матричных элементов.

В противоположность гауссовскому исключению существует ряд методов решения линейных систем $Ax=b$, которые используют только исходную матрицу A. Эти методы состоят из относительно простых алгоритмов для преобразования некоторого вектора $x^{(k)}$ в другой вектор $x^{(k+1)}$, зависящий от $x^{(k)}$, A и b. (В некоторых итерационных процессах $x^{(k+1)}$ может зависеть от других векторов $x^{(k-r)}$ при $r>0$, но мы здесь не рассматриваем такую возможность.) Подробное изложение итерационных методов составляет основное содержание многих книг, и мы можем дать здесь только их краткую характеристику.

Чтобы дать читателю представление об уравнениях, для которых используется итерационная техника, мы опишем задачу с дифференциальными уравнениями в частных производных, а затем заменим эту задачу простой конечно-разностной схемой. Для читателя не обязательно знать основы теории дифференциальных уравнений в частных производных.

По этому вопросу гораздо больше написано у Вазова и Форсайта [40].

Пусть R — открытый прямоугольник со сторонами длины 4 и 3,

$$R=\{(x,\ y):\ 0<x<4 \text{ и } 0<y<3\}.$$

Пусть $\overline{R}$ — замкнутый прямоугольник, являющийся замыканием R,

$$\overline{R}=\{(x,\ y):\ 0\leqslant x\leqslant 4 \text{ и } 0\leqslant y\leqslant 3\}.$$

Пусть B означает границу R, за исключением четырех углов. Мы рассматриваем B как объединение двух множеств T и S. Здесь T означает верхнюю сторону прямоугольника, а S — остальные, см. рис. (24.3). Тогда

$$\begin{aligned} T &= \{(x,\ y):\ 0<x<4 \text{ и } y=3\}, \\ S &= \{(x,\ y):\ 0<x<4 \text{ и } y=0, \\ &\qquad \text{или } x=0 \quad \text{и } 0<y<3, \\ &\qquad \text{или } x=4 \quad \text{и } 0<y<3\}; \\ B &= T \cup S. \end{aligned}$$

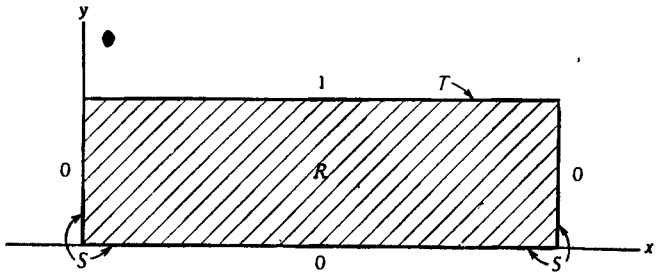
Задача состоит в отыскании функции $u=u(x, y)$, определенной на $\bar{R}$ и такой, что

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0 \text{ на } R, \tag{24.1}$$

с граничными условиями

$$\begin{aligned} u &= 1 \quad \text{на } T, \\ u &= 0 \quad \text{на } S. \end{aligned} \tag{24.2}$$

(24.3)



Задача (24.1, 24.2) называется задачей Дирихле для дифференциального уравнения Лапласа. Распределение тепла — наиболее частый источник таких задач. Пусть $\bar{R}$ представляет собой поперечное сечение очень длинного однородного металлического стержня, и пусть стержень имеет температуру 1° на всей поверхности, соответствующей T, и температуру 0° на всей поверхности, соответствующей S. Тогда установившееся распределение температуры внутри стержня будет удовлетворять системе уравнений (24.1, 24.2).

Существует много путей приближенного решения задачи типа (24.1, 24.2). Так называемый *метод конечных разностей* принадлежит к числу широко применимых методов. Мы мо-

жем рассматривать его как некоторое конечное приближение исходной задачи. Метод можно реализовать следующим образом. Пусть N — положительное целое число, и пусть $h=1/N$. Сначала мы заменяем R прямоугольным массивом R_h, состоящим из $(3N-1)(4N-1)$ точек:

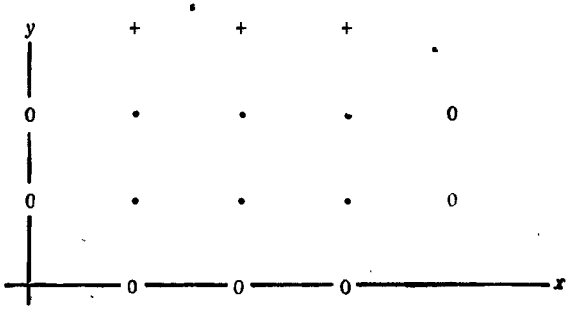
$$R_h=\left\{\left(\frac{i}{N},\ \frac{j}{N}\right):\ i=1,\ 2,\ \ldots,\ 4N-1 \text{ и } j=1,\ 2,\ \ldots,\ 3N-1\right\}.$$

Аналогично заменяем T линейным массивом из $4N-1$ точек:

$$T=\left\{\left(\frac{i}{N},\ 3\right):\quad i=1,\ 2,\ \ldots,\ 4N-1\right\}.$$

Пусть также S_h означает множество всех точек $(i/N,\ j/N)$, лежащих в S. Положим $\bar{R}_h=R_h\cup T_h\cup S_h$. См рис. (24.4) для $N=1$.

(24.4)



$N=1$

$+$ — точки T_h; $\bullet$ — точки R_h; 0 — точки S_h.

Теперь неизвестная функция u, определенная на $\bar{R}$, заменяется неизвестной функцией v, определенной на конечном точечном множестве $\bar{R}_h$. Дифференциальное уравнение в частных производных (24.1) заменяется системой алгебраических уравнений

$$\frac{v(x-h,\ y)-2v(x,\ y)+v(x+h,\ y)}{h^2}+$$
$$+\frac{v(x,\ y-h)-2v(x,\ y)+v(x,\ y+h)}{h^2}=0$$

для каждой точки (x, y) из R_h. Эти разделенные разности отлично аппроксимируют $\partial^2 v/\partial x^2$, $\partial^2 v/\partial y^2$ соответственно для

любой гладкой функции $v(x, y)$, определенной всюду в R. Граничные условия (24.2) можно заменить условиями $v=1$ на T_h и $v=0$ на S_h.

Таким образом, задача (24.1, 24.2) с дифференциальными уравнениями в частных производных переходит для каждого целого N в следующую алгебраическую задачу: найти v, определенное на $\bar{R}_h$ и такое, что

$$4v(x, y)-v(x, y+h)-v(x-h, y)-v(x, y-h)- \\ -v(x+h, y)=0 \text{ для каждого } (x, y) \text{ в } R_h \tag{24.5}$$

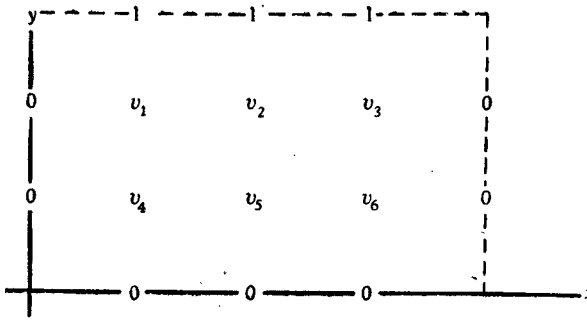
при

$$v(x, y)=1 \text{ на } T_h \text{ и } v(x, y)=0 \text{ на } S_h. \tag{24.6}$$

Современные вычислительные машины позволяют легко оперировать с сетками, имеющими свыше 15 000 точек. Следовательно, полагая $(3N+1)(4N+1)$ приблизительно равным 15 000, мы можем решать систему (24.5, 24.6) с $N=34$. Заметим, что в этой системе имеется $n=(3N-1)(4N-1)$ неизвестных значений функции. Таким образом, соответствующая системе (24.5, 24.6) матрица имеет порядок n и, следовательно, имеет $n^2=(3N-1)^2(4N-1)^2\simeq 144N^4$ элементов. Однако в каждой строке матрицы не более 5 элементов отличны от нуля. Таким образом, плотность матрицы равна приблизительно $5/(12N^2)$.

Для иллюстрации некоторых аспектов решения такой системы положим $N=1$, так что R_h состоит из шести точек, как показано на рис. (24.7), где также представлены соответствующими граничными значениями десять точек T_h и S_h.

(24.7)



Занумеровав шесть неизвестных значений v, как показано на рис. (24.7), мы можем систему уравнений (24.5, 24.6) записать

следующим образом:

$$
(24.8)\qquad
\begin{aligned}
4v_1 - v_2 \qquad\quad - v_4 \qquad\qquad &= 1,\\
-v_1 + 4v_2 - v_3 \qquad\quad - v_5 \qquad &= 1,\\
- v_2 + 4v_3 \qquad\qquad\quad - v_6 &= 1,\\
-v_1 \qquad\qquad + 4v_4 - v_5 \qquad &= 0,\\
- v_2 \qquad\quad - v_4 + 4v_5 - v_6 &= 0,\\
- v_3 \qquad\quad - v_5 + 4v_6 &= 0.
\end{aligned}
$$

Отметим, что 16 из 36 коэффициентов в уравнениях (24.8) равны нулю. Хотя систему (24.8) можно легко решить различными способами, в том числе методом исключения Гаусса, мы проиллюстрируем применение некоторых итерационных методов, которые могут быть использованы для много бо́льших значений N.

(24.9) Упражнение. Доказать, что система (24.8) имеет единственное решение. *Указание*: предположим, что существуют два различных решения w и y, и пусть $z=w-y$. Тогда z будет удовлетворять системе уравнений типа (24.8) с нулевой правой частью. Показать, что из этого будет следовать $z=\theta$. См. упражнение (23.14).

Во всяком итерационном методе при решении системы (24.5, 24.6) начинают с некоторого разумного или удобного множества значений v_i. Возьмем значения $v_1=v_3=0{,}400$, $v_2=0{,}600$, $v_4=v_6=0{,}100$, $v_5=0{,}300$. Имеем:

$$
\begin{array}{c|ccc|c}
 & 1 & 1 & 1 & \\
\hline
0 & 0{,}400 & 0{,}600 & 0{,}400 & 0\\
0 & 0{,}100 & 0{,}300 & 0{,}100 & 0\\
\hline
 & 0 & 0 & 0 &
\end{array}
$$

Мы теперь опишем процесс, известный под разными названиями: *метод Гаусса — Зейделя*, *метод Либмана* или *метод последовательных смещений*. Прежде всего зафиксируем порядок, в котором мы будем менять компоненты v_i. Будем здесь использовать порядок, в котором мы их уже занумеровали: v_1, v_2, v_3, v_4, v_5, v_6. Изменим теперь v_1 следующим образом: используя имеющиеся значения v_2, v_3, v_4, v_5, v_6 и значения на границе, пересчитываем значение $v_1=v\,(1,2)$ так, чтобы уравнение (24.5) удовлетворялось в точке $(x, y)=(1, 2)$. То есть на этой стадии определяем новое v_1 так, что

$$
(24.10)\qquad 4v_1 - 0 - v_4 - v_2 - 1 = 0,
$$

где v_2 и v_4 принимают начальные значения, а именно $v_2 = 0{,}600$, $v_4 = 0{,}100$. Разрешая (24.10) относительно v_1, получаем

$$v_1 := (1 + 0 + v_4 + v_2)/4. \tag{24.11}$$

Отметим, для определенности, что мы будем вычислять (24.11) и аналогичные формулы ниже в системе с плавающей запятой с тремя десятичными разрядами, производя вычисления слева направо. Тогда мы получаем из (24.11), что $v_1 = 0{,}425$, и этим значением немедленно заменяем старое значение v_1.

Аналогичным образом вычисляем v_2 так, чтобы оно удовлетворяло уравнению (24.5) в точке $(x, y) = (2, 2)$. Мы имеем

$$v_2 := (1 + v_1 + v_5 + v_3)/4,$$

т. е.

$$\begin{aligned} v_2 &:= (1{,}00 + 0{,}425 + 0{,}300 + 0{,}400)/4 = \\ &= (1{,}43 + 0{,}300 + 0{,}400)/4 = \\ &= (2{,}13)/4 = \\ &= 0{,}533. \end{aligned}$$

Вычисления проводим с тщательным округлением до трех значащих десятичных разрядов. Далее, вычисляем

$$\begin{aligned} v_3 &:= (1 + v_2 + v_6 + 0)/4 = \\ &= (1{,}00 + 0{,}533 + 0{,}100)/4 = \\ &= 0{,}408; \end{aligned}$$

$$\begin{aligned} v_4 &:= (v_1 + 0 + 0 + v_5)/4 = \\ &= (0{,}425 + 0{,}300)/4 = \\ &= 0{,}181; \end{aligned}$$

$$\begin{aligned} v_5 &:= (v_2 + v_4 + 0 + v_6)/4 = \\ &= (0{,}533 + 0{,}181 + 0{,}100)/4 = \\ &= 0{,}204; \end{aligned}$$

$$\begin{aligned} v_6 &:= (v_3 + v_5 + 0 + 0)/4 = \\ &= (0{,}408 + 0{,}204)/4 = \\ &= 0{,}153. \end{aligned}$$

Этим самым завершается один цикл итераций, в результате которого все v_i принимают новые значения. Слово «последовательный» в последовательных смещениях отражает тот факт, что каждое v_i замещается своим новым значением

перед тем, как будет вычислено следующее v_i. После первого цикла получается такая картина:

$$\begin{array}{c|ccc|c} & 1 & 1 & 1 & \\ \hline 0 & 0{,}425 & 0{,}533 & 0{,}408 & 0 \\ 0 & 0{,}181 & 0{,}204 & 0{,}153 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

Итерации продолжаются таким же образом для нескольких циклов. Мы приведем результаты каждого цикла:

$$\begin{array}{c|ccc|c} & 1 & 1 & 1 & \\ \hline 0 & 0{,}428 & 0{,}510 & 0{,}415 & 0 \\ 0 & 0{,}158 & 0{,}205 & 0{,}155 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

$$\begin{array}{c|ccc|c} & 1 & 1 & 1 & \\ \hline 0 & 0{,}418 & 0{,}513 & 0{,}418 & 0 \\ 0 & 0{,}156 & 0{,}206 & 0{,}156 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

$$\begin{array}{c|ccc|c} & 1 & 1 & 1 & \\ \hline 0 & 0{,}418 & 0{,}513 & 0{,}418 & 0 \\ 0 & 0{,}156 & 0{,}206 & 0{,}156 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

Заметим, что в четвертом цикле значения v_i не меняются. Следовательно, они не будут изменяться и дальше, и численные результаты сходятся к предельным значениям. При этом достигаемая точность является наилучшей для обычной арифметики с плавающей запятой с тремя десятичными разрядами.

Нетрудно видеть, что наш процесс имеет характер усреднения и оставляет каждую компоненту v_i между наибольшим и наименьшим начальными значениями v_i и граничными значениями, за исключением возможных небольших эффектов округления. Отсюда следует, что существует только конечное число векторов v, достижимых в данном итерационном процессе. Поэтому последовательность значений вектора v после ряда итерационных циклов должна, очевидно, либо достигать определенного фиксированного значения (как в приведенном выше примере), либо входить в периодический режим, в котором векторы повторяются после фиксированного числа циклов. Мы знаем, что возможно любое из двух окончаний

итерационного процесса, но у нас нет сведений о том, какое из них встречается чаще.

(24.12) Упражнение. Решить ту же задачу в арифметике с плавающей запятой с тремя десятичными разрядами при начальном векторе $v_1=v_2=v_3=v_4=v_5=v_6=0$. Показать, что в седьмом цикле получается тот же самый результат.

Тот факт, что в нашем случае итерации сошлись к фиксированному предельному вектору, не должен создавать у нас уверенность, что предел является в точности округленным решением задачи (24.8). Ошибки округления можно оценить, пользуясь методикой разд. 22 и 23, но мы поступим иначе.

Неискушенный в численном анализе читатель может предположить, что ошибка округления проявится самое большее в одной единице последнего разряда. Однако точное округленное решение нашей задачи с пятью десятичными разрядами имеет вид

$$\begin{array}{c|ccc|c} & 1 & -1 & 1 & \\ \hline 0 & 0{,}41615 & 0{,}50932 & 0{,}41615 & 0 \\ 0 & 0{,}15528 & 0{,}20497 & 0{,}15528 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

Заметим, что наибольшая ошибка предельного вектора с тремя десятичными разрядами есть 0,00368, что равно почти четырем единицам в последнем десятичном разряде. Как и в методе исключения Гаусса, ошибка в арифметике с тремя десятичными разрядами может быть сильно уменьшена за счет выполнения основных операций в числителях формулы (24.11) и аналогичных последующих формул с накоплением с двойной точностью с последующим округлением результата деления на четыре в число с двойной точностью.

Если бы это проделать, начиная с последней стадии предыдущих вычислений, мы получили бы сходимость к

$$\begin{array}{c|ccc|c} & 1 & 1 & 1 & \\ \hline 0 & 0{,}417 & 0{,}510 & 0{,}417 & 0 \\ 0 & 0{,}156 & 0{,}206 & 0{,}156 & 0 \\ \hline & 0 & 0 & 0 & \end{array}$$

за одну дополнительную итерацию. Отметим, что максимальная ошибка теперь равна 0,00103, что составляет менее трети максимальной ошибки без накопления с двойной точностью.

Полный анализ любого итерационного процесса при наличии округлений чрезвычайно труден и редко проделывается. Много легче изучение того же итерационного процесса при точных вычислениях без каких-либо округлений. Это дает очень полезную математическую модель поведения реального итерационного процесса при наличии округления, модель, которая в некотором смысле является отличным индикатором фактического поведения итераций с округлениями до тех пор, пока ошибка находится на уровне «шумов» округлений.

При отсутствии округлений описываемый нами итерационный процесс порождает бесконечную последовательность (обычно) меняющихся с каждым шагом векторов, которые мы обозначим $v^{(0)}, v^{(1)}, \ldots, v^{(k)}, v^{(k+1)}, \ldots$. Мы хотим исследовать условия, при которых $\{v^{(k)}\}$ является сходящейся последовательностью векторов.

Поучительно описать итерационный процесс в обозначениях системы уравнений (24.8). Пусть $v^{(k)} = (v_1^{(k)}, \ldots, v_6^{(k)})$ есть результат k-го цикла итерационного процесса. Читатель может легко показать, что вектор $v^{(k+1)}$ удовлетворяет системе уравнений

$$
(24.13)\quad \begin{cases}
4v_1^{(k+1)} - v_2^{(k)} - v_4^{(k)} = 1, \\
-v_1^{(k+1)} + 4v_2^{(k+1)} - v_3^{(k)} - v_5^{(k)} = 1, \\
-v_2^{(k+1)} + 4v_3^{(k+1)} - v_6^{(k)} = 1, \\
-v_1^{(k+1)} + 4v_4^{(k+1)} - v_5^{(k)} = 0, \\
-v_2^{(k+1)} - v_4^{(k+1)} + 4v_5^{(k+1)} - v_6^{(k)} = 0, \\
-v_3^{(k+1)} - v_5^{(k+1)} + 4v_6^{(k+1} = 0.
\end{cases}
$$

Важно заметить, что все компоненты v, соответствующие матричным элементам, лежащим на главной диагонали и ниже, имеют индекс $k+1$, тогда как компоненты, соответствующие элементам выше главной диагонали, имеют индекс k. Это отражает тот факт, что каждая компонента v_i заменяется во всех последующих вычислениях ее новым значением.

Мы теперь выразим (24.13) в матричной форме, которая допускает обобщение на любую систему линейных уравнений. Обозначим через A матрицу коэффициентов в (24.13). Пусть матрица A выражается как сумма двух матриц F и G. Здесь F есть нижняя треугольная матрица, состоящая из элементов A, лежащих на главной диагонали и ниже, и с нулями, расположенными выше главной диагонали, а G — верхняя треугольная матрица с нулями на главной диагонали и ниже

и с элементами матрицы A выше главной диагонали. Таким образом,

(24.14) $$A = F + G.$$

Обозначим правую часть (24.13) через b. Тогда система (24.8) может быть записана как система уравнений $Av = b$, а итерационный процесс (24.13) можно записать в виде

(24.15) $$Fv^{(k+1)} + Gv^{(k)} = b.$$

Для исследования сходимости $v^{(k)}$ допустим, что $e^{(k)}$ есть *ошибка* $v^{(k)}$ как решения уравнения (24.13):

$$e^{(k)} = v^{(k)} - v, \text{ где } v = A^{-1}b.$$

Вычитая (24.15) из тождества $Fv + Gv = b$, мы видим, что

(24.16) $$Fe^{(k+1)} + Ge^{(k)} = \theta.$$

Если ни один диагональный элемент a_{ii} матрицы A не равен нулю, то матрица F невырожденная, и мы можем определить новую матрицу $H = -F^{-1}G$. Тогда из (24.16) имеем

(24.17) $$e^{(k+1)} = He^{(k)}.$$

Наконец, применяя (24.17), мы можем по индукции доказать окончательный результат

(24.18) $$e^{(k)} = H^k e^{(0)}.$$

Мы заключаем, что для данного $v^{(0)}$ вектор $e^{(k)}$ стремится к θ при $k \to \infty$ тогда и только тогда, когда $H^k e^{(0)} \to \theta$ при $k \to \infty$, где $e^{(0)} = v^{(0)} - A^{-1}b$. Так как вектор $A^{-1}b$ не известен, то непрактично проверять, будет ли $H^k e^{(0)} \to \theta$ для данного вектора $e^{(0)}$.

Более полезен следующий общий результат:

(24.19) Теорема. *Пусть A — матрица с ненулевыми диагональными элементами a_{ii}. Предположим, что в итерационном методе последовательных смещений мы все вычисления проделываем точно. Тогда $v^{(k)} \to A^{-1}b$ при $k \to \infty$ для любого начального вектора $v^{(0)}$ тогда и только тогда, когда $H^k w \to \theta$ при $k \to \infty$ для любого вектора w.*

Мы напомним читателю важную теорему из теории матриц:

(24.20) Теорема. *Если H есть $(n\times n)$-матрица, то $H^k x \to \theta$ при $k\to\infty$ для любого вектора x тогда и только тогда, когда каждое собственное значение λ_i матрицы H по модулю меньше* 1.

За доказательством этой теоремы читателя можно отослать, например, к Вазову и Форсайту [40]. Основная идея состоит в том, что с помощью преобразования подобия матрицу H можно привести к блочно-диагональной матрице $S^{-1}HS$ (каноническая форма Жордана для матрицы H), каждый диагональный блок которой имеет вид

$$\begin{bmatrix} \lambda_i & 1 & & & & 0 \\ & \lambda_i & 1 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & 1 \\ 0 & & & & & \lambda_i \end{bmatrix}. \tag{24.21}$$

После этого надо только доказать теорему для произвольного блока вида (24.21).

Из (24.19) и (24.20) следует, что необходимым и достаточным условием сходимости метода последовательных смещений является то, что все собственные значения матрицы $-F^{-1}G$ должны быть по модулю меньше 1.

(24.22) Упражнение. Доказать, что для сходимости метода последовательных смещений необходимо и достаточно, чтобы все корни уравнения

$$\det(\lambda F + G) = 0$$

были по модулю меньше 1.

В большинстве практических случаев ни одно из этих двух последних условий нельзя легко проверить и для общих матриц трудно решить, будет ли метод последовательных смещений сходиться.

Для матрицы A в (24.8) или аналогичной матрицы при произвольном N метод последовательных смещений действительно сходится. Доказать это можно различными путями. Можно использовать общую теорему, устанавливающую сходимость всякий раз, когда A — положительно определенная симметричная матрица (см. Вазов и Форсайт [40]). В других случаях используется теорема, утверждающая сходимость для любой неразложимой матрицы A с диагональным преобла-

данием (см. (6.1) и (6.4) для определения этих понятий и сравни с упражнением (23.10)). Мы не будем приводить здесь ни одно из доказательств. (См. также Варга [7].) Из (24.18) вытекает и другое следствие. Предположим, для простоты примера, что H имеет единственное максимальное по модулю собственное значение λ_1 с соответствующим собственным вектором w, такое, что

$$Hw = \lambda_1 w.$$

Тогда нетрудно доказать, что

$$e^{(k)} = H^k e^{(0)} \simeq c\lambda_1^k w \text{ при } k \to \infty, \tag{24.23}$$

где e — некоторая постоянная. Более точно,

$$\| e^{(k)} - c\lambda_1^k w \| / \| e^{(k)} \| \to 0 \text{ при } k \to \infty.$$

Обычно для больших задач $|\lambda_1|$ близко к 1 и сходимость $e^{(k)}$ к θ является чрезвычайно медленной. Например, при $N \to \infty$ для задачи (24.5, 24.6) можно показать, что

$$\lambda_1 \sim 1 - \frac{25\pi^2}{144N^2} \simeq 1 - \frac{1{,}7}{N^2}.$$

Из-за слабой сходимости метода последовательных смещений программист должен обычно уделять внимание «ускорению» сходимости $v^{(k)}$ к θ, основанному на уравнениях типа (24.23). Построение и использование методики ускорения является важным при использовании итерационных методов. Мы здесь не можем входить в подробности.

Если память вычислительной машины устроена так, что $v^{(k)}$ хранится в запоминающем устройстве только со считыванием, может оказаться более удобным использовать значения одного фиксированного $v^{(k)}$ на всех стадиях вычисления $v^{(k+1)}$. Например, может оказаться удобным хранить все компоненты $v^{(k)}$ на магнитной ленте, считывать их, вычислять все компоненты $v^{(k+1)}$ из уравнения вида

$$\sum_{j=1}^{i-1} a_{ij} v_j^{(k)} + a_{ii} v_i^{(k+1)} + \sum_{j=i+1}^{n} a_{ij} v_j^{(k)} = b_i \quad (i = 1, 2, \ldots, n) \tag{24.24}$$

и заносить их на другую ленту — все в один прием. Заметим, что уравнения (24.24) отличаются от уравнений (24.13) тем, что внедиагональные компоненты v_i в (24.24) связаны со старым вектором $v^{(k)}$. Этот итерационный процесс, известный как *метод одновременных смещений* или *метод Якоби*, может быть также записан в виде (24.15), где под F мы теперь должны понимать матрицу диагональных элементов A. С учетом изме-

нения F мы можем сформулировать условия сходимости, аналогичные (24.20) и (24.22). Существуют интересные теоремы сравнения методов одновременных и последовательных смещений (см. Варга [7]).

Иногда системы уравнений типа (24.8) приближенно решаются с помощью электрических контуров или других аналоговых устройств. Как правило, эти методы тесно связаны с методом одновременных смещений, так как электрический ток течет в одно и то же время по всей цепи.

Более сложным итерационным методом является *метод последовательной верхней релаксации* (SOR). Это разновидность метода последовательных смещений, в которой каждая компонента v_i меняется поочередно, но не на величину Δv_i, необходимую для того, чтобы точно удовлетворялось i-е уравнение, а на $\omega \cdot \Delta v_i$, где ω $(1<\omega<2)$ выбирается так, чтобы скорость асимптотической сходимости $v^{(k)}$ при $k \to \infty$ была по возможности больше. Изложение этого метода, сходящегося для систем уравнений типа (24.8) много быстрее метода последовательных смещений, можно найти у Вазова и Форсайта [40].

Мы особенно подчеркиваем то, что большие и редкие матрицы A обычно не хранятся в памяти машины как массив из n^2 элементов. Требуется хранить самое большее ненулевые элементы A, а для простых операторов хранить вообще ничего не нужно, лучше вычислять элементы матрицы по мере надобности. Часто надо делать исключение для элементов A, соответствующих точкам сетки вблизи границы области или на границах раздела между различными средами. В основном же требуется хранить вычисляемый вектор $x^{(k)}$, а иногда также правую часть b. Вот поэтому, храня в памяти только один вектор x, мы можем существенно уменьшить объем памяти, используемой в итерационных процессах.

25. НЕЛИНЕЙНЫЕ СИСТЕМЫ УРАВНЕНИЙ

Часто оказывается необходимым решить нелинейную систему алгебраических или трансцендентных уравнений. Запишем такие уравнения в виде

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0, \\ \cdots\cdots\cdots \\ f_n(x_1, \ldots, x_n) = 0, \end{cases} \tag{25.1}$$

или в векторной форме

$$f(x) = \theta. \tag{25.2}$$

Пока не существует простой теории, которая могла бы нам сказать, имеет ли система (25.1) решение и является ли оно единственным. Если f_i являются полиномами от x_j, то часто можно дать оценку общего числа векторных решений x в комплексном проективном n-мерном пространстве, а это иногда помогает нам убедиться, что решения получены все.

Однако обычно каждая нелинейная система должна рассматриваться как специальная задача. Зачастую требуется приемлемое первое приближение к решению, которое затем может быть улучшено.

Имеются два общих метода для решения системы (25.1): метод спуска и метод Ньютона.

В методах спуска составляется вещественный функционал, который обращается в нуль на любом решении и положителен во всех других случаях. Один такой функционал имеет вид

$$G(x_1, \ldots, x_n) = \sum_{i=1}^{n} |f_i(x_1, \ldots, x_n)|^2. \tag{25.3}$$

Тогда решение (25.1) сводится к отысканию минимума функционала G и, в частности, минимума с нулевым значением.

Для минимизации функционала $G(x) = G(x_1, \ldots, x_n)$ по методу спуска сначала выбирается (обычно в соответствии с исходной задачей) начальный вектор x^0. Затем ищется направление d^0, такое, что $G(x) - G(x^0) < 0$ для векторов x вида $x^0 + \alpha d^0$ для небольших значений $\alpha > 0$. Одним из наилучших направлений является

$$d^0 = -\operatorname{grad} G(x) \text{ в точке } x^0,$$

где $\operatorname{grad} G(x)$ есть вектор

$$\left(\frac{\partial G}{\partial x_1}, \ldots, \frac{\partial G}{\partial x_n}\right)^T.$$

Как показывают вычисления, $\operatorname{grad} G(x)$ есть направление, в котором $G(x)$ увеличивается наиболее быстро. Так как $-\operatorname{grad} G(x)$ есть направление наибыстрейшего убывания $G(x)$, то предпочтительней именно в этом направлении пытаться существенно уменьшить $G(x)$. Образно говоря, требуется как можно быстрее спуститься по поверхности $G(x)$, чтобы достигнуть ее нижней точки. В принципе применение метода будет успешным, лишь если мы не натолкнемся на локальный минимум $G(x)$, значение которого не равно нулю. Практически спуск оказывается часто очень медленным даже при отсутствии ненулевых минимумов. Чтобы понять трудности для случая $n=2$, представьте, что вы застряли в тумане где-то на почти плоском участке очень узкой долины между двумя крутыми склонами и пытаетесь достигнуть нижней точки долины. Где бы вы ни блуждали, вы будете непрерывно натыкаться на склоны, почти не продвигаясь к цели. Трудности почти неограниченно возрастают, если значение n существенно больше двух. Таким образом, в этих проблемах является важным выбор ускорения метода.

Иногда характер $G(x)$ таков, что вычислить $\operatorname{grad} G(x)$ очень трудно. В таких случаях производные часто можно аппроксимировать конечно-разностными выражениями. Например, $\frac{\partial G(x)}{\partial x_i}$ в точке x^0 можно аппроксимировать выражением

$$[G(x^0+he_i)-G(x^0-he_i)]/2h,$$

где e_i есть i-й единичный вектор. Тогда n разностных выражений могут быть совместно использованы для аппроксимации вектора градиента $\operatorname{grad} G(x)$. Таким образом, $2n$ значений $G(x)$ могут дать приемлемое приближение для градиента.

Если вычисление $G(x)$ сложно, можно использовать значения только в $n+1$ вершинах правильного симплекса в n-мерном пространстве, а затем аппроксимировать производные соответствующими комбинациями этих $n+1$ величин. Такие методы исследованы сравнительно мало.

Примером задач, где вычисление $G(x)$ очень громоздко, а определение градиента почти невозможно, являются задачи, в которых вычисление $G(x)$ требует расчета длинной траектории.

Короче говоря, методы спуска могут использоваться для решения нелинейных систем даже в тех случаях, когда вычисление производных непрактично.

В методе Ньютона сначала выбирается x^0, затем нелинейная задача аппроксимируется соответствующей линейной задачей, а также вычисляются производные от f_i в точке x^0.

Такая линеаризованная задача представляет систему линейных алгебраических уравнений. Ее решение дает приращение, которое складывается с x^0 для получения x^1, а последнее обычно является более хорошим приближением к решению нелинейной системы. Этот процесс продолжается до сходимости. Метод является точным обобщением на n-мерный случай ньютоновского процесса нахождения нулей вещественной функции вещественной переменной.

Более подробно ньютоновский процесс заключается в следующем. Сначала мы имеем первое приближение x^0. Мы аппроксимируем каждую функцию f_i из (25.1) двумя членами ряда Тейлора в точке x^0:

$$(25.4)\quad f_i(x) \simeq f_i(x^0) + \sum_{j=1}^{n} \frac{\partial f_i(x^0)}{\partial x_j}(x_j - x_j^0) \qquad (i = 1, 2, \ldots, n).$$

Обозначим через $J(x)$ значение в точке x якобиана системы $f_i(x)$, т. е.

$$J(x) = \begin{vmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{vmatrix}.$$

Тогда (25.4) примет вид

$$(25.5)\qquad f(x) \simeq f(x^0) + J(x^0)(x - x^0).$$

Правая часть представляет собой линейную векторную функцию от x, которая наилучшим образом аппроксимирует нелинейную функцию f в точке x^0. Это гиперплоскость, касательная к поверхности $f(x)$ в точке x^0. В ньютоновском процессе мы полагаем правую часть (25.5) равной θ, решаем полученное уравнение относительно x и обозначаем решение через x^1. Таким образом,

$$(25.6)\qquad x^1 := x^0 - [J(x^0)]^{-1} f(x^0).$$

Аналогично, общий итерационный шаг есть

$$(25.7)\qquad x^{k+1} := x^k - [J(x^k)]^{-1} f(x^k).$$

В двумерном случае система (25.1) отыскивает точку пересечения двух кривых в плоскости (x_1, x_2), например $f(x_1, x_2) = 0$ и $g(x_1, x_2) = 0$. Рассмотрим поверхность S_1, описываемую уравнением $x_3 = f(x_1, x_2)$, и другую поверхность S_2 с

уравнением $x_3=g(x_1, x_2)$. Поверхности S_1 и S_2 пересекаются по кривой C. Решение системы (25.1) может быть также интерпретировано как точка P, в которой C пересекает плоскость $x_3=0$. В ньютоновском процессе мы строим касательную плоскость Π_1 к поверхности S_1 в точке x^k и вторую касательную плоскость Π_2 к поверхности S_2 в той же точке. Плоскости Π_1 и Π_2 пересекаются по прямой L, которая пересекает плоскость $x_3=0$ в точке x^{k+1}. Таким образом, ньютоновский процесс заменяет поверхности касательными плоскостями, а это в данном случае означает линеаризацию.

На каждом итерационном шаге вычисления включают нахождение n^2 производных и решение линейной алгебраической системы для получения решения системы (25.7). После решения системы следующий шаг состоит в определении вектора невязки $f(x^k)$, который должен быть вычислен с высокой точностью. Сам якобиан не требуется знать очень точно, фактически часто можно оставлять $J(x)$ неизменным на протяжении нескольких шагов итерации. Читатель может вспомнить, что это дает большую экономию при решении линейной системы, так как при этом может быть использована процедура *SOLVE* вместо всего блока решения линейных систем.

Мы не обсуждаем вопрос о выборе первого приближения x^0, готового ответа на него нет. Часто постановщик задачи знает достаточно, чтобы предложить довольно хорошее первое приближение. В других случаях для получения x^0 можно использовать метод спуска. Иногда бывают задачи с исследованием влияния параметров, в которых каждое решение незначительно отличается от предыдущего. Тогда решение одной задачи может служить начальным приближением для следующей.

Характер сходимости ньютоновского процесса подробно обсуждается у Хенричи [42]. Известен следующий результат. Пусть a будет решением системы (25.1), $f(a)=\theta$. Предположим, что f_i имеют несколько непрерывных производных. Тогда существует некоторое число r, такое, что выполнение условия $\|x^0-a\|\leqslant r$ обеспечивает $x^k\to a$ при $k\to\infty$. Более того, если якобиан $J(a)$ невырожден, то сходимость x^k к a будет квадратичной, т. е.

$$\lim_{k\to\infty}\frac{\|x^{k-1}-a\|}{\|x^k-a\|^2}$$

является ограниченным неотрицательным числом (обычно не равным нулю). Квадратичная сходимость является очень быстрой в том смысле, что на каждом итерационном шаге число верных десятичных знаков примерно удваивается (пока не достигнут уровень округлений).

Что случится, если x^0 окажется далеким от любого решения a? Это вопрос сложный, а ответ зависит от f. Вычислительная практика показывает, что x^k довольно часто достигает какого-либо решения a, но может совершать скачки в некоторых итерациях. Может оказаться невозможным выяснить, какое решение будет достигнуто.

Наконец, нередки случаи, когда x^k не сходится. Вместо этого может оказаться, что x^k будет с небольшим периодом принимать конечное число предельных значений. Это может, например, случиться при использовании ньютоновского процесса для одной переменной с вещественным начальным значением x^0, если f имеет только комплексные корни.

Несмотря на возможные трудности, нет оснований терять оптимизм. Если неизвестно, где находится решение a и сколько всего имеется решений, можно выбрать какое-либо x^0 и начать ньютоновский алгоритм. Обычно он к чему-нибудь сходится. Затем можно начать алгоритм еще откуда-нибудь и найти другое решение. Таким образом, часто можно найти ряд решений нелинейной системы. Если, однако, требуется найти *все* решения, то указанный выше метод не дает ключа к решению вопроса. Более того, для некоторых систем мы можем находить одно и то же решение. Последнее можно часто предотвратить изменением функции по следующему методу. Предположим, что мы нашли решение a системы $f(x)=\theta$. Положим

$$g(x)=f(x)/\|x-a\|.$$

Если мы применим ньютоновский процесс к функции g, то близость числителя к нулю будет уравновешиваться близостью к нулю знаменателя, а это будет удерживать приближение вдали от a. Вычисление $g(x)$ для данного x ненамного труднее, чем определение $f(x)$. Однако производные становятся более сложными, особенно после того, как уже найдено несколько корней и в знаменателе введены соответствующие множители.

По решению нелинейных систем литературы имеется сравнительно мало. Теперь, когда мы имеем такие надежные средства, как программы решения линейных систем, можно направить исследования на нелинейные задачи. Читатель может обратиться к работам Вендроффа [8], гл. 4, и Ралстона [24], гл. 8.

ПРИЛОЖЕНИЕ

(1) Схема доказательства утверждения (8.21). Прежде всего доказать, что $a^2+b^2+c^2+d^2$ и $|ad-bc|$ инвариантны при преобразовании A в U^TAV для любых ортогональных матриц U и V. Затем использовать (3.1) для преобразования A к виду

$$\begin{bmatrix} \mu_1 & 0 \\ 0 & \mu_2 \end{bmatrix}, \text{ где } \mu_1 \geqslant \mu_2 > 0.$$

Далее,

$$\sigma = \frac{\mu_1^2+\mu_2^2}{2\mu_1\mu_2} = \frac{1}{2}\left(\frac{\mu_1}{\mu_2}+\frac{\mu_2}{\mu_1}\right) = \\ = \frac{1}{2}\left[\operatorname{cond}(A)+\operatorname{cond}(A)^{-1}\right].$$

Разрешая последнее уравнение относительно $\operatorname{cond}(A)$, получаем

$$\operatorname{cond}(A) = \sigma + \sqrt{\sigma^2-1},$$

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

(2) Частное решение для задачи (6.5). Если бы матрица A была вырожденной, то существовал бы вектор $x=(x_1, \ldots, x_n)^T$, такой, что $Ax=\theta$. Пусть $|x_k| = \max\limits_j |x_j|$. Тогда

$$-a_{kk}x_k = \sum_{\substack{j=1 \\ (j\neq k)}}^{n} a_{kj}x_j.$$

Следовательно:

$$|a_{kk}| \leqslant \sum_{j\neq k} |a_{kj}| \frac{|x_j|}{|x_k|}.$$

(a) Предположим, что для некоторого j $|x_k|>|x_j|$ и $|a_{kj}|\neq 0$. Тогда

$$|a_{kk}| < \sum_{j\neq k} |a_{kj}|,$$

что противоречит (6.2)

(b) Мы оставляем читателю случай, когда п. (a) не выполняется. Полное доказательство см. Тауски [26].

БИБЛИОГРАФИЯ

1. Бауман, Феличиано, Бауэр и Самелсон (Baumann R., Feliciano M., Bauer F. L. and Samelson K.), Introduction to Algol, Englewood Cliffs, N. J., Prentice-Hall, 1964.

2. Бауэр (Bauer F. L.), On the Definition of Condition Numbers and on Their Relation to Closed Methods for Solving Linear Systems, стр. 109—110 из [44].

3. —, Optimal Scaling of Matrices and the Importance of Minimal Condition, стр. 198—201 из [23].

4. —, Optimally Scaled Matrices, *Numer. Math.*, 5 (1963), 73—87.

5. Боудлер, Мартин, Питерс и Уилкинсон (Bowdler H. J., Martin R. S., Peters G. and Wilkinson J. H.), Solution of Real and Complex Systems of Equations, *Numer. Math.*, 8 (1966), 217—234.

6. Бузингер и Голуб (Businger P. and Golub G. H.), Linear Least Squares Solutions by Householder Transformations, *Numer. Math.*, 7 (1965), 269—276.

7. Варга (Varga R. S.), Matrix Iterative Analysis, Englewood Cliffs, N. J., Prentice-Hall, 1962.

8. Вендрофф (Wendroff B.), Theoretical Numerical Analysis, New York, Academic Press, 1966.

9. Гивенс (Givens W.), Numerical Computation of the Characteristic Values of a Real Symmetric Matrix, Report ORNL 1574; Oak Ridge, Tenn., Oak Ridge National Laboratory, 1954.

10. Гильберт (Hilbert D.), Ein Beitrag zur Theorie des Legendreschen Polynoms, *Acta Math.*, 18 (1894), 155—160.

11. Голуб (Golub G.), Numerical Methods for Solving Linear Least Squares Problems, *Numer. Math.*, 7 (1965), 206—216.

12. Голуб и Кахан (Golub G. and Kahan W.), Calculating the Singular Values and Pseudoinverse of a Matrix, *J. SIAM Numer. Anal.*, Ser. B, 2 (1965), 205—224.

13. Данциг (Danzig G. B.), Linear Programming and Extensions, Princeton, N. J., Princeton University Press, 1963.

14. Кахан (Kahan W.), The Floating Point Over/Underflow Trap Routine FPTRP, Programmers' Reference Manual, Section 4. 1, Toronto, Canada, University of Toronto, Institute of Computer Science, March 1965.

15. —, Writeups of library tape subroutines LEQU, LEQUN, FLEQU, CLEQU, DLEQU, to run under Toronto modifications of FORTRAN IV and MAP under IBSYS on a 7094-II, Toronto, Canada, University of Toronto, Institute of Computer Science, 1965.

16. Клюев и Коковкин-Щербак, О минимизации числа арифметических операций при решении линейных алгебраических систем уравнений. *ЖВММФ*, 5 (1965) № 1, 21—33.

17. Ланцош (Lanczos C.), Applied Analysis, Englewood Cliffs, N. J., Prentice-Hall, 1956.

18. Мак-Киман (McKeeman W. M.), Algorithm 135; Crout with Equilibration and Iteration, *Comm. Assoc. Comput. Mach.*, 5 (1962), 553—555.

19. Мартин, Питерс и Уилкинсон (Martin R. S., Peters G. and Wilkinson J. H.), Iterative Refinement of the Solution of a Positive Definite System of Equations, *Numer. Math.*, 8 (1966), 203—216.

20. Молер (Moler C. B.), SOLVE, Accurate Simultaneous Linear Equation Solver with Iterative Improvement, SHARE distribution № 3194, 1964.

21. —, Iterative Refinement in Floating Point, *J. Assoc. Comput. Mach.*, 14 (1967).

22. Наур (ред.) и др. (Naur P. (editor) et al.), Revised Report on the Algoritmic Language ALGOL 60, *Comm. Assoc. Comput. Mach.*, 6 (1963), 1—17.

23. Поплвелл (ред.) (Popplewell G. M. (editor)), Information Processing, Amsterdam, North-Holland Publishing Co., 1962.

24. Ралстон (Ralston A.), A First Course in Numerical Analysis, New York, McGraw-Hill Book Company, 1965.

25. Саваж и Лукас (Savage I. R. and E. Lukacs), Table of Inverses of Finite Segments of the Hilbert Matrix, стр. 105—108 из [27].

26. Тауски (Taussky Olga), A Recurring Theorem on Determinants, *Amer. Math. Monthly*, 51 (1949), 672—676.

27. — (ред.), Contributions to the Solution of Systems of Linear Equations and the Determination of Eigenvalues, U. S. National Bureau of Standards, Applied Mathematics Series, 39, Washington D. S., Supt. of Documents, 1954.

28. Тодд (Todd J.), The Condition of the Finite Segments of the Hilbert Matrix, стр. 109—116 из [27].

29. —, Computational Problems Concerning the Hilbert Matrix, *J. Research Nat. Bur. Standards*, Ser. B., **65** (1961), 19—22.

30. Торнхейм (Tornheim L.), Maximum Third Pivot for Gaussian Reduction, unpublished manuscript, Richmond, Calif. California Research Corp., 1965.

31. Уилкинсон (Wilkinson J. H.), Error Analysis of Direct Methods of Matrix Inversion, *J. Assoc. Comput. Mach.*, **8** (1961), 281—330.

32. —, Rounding Errors in Algebraic Processes, Englewood Cliffs, N. J., Prentice-Hall, 1963.

33. —, The Algebraic Eigenvalue Problem, Oxford, Clarendon Press, 1965.

34. Фаддеев Д. К. и Фаддеева В. Н., Вычислительные методы линейной алгебры, Физматгиз, 1960.

35. Фаддеева В. Н., Вычислительные методы линейной алгебры, Гостехиздат, 1950.

36. Фокс (Fox L.), Introduction to Numerical Linear Algebra, Oxford, Clarendon Press, 1964.

37. Форсайт (Forsythe G. E.), Singularity and Near Singularity in Numerical Analysis, *Amer. Math. Monthly*, **65** (1958), 229—240.

38. —, Algorithm 16; Crout with Pivoting in ALGOL 60, *Comm. Assoc. Comput. Mach.*, **3** (1960), 507—508.

39. —, Today's Computational Methods of Linear Algebra, *SIAM Review*, **9** (1967).

40. Форсайт и Вазов (Forsythe G. E. and W. R. Wasow), Finite Difference Methods for Partial Differential Equations, New York, Wiley, 1960. (Русский перевод: Вазов В., Форсайт Дж., Разностные методы решения дифференциальных уравнений в частных производных, ИЛ, 1963.)

41. Хаусхолдер (Householder A. S.), Principles of Numerical Analisis, New York, McGraw-Hill Book Company, 1953. (Русский перевод: Хаусхолдер А. С., Основы численного анализа, ИЛ, 1956.)

42. Хенричи (Henrici P.), Elements of Numerical Analysis, New York, Wiley, 1964.

43. American Standards Association, A Programming Language for Information Processing on Automatic Data Processing Systems, FORTRAN vs. Basic FORTRAN, *Comm. Assoc. Comput. Mach.*, **7** (1964), 591—625.

44. Anonymous, Information Processing; Proceedings of the International Conference on Information Processing, Unesco, Paris, 15—20 June 1959; Paris: Unesco; Munich: Oldenbourg; London: Butterworth, 1960.

45. Burroughs, B 5500 Information Processing System Extended Algol Reference Manual, Detroit, Mich., Burroughs Corp., 1964.

46. Control Data, FORTRAN-63 Reference Manual, v. I, II, Publication Numbers 527, 528, Palo Alto, Calif., Control Data Corp. 1963.

47. International Business Machines, IBM 7090/7094 IBSYS Operating System; Version 13; FORTRAN IV Language, IBM Systems Reference Library, File № 7090-25, Form C28-6390-2, White Plains, N. Y., IBM, 1965.

48. —, IBM System/360: Basic Programming Support FORTRAN IV, IBM Systems Reference Library, File № S360-25, Form C28-6504-2, White Plains, N. Y., IBM, 1965.

49. —, IBM Operating System/360; PL/I: Language Specifications, IBM Systems Reference Library, File № S360-29, Form C28-6571-2, White Plains, N. Y., IBM, 1966.

50. National Physical Laboratory, Modern Computing Metholds, 2nd ed., Notes on Applied Science, № 16, London, Her Majesty's Stationery Office, 1961.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

СОДЕРЖАНИЕ