С. Писсанецки

Технология разреженных матриц

Sparse Matrix Technology

Sergio Pissanetzky
Centro Atomico Bariloche,
Bariloche, Argentina

1984
ACADEMIC PRESS INC.
HARCOURT BRACE JOVANOVICH, PUBLISHERS
LONDON ORLANDO SAN DIEGO
NEW YORK AUSTIN TORONTO MONTREAL
SYDNEY TOKYO

С.Писсанецки

Технология разреженных матриц

Перевод с английского
Х. Д. ИКРАМОВА и И. Е. КАПОРИНА

под редакцией
Х. Д. ИКРАМОВА

Москва «Мир» 1988

ББК 22.193
П34
УДК 512.83

Писсанецки С.

П34 Технология разреженных матриц: Пер. с англ.—М.: Мир, 1988. —410 с, ил.

ISBN 5-03-000960-4

Книга американского специалиста, охватывающая практически все алгебраические задачи по разреженным матрицам. Представлено много фактического материала по технологии применения разреженных матриц, изложение последовательное и наглядное. Книга удачно дополняет имеющуюся на русском языке литературу по этой тематике.

Для математиков-вычислителей, инженеров, аспирантов и студентов вузов.

П $\frac{1702070000—190}{41\ (01)—88}$ 38—88 ч. 1 ББК 22.193

Редакция литературы по математическим наукам

ISBN 5-03-C00960-4 (русск.)
ISBN 0-12-557580-7 (англ.)

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Литература на русском языке, посвященная разреженным матрицам, пополнилась за последние годы несколькими переводными книгами: Парлетт Б. Симметричная проблема собственных значений. Численные методы. — М.: Мир, 1983; Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. — М.: Мир, 1984; Эстербю О., Златев З. Прямые методы для разреженных матриц. — М.: Мир, 1987. И вот еще одна книга на ту же тему. Читатель, знакомый с упомянутыми изданиями, вправе знать, чем новая книга отличается от своих предшественниц.

Чтобы ответить на этот вопрос, напомним, что в каждой из названных выше книг рассматривается лишь какая-нибудь одна задача линейной алгебры, а подчас даже ее специальный случай — симметричный спектральный анализ (Парлетт), решение симметричных положительно определенных систем (Джордж), несимметричные системы (Эстербю, Златев). В книге Писсанецки изложены методы решения всех этих задач и, кроме того, в соответствии с названием уделено много внимания технологической стороне дела. Структуры данных и типовые операции, используемые при обработке разреженных матриц, четко систематизированы, а операции к тому же записаны фортранными текстами. Должное внимание уделено вопросу об ошибках округлений. Рассмотрены приложения технологии разреженных матриц к задачам метода конечных элементов, включая фронтальные методы для решения соответствующих линейных систем как симметричных, так и несимметричных.

Итак, перед читателем — наиболее полная из книг по разреженным матрицам. Ее можно использовать и как справочник, и как учебник, потому что изложение ведется на уровне, доступном студенту.

В зарубежных реферативных изданиях и журналах, дающих обзоры книжных новинок, эта книга уже получила положительные отзывы ряда известных специалистов по разреженным матрицам: Р. Тьюарсона («Zentralblatt fur Mathematik», 536, *65019), И. Даффа («Mathematical Review», 86j, *65003), Р. Уиллафби (SIAM Review, 1986, 28, № 1, p. 130). Справедливости

ради отметим, что Дафф высказывает и некоторые критические замечания, относящиеся, в частности, к оформлению фортранных текстов.

Автор с энтузиазмом принял сообщение о подготовке русского издания книги и предполагал написать дополнение к ней. Однако обстоятельства помешали ему сделать это. С его согласия дополнение написано мною. В нем подробно рассказано о четырех наиболее важных подпрограммах MA28, MA32, MA27, MA37 знаменитой английской программной серии по решению разреженных задач.

В переводе книги я сотрудничал с И. Е. Капориным. Работа между нами была распределена так: гл. 2, 3, 6 и § 1—13 главы 4 переводил И. Е. Капорин, остальное переведено мною.

Х. Икрамов

Авроре, Людмиле и Пабло

ПРЕДИСЛОВИЕ

По мере того как растут производительность и быстродействие вычислительных машин, растут размеры матриц. В 1968 г. типовой практической задачей считалось решение линейной алгебраической системы порядка 5000; систему порядка 10 000 или более оценивали как «большую» [Tinney, 1969, p. 28; Willoughby, 1971; p. 271]. В 1978 г. появилось сообщение [Kolata, 1978] о переопределенной задаче с 2.5 млн. уравнений и 400 000 неизвестными; в 1981 г. размеры этой задачи еще больше выросли [Golub, Plemmons, 1981, p. 31: 6 000 000 уравнений при прежнем количестве неизвестных. Матрица задачи имеет $2.4 \chi 10^{12}$ элементов, большая часть которых нули; это — разреженная матрица. Та же тенденция к увеличению размеров наблюдается в спектральных расчетах, где «большой» называют матрицу порядка 4900 или 12 000 [Cullum, Willoughby, 1981, p. 329; Parlett, 1980, p. XIII]. Будут ли порядки матричных задач расти и дальше? И будет ли поспевать за этим ростом наша способность решать большие задачи?

Но это только одна сторона вопроса. Другая сторона — микрокомпьютерный взрыв. МикроЭВМ имеют сейчас ту же производительность, какую имели два десятка лет назад большие машины. Значит ли это, что пользователи микрокомпьютеров могут решать матричные задачи только таких порядков, как двадцать лет тому назад?

Владелец микроЭВМ может не слишком заботиться о стоимости счета; основное ограничение — память. На большой машине с ростом порядка матричной задачи быстро растет и стоимость ее решения, что связано с увеличением и трудозатрат, и памяти. Общая стоимость становится здесь решающим фактором. Как минимизировать эту стоимость для конкретной задачи и вычислительной установки?

В предлагаемой книге даны ответы на эти и родственные вопросы для следующих классов матричных задач: прямого решения разреженных линейных алгебраических уравнений, разреженных обычных и обобщенных спектральных задач, алгебры разреженных матриц. Описаны и очень простые, но на удивление эффективные методы и весьма сложные алгоритмы. Технология разреженных матриц является сейчас вполне установившейся

дисциплиной, определенной как «искусство обращаться с разреженными матрицами» [Harary, 1971]. Она представляет собой замечательный сплав теоретических достижений, численной практики и здравого смысла. Она не только служит важным вычислительным средством во многих прикладных областях [Rose, Willoughby, 1972], но и сама по себе составляет ценный вклад в общий процесс развития машинного программного обеспечения. Для ряда матричных задач, основываясь на идеях, появившихся за последние пятнадцать лет, удалось создать почти оптимальные алгоритмы. Исследования в этой области в настоящее время очень активны, и спектр приложений постоянно расширяется. Технология разреженных матриц существует и будет существовать.

Предмет, представляющий для нас главный интерес, отражен в названии книги. «Технология — прикладная наука, исследование какого-либо ремесла» (Webster's dictionary, 2-nd ed., 1957 — словарь Уэбстера, второе издание, 1957). В начале 1970-х годов термин «технология разреженных матриц» был в повседневном употреблении в Научно-исследовательском центре имени Уотсона фирмы IBM [Willoughby, 1971; Rose, Willoughby, 1972, Preface; Willoughby, 1972; Hachtel, 1976, p. 349]. Сейчас он, кажется, вышел из моды. Материал для книги отбирался из публикаций нескольких симпозиумов и конгрессов по большим матричным задачам; они регулярно проводятся с 1968 г. [Willoughby, 1969; Reid, 1971a; Rose, Willoughby, 1972; Bunch, Rose, 1976; Duff, Stewart, 1979; Duff, 1981b]. Труды симпозиума, происходившего в Фэйрфилд Глейд, Теннесси, в 1982 г., опубликованы в специальном выпуске журнала: SIAM Journal on Scientific and Statistical Computing, 1984, v. 5, № 3. Существует каталог по программному обеспечению, подготовленный в связи с этим симпозиумом [Heath, 1982]. Основное влияние на отбор материала оказали: курс повышенного типа с четырьмя обзорными статьями [Barker, 1977], прекрасные обзоры [Duff, 1972; 1982] и книги [Wilkinson, 1965; Parlett, 1980, George, Liu, 1981], сборник статей [Bjorck et al., 1981] и многие отдельные публикации, цитируемые в тексте книги. Несколько важных идей заимствовано из литературы, опубликованной до 1973 г. [Brayton, et al., 1970; Willoughby, 1972; Tewarson, 1973]. Однако мы не пытаемся охватить весь этот огромный материал. Напротив, вводятся и подробно описываются лишь основные методы и процедуры; обсуждение по каждому предмету доведено до того места, где читатель сможет уже самостоятельно разобраться в специальной литературе. Насколько возможно, мы пытались дать единообразное изложение, хотя, как во всякой быстро растущей области человеческого знания, технология разреженных матриц развивалась неровно. Некоторые

разделы продвинуты далеко, другие еще требуют разработки. Мы не приводим доказательства теорем, за исключением тех, что тесно связаны с используемыми в дальнейшем практическими методами. Понятия и методы вводятся на элементарном уровне, во многих случаях с помощью простых примеров. Описаны и тщательно разобраны многие важные алгоритмы. Приведены готовые к использованию эффективные и профессиональные записи алгоритмов на Фортране. Предполагается, что читатель знаком с этим популярным языком. Впрочем, алгоритмы разъяснены так детально, что и читатель с ограниченным знанием Фортрана сможет разобраться в них и при необходимости перевести их на другие языки. В книге широко используются линейная алгебра и теория графов. Однако глубокое знание этих дисциплин не обязательно, поскольку вводятся все нужные определения и свойства, начиная с азов. Некоторая подготовка все же может быть полезна. Во многих разделах приведены обзоры соответствующей литературы и подробная библиография. Книга заполняет разрыв между монографиями на тему о конструировании машинных алгоритмов и специализированной литературой по методам для разреженных матриц, с одной стороны, и потребностями пользователей и запросами практики, с другой.

Цель книги — сделать технологию разреженных матриц доступной инженерам, программистам, аналитикам, преподавателям и студентам. Эта книга будет полезна каждому, кто желает написать свою собственную программу для разреженных матриц, или тому, кто пользуется готовой программой, но хотел бы разобраться, как она работает, или, наконец, тому, кто планирует приобрести пакет для разреженных матриц и нуждается в расширении своего понимания этого предмета. Преподаватель, заинтересованный в профессиональном, но элементарном изложении методов для разреженных матриц и в различных примерах их приложений, найдет такой материал в этой книге.

В гл. 1 рассмотрены вводные темы: схемы хранения, основные определения и вычислительные приемы, используемые в технологии разреженных матриц. Имеет смысл для начала прочесть хотя бы § 1—9 и 12 гл. I. Однако первое знакомство может быть поверхностным; Читатель получит мотивы для более детального изучения этого материала при чтении последующих глав книги, где даны многочисленные ссылки на гл. 1.

В гл. 2—5 рассматривается задача решения линейных алгебраических уравнений. Эти главы не независимы. Содержание гл. 2 элементарно, но форма его представления служит подготовкой к гл. 4 и 5, где изложен важный материал. В гл. 3 речь идет об ошибках вычислений в случае разреженной линейной системы; эта глава также служит введением в гл. 4—5, и ее содержание не является стандартным. Алгоритмы прямого решения разрежен-

ных линейных уравнений обсуждаются в гл. 4 и 5. Глава 4 посвящена симметричным матрицам, гл. 5 — матрицам общего вида.

Вычисление собственных значений и собственных векторов разреженной матрицы, или пары разреженных матриц в случае обобщенной проблемы собственных значений, — предмет гл. 6. Ее можно читать независимо от других глав, исключая некоторые ссылки на гл. 1 и 7.

В гл. 7—9 рассматриваются разреженные матрицы, хранимые в строчном формате. В гл. 7 подробно обсуждаются алгоритмы для алгебраических операций, треугольного разложения и обратной подстановки; приведена их запись на языке Фортран. Используется материал гл. 1, в особенности § 8—10 и 12—17. Кроме того, для чтения § 23—28 нужно знакомство с гл. 2. В гл. 8 изучаются алгоритмы для разреженных задач сеточного происхождения, связанных главным образом с методом конечных элементов. В гл. 9 представлены некоторые фортранные алгоритмы общего назначения.

Технология разреженных матриц применяется почти во всякой области, где вообще используются матрицы. Тому, кто интересуется конкретным приложением, будет полезно кроме соответствующих глав книги изучить литературу, где это приложение описано в деталях. Существует библиографический обзор [Duff, 1977, p. 501], систематизированный в соответствии с приложениями. Много статей, посвященных различным приложениям, можно найти в трудах конференции IMA 1980 г. [Duff, 1981b] и других публикациях [Bunch, Rose, 1976; Duff, Stewart, 1979].

На коммерческом рынке сейчас имеются хорошие робастные программы для разреженных матриц. В каталоге [Heath, 1982] перечислено более 120 программ. Многие программы описаны в каталоге Харуэлла [Hopper, 1980] и двух обзорных статьях [Duff, 1982; Parlett, 1983]. Составление хорошей программы для разреженных матриц — дело непростое. Оно требует мастерского программирования. Как и в любой области техники, конструктор программы должен построить прототип, тщательно испытать [Duff, 1979, Eisenstat et al., 1979, Duff et al., 1982] и усовершенствовать его, прежде чем будет получена окончательная модель и начнется массовое производство. В применении к программному обеспечению массовое производство означает многократное копирование программы и перенос ее на многие разные машины. Следовательно, программа должна быть переносимой. С точки зрения пользователя, инженер по программному обеспечению ответствен за правильный выбор программы и файловых структур и установку их на машине. Для пользователя продуктом служит не программа, а результат. Обеспечить выполнение требований, предъявляемых к хорошей программе, нелегко. В соответствующих разделах этой книги обсуждаются характеристики и нали-

чие программного обеспечения для каждого конкретного приложения.

Я хотел бы с признательностью отметить сотрудничество с Нейлом Колвудом. Он несколько раз прочел рукопись, исправив многие мои грамматические погрешности, и ответствен за «английский акцент», который местами может ощутить читатель. Хотелось бы также поблагодарить миссис Карлотту Р. Глюклих за терпение и самоотверженность при печатании рукописи и ее переработанных разделов.

Январь 1984 *Серхио Писсанецки*

ВВЕДЕНИЕ

Во многих областях человеческой деятельности информацию часто представляют в форме матриц. Матрица — это регулярный числовой массив. В специальной литературе имеется несколько определений *разреженной матрицы*. Суть их состоит в том, что матрица разрежена, если в ней «много» нулевых элементов. Некоторые авторы используют понятия, связанные с предельным переходом: матрицу порядка n они называют разреженной, если число ее ненулевых элементов есть $O(n)$. Это означает, что число отличающихся от нуля элементов при достаточно больших n пропорционально n. Однако для заданной матрицы n не является достаточно большим, а лишь вполне конкретным числом. Приведенное определение полезно только для теоретических целей типа попытки оценить асимптотическое поведение алгоритма. В [Brayton et al., 1970] критерием разреженности предлагается считать ограниченность числа ненулевых элементов в строке, в типичном случае от 2 до 10. Другое определение принадлежит Альварадо [Alvarado, 1979]: чтобы матрица порядка n была разреженной, число ее ненулевых элементов должно выражаться как $n^{1+\gamma}$, где $\gamma < 1$. Типичные значения γ: 0.2 для электрических задач; 0.5 для ленточных матриц, ассоциированных с сетками. Матрица порядка 1000 при $\gamma = 0.9$ имеет 501187 ненулевых элементов, и возникает вопрос, стоит ли считать такую матрицу разреженной.

Более практичный подход к определению разреженной матрицы опирается на взаимосвязь трех основных ингредиентов: данной матрицы, данного алгоритма и данной вычислительной машины. Это понятие по своей природе является эвристическим: матрица разрежена, если имеет смысл извлекать выгоду из наличия в ней многих нулей. Всякую разреженную матрицу можно обрабатывать так, как если бы она была плотной: напротив, всякую плотную, или заполненную, матрицу можно обрабатывать алгоритмами для разреженных матриц. В обоих случаях получатся правильные численные результаты, но вычислительные затраты возрастут. Приписывание матрице свойства разреженности эквивалентно утверждению о существовании алгоритма, использующего ее разреженность и делающего вычисления с ней дешевле по сравнению со стандартными алгоритмами.

Разреженную матрицу можно рассматривать как числовой массив, в общем случае не регулярный. Однако мы ассоциируем с этими числами позиции в значительно большем регулярном массиве, поскольку мы привыкли думать в терминах регулярных массивов и исходить из них при конструировании алгоритмов. Рассмотрим следующую систему восьми линейных уравнений с восемью неизвестными:

$$\begin{aligned} x_3 &= 3, \\ x_1 + 2x_6 &= 16, \\ x_1 + x_2 &= 3, \\ x_2 - x_8 &= -6, \\ x_4 - 2x_5 + x_7 &= 1, \\ -x_3 + x_6 &= 3, \\ -x_5 + x_7 &= 2, \\ -2x_4 + x_8 &= 0. \end{aligned}$$

Этой системе можно поставить в соответствие такую матрицу коэффициентов:

	1	2	3	4	5	6	7	8
1	0	0	1	0	0	0	0	0
2	1	0	0	0	0	2	0	0
3	1	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0	−1
5	0	0	0	1	−2	0	1	0
6	0	0	−1	0	0	1	0	0
7	0	0	0	0	−1	0	1	0
8	0	0	0	−2	0	0	0	1

Эта система имеет только 16 коэффициентов, а мы используем 64 позиции! С этой точки зрения даже кажется весьма неестественным заполнять несущественные позиции нулями, а затем пытаться извлечь выгоду из наличия столь большого числа нулей. Строго говоря, разреженную матрицу следует вообще представлять себе не как матрицу, а скорее как граф, где каждая пара уравнение — неизвестное ассоциируется с вершиной, а каждый коэффициент — с ребром. Вот почему теория графов играет такую важную роль в технологии разреженных матриц. И в этом как раз причина возникновения технологии разреженных матриц. Разреженная матрица, будучи множеством чисел, не имеющим регулярности, не может быть представлена в памяти машины тем же простым способом, что и полная матрица. Если мы сохраняем численные значения коэффициентов нашей системы уравнений, мы должны вместе со значением каждого коэффициента хранить номер соответствующего уравнения и номер соответству-

ющего неизвестного. На жаргоне разреженных матриц мы можем сказать, что храним значения ненулевых элементов плюс индексная информация, указывающая расположение каждого элемента в *регулярном* массиве. Эта дополнительная информация составляет накладные расходы — цену, которую приходится платить за отказ от хранения нулей.

Матричные алгоритмы должны проектироваться таким образом, чтобы обрабатывались только ненулевые элементы и чтобы на основании предварительного знания о расположении ненулевых элементов избегались операции типа сложения с нулем или умножения на нуль. Таким образом, число операций, производимых машиной при исполнении алгоритма, пропорционально числу ненулевых элементов, а не числу всех элементов матрицы. Заметим, что было бы неправильно хранить все элементы, включая нули, а затем обходить операции с нулями с помощью оператора IF. Этот оператор будет без необходимости выполняться n^2 раз или более, превращая алгоритм в квадратичный [1]) по порядку n матрицы. Хороший алгоритм для разреженных матриц использует имеющиеся у него сведения о позициях ненулевых элементов, чтобы делать только необходимые операции. Если для разреженной матрицы число ненулевых элементов в строке постоянно [2]), то во многих случаях порядок хорошего алгоритма может уменьшаться до n.

Разреженная матрица, представляющая заданную информацию, имеет определенную заданную разреженность. Однако если разреженная матрица порождается неким алгоритмом как промежуточный результат в рамках более обширного счета, то возникает вопрос: нет ли способа увеличить ее разреженность путем генерирования меньшего числа ненулевых элементов? Наиболее важный случай, когда ответ утвердительный, — это гауссово исключение. В гауссовом исключении разреженность матрицы уменьшается по сравнению с исходной, поскольку возникают новые ненулевые элементы. Результирующая матрица менее разрежена, чем исходная, но степень ее заполнения сильно зависит от порядка, в каком выбираются главные элементы. Хороший алгоритм для разреженных матриц пытается сохранить разреженность, по возможности уменьшая заполнение.

Итак, мы сформулировали три основных идеи, которые направляли развитие большей части технологии разреженных матриц: хранить только ненулевые элементы, оперировать только с ненулевыми элементами и сохранять разреженность. Конечно, не все алгоритмы для разреженных матриц достигают этих целей, а только наиболее изощренные. Многие схемы хранения допу-

[1]) По меньшей мере. — *Прим. перев.*
[2]) И не зависит от n. — *Прим. перев.*

скают определенную долю нулей, и алгоритм обрабатывает их, как если бы они не были нулями. Алгоритм, хранящий и обрабатывающий меньшее число нулей, более сложен, труднее программируется и целесообразен только для достаточно больших матриц. Существует полный спектр алгоритмов, рассчитанных на различные типы матриц, от плотных до очень разреженных, с различными необходимыми для практики степенями эффективности, сложности или простоты.

Глава 1

Основы

1.1. ВВЕДЕНИЕ

В этой главе обсуждаются некоторые машинные методы с достаточно большой областью применений, которые можно рассматривать как основы технологии разреженных матриц. Глава начинается с описания структур и способов внутреннего представления, используемых при хранении числовых списков, графов, разреженных матриц различных типов, а также блочных разреженных матриц. Цель главы состоит в том, чтобы изложить лишь относящиеся к данному предмету идеи и проиллюстрировать их простыми примерами. Исследуются символическая и численная обработка разреженных матриц и динамическое распределение памяти. Наконец, обсуждается численная алгебра разреженных векторов — ведь строка разреженной матрицы является разреженным вектором, **а** алгебра разреженных матриц большей частью требует выполнения операций над разреженными векторами.

1.2. ХРАНЕНИЕ МАССИВОВ, СПИСКОВ, СТЕКОВ И ОЧЕРЕДЕЙ

Технология разреженных матриц часто требует хранения и обработки списков, элементами которых могут быть числа, целые, вещественные или комплексные, либо объекты более сложной структуры, такие, как матрица, массив, вершина графа вместе с соответствующими ребрами или переключатель. Вот примеры операций, обычно выполняемых над списками: добавление элемента в конец списка, удаление элемента из конца списка, вставка или удаление элемента в середине или начале списка, определение позиции некоторого элемента или элемента, следующего за данным, сортировка, упорядочение и т. д. Выбор схемы хранения зависит от операций, которые предполагается производить. Дело в том, что эффективность исполнения данной операции весьма различна для разных схем хранения.

Простейшая структура данных — это *массив,* с которым, полагаем, читатель достаточно знаком; примеры — A (I), B (I, J) и т. д. В массиве могут храниться просто числа. Другая возможность: массив содержит указатели на элементы более сложной природы, хранящиеся в действительности где-то в другом месте. Все элементы массива доступны непосредственно за время, не зависящее от его размера. Однако следует иметь в виду, что машинная память одномерна, а потому за использование двойных или кратных индексов приходится расплачиваться. Нужно заметить также, что в машинах с виртуальной памятью массив может находиться на периферийном запоминающем устройстве и не быть легко доступным.

Линейный *связный список* — это последовательность *ячеек,* связанных в том или ином порядке. Каждая ячейка содержит элемент списка и указатель, сообщающий положение следующей ячейки. Например, предположим, что мы хотим хранить числа *a, b, c* и *d* в указанном порядке в массиве A (I). Схема хранения может выглядеть так (символом *x* отмечены несущественные для нас значения):

```
позиция = 1 2 3 4 5 6 7 8
  A (I) = x b x d a x c x
 NEXT (I) = x 7 x 0 2 x 4 x
       IP = 5
признак конца = 0
```

В массиве A (I) хранятся элементы списка, а в NEXT (I)— указатели позиций следующих элементов. Необходим еще указатель входа IP, показывающий расположение первого элемента. В данном случае он равен 5. В позиции 5 находим первый элемент A (5) = *a,* а NEXT (5) = 2 сообщает, что следующий элемент нужно искать в позиции 2. Этим путем можно просмотреть весь

список. В последнюю ячейку должен быть помещен *признак конца,* указывающий окончание списка; в нашем примере таким признаком служит 0. Еще один способ состоит в том, чтобы хранить общее число элементов списка и с помощью этого числа определять, когда список исчерпан. Пустой список удобно задавать, присваивая указателю входа значение, которое не может адресовать какую-либо позицию массивов, например неположительное число.

Элементы легко вставляются или удаляются в любом месте. Предположим, например, что между *b* и *c* нужно вставить число *e.* Предположим еще, что известно: ячейка 3 пуста, а *b* находится в позиции 2. Следующая процедура выполняет требуемую операцию:

$$A(3) \leftarrow e$$

$$NEXT(3) \leftarrow NEXT(2)$$
$$NEXT(2) \leftarrow 3$$

Процедура удаления элемента (скажем, *c* из позиции 7 исходного связного списка) еще проще:

$$NEXT(2) \leftarrow NEXT(7)$$

Разумеется, для выполнения этой процедуры необходимо знать, что элемент, предшествующий *c,* расположен в позиции 2, так что фактически она удаляет «элемент, следующий за *b*», а не сам *c.* Если нужно вставить или удалить элемент в самом начале списка, следует переопределить указатель входа. Вставка или удаление элементов не изменяет порядка, в котором хранятся остальные элементы.

Если список хранится в массиве, важно иметь информацию о свободных позициях последнего. Обычно их также связывают в список, что требует еще одного указателя входа. Очевидно, что оба списка не пересекаются и потому могут быть хранимы в одних и тех же массивах. Нижеследующая структура данных получится, если связать свободные позиции в нашем примере (IE — указатель входа для нового списка):

```
позиция =  1  2  3 4 5 6 7 8
    A ( I ) = x  b  x  d  a  x  c  x
NEXT (I)  = 3  7  6  0  2  8  4  0
       IP = 5
       IE = 1
признак конца = 0
```

Процедуры вставки или удаления элемента теперь несколько усложняются; мы оставляем их читателю как упражнение.

Связный список становится *кольцевым связным списком,* если в его последнюю позицию вместо признака конца поместить указатель на начальную позицию. У кольцевого списка нет ни начала, ни конца, но он все же требует хранимого отдельно указателя входа, который теперь может указывать на любую занятую позицию. В нашем примере было бы NEXT (4) = 5, так что *a* следует за *d;* указатель входа мог бы иметь значение 7, что соответствовало бы списку *c, d, a, b.* Кольцевые списки не требуют признака конца. Окончание списка распознается благодаря тому, что 7 — значение указателя входа — повторяется в NEXT (2). В кольцевом списке элементы можно удалять или добавлять, не меняя порядка остальных, а свободные позиции связывать таким же образом, как в линейном списке.

Двунаправленный связный список, линейный или кольцевой, получится, если ввести еще массив, содержащий для каждой ячейки адрес предыдущей ячейки. Двунаправленный список можно просматривать в обоих направлениях; он имеет то достоинство, что можно вставить или удалить элемент, не зная позиции предыдущего элемента. Линейный двунаправленный список требует двух указателей входа: один указывает начало списка, другой — его конец. Для кольцевого двунаправленного списка достаточно одного указателя входа. Свободные позиции двунаправленного списка можно связать между собой, хотя для этой цели редко бывает нужен новый двунаправленный список.

Стек — это список с упрощенным способом хранения и обработки [Aho et al., 1976]. В стеке элементы хранятся в последовательных позициях, чем устраняется необходимость в связках. Для указания позиции последнего элемента (так называемого верха стека) используется специальный указатель. Стеки применяют в ситуациях, когда элементы нужно добавлять или удалять только через верх стека. Чтобы включить или *опустить* новый элемент в стек, увеличивают значение указателя на единицу, проверяют, достаточно ли памяти, а затем записывают элемент в позицию, сообщаемую указателем. Чтобы исключить или *поднять* последний элемент из верха стека, попросту уменьшают на единицу значение указателя. Пустой стек опознается по нулевому значению указателя. Стеки используются в нескольких алгоритмах для разреженных матриц. Примерами могут служить алгоритм Джорджа для древовидного разбиения, обсуждаемый в § 4.9, и алгоритм Тарьяна для блочной триангуляризации матрицы, рассматриваемый в гл. 5.

Очередь — это список элементов, хранимых в последовательных позициях, причем включение элементов производится через *начало* очереди, а исключение — через ее *конец* [Aho et al., 1976]. Для указания позиций начала и конца очереди используются два указателя. Пустая очередь опознается благодаря тому, что

для нее значение указателя конца на единицу больше значения указателя начала. Для очереди с единственным элементом значения обоих указателей совпадают. Если участок памяти, отведенный для хранения очереди, исчерпан, то новые элементы можно записывать в начало этого участка, закольцевав тем самым очередь; при этом начало очереди не должно пересекаться с ее концом.

Все описанные схемы хранения опираются на *массив* как единственную структуру данных, поддерживаемую Фортраном. Элегантные свойства связных списков можно эффективно реализовать, используя и такую структуру данных, как *запись,* допускаемую, например, языками Паскаль и Алгол *ω.* В этом случае не требуется косвенной адресации и память может распределяться динамически ценой некоторых системных затрат памяти. Применение записей для разреженных матриц обсуждается в [Houbak, 1981]; мы еще вернемся к этому вопросу в § 1.12.

1.3. ХРАНЕНИЕ ЦЕЛЫХ СПИСКОВ

Целые списки заслуживают специального рассмотрения из-за их важности для технологии разреженных матриц. Элементы списка принадлежат множеству (1, 2, ..., *n*); некоторые из них могут повторяться, хотя случай, когда все они различны, представляет для нас особый интерес. Пусть *m* — число элементов списка; *n* называется *размахом* [1]) списка. Будем говорить, что список *разреженный,* если $m < n$. Целый список можно хранить в *компактной* форме, пользуясь массивом (скажем, JA) длины, не меньшей *m*; так, для списка 3, 11, 7, 4:

```
позиция = 1  2  3 4 5 6
     JA = 3 11 7 4
      M = 4
```

Кроме массива JA нужно еще хранить (посредством переменной M) число *m* элементов списка; для нашего примера $m = 4$. Пустой список опознается по нулевому значению M. Чтобы включить в список новый элемент, достаточно увеличить M на единицу и затем записать новый элемент в позицию с номером M. Чтобы исключить элемент из позиции *i,* мы просто переносим в эту позицию последний элемент и уменьшаем M на единицу. Разумеется, если требуется сохранять упорядоченность целых чисел, то включение или исключение элемента в какой-то позиции вызовет сдвиг всех элементов, находящихся справа от нее.

Целые списки с повторениями или без них могут храниться как связные линейные или кольцевые списки посредством массива JA длины, не меньшей *m,* и дополнительного массива NEXT;

[1]) В оригинале the range. — *Прим. перев.*

см. обсуждение в § 1.2. Этот тип хранения также является компактным и потому вполне подходит для разреженных списков.

Для целых списков с различными элементами имеется важная альтернатива. Такой список может храниться одним массивом (скажем, JA) длины n или большей в форме связного списка, линейного или кольцевого. Если в приведенном выше примере использовать кольцевой связный список, а несущественные для нас позиции пометить символом x, то получим

позиция =	1	2	3	4	5	6	7	8	9	10	11	12
JA =	x	x	11	3	x	x	4	x	x	x	7	x
IP = 3												

Число, хранимое в каждой позиции JA, дает одновременно значение элемента списка и адрес следующего элемента; тем самым дополнительный массив NEXT становится излишним. Для входа в цепь необходим еще указатель IP, задающий позицию первого числа в списке. Этот тип хранения называют *расширенным,* в противоположность компактному, потому что для него нужен массив длины по крайней мере n. В любом месте списка несложно включить или исключить элемент, не меняя порядка, в котором хранятся прочие элементы. Предположим, например, что i, k — два последовательных элемента списка, находящегося в массиве JA, так что JA (i) = k. Пусть теперь требуется вставить j между i и k. Тогда мы просто полагаем

$$\text{JA}(j) \leftarrow \text{JA}(i)$$
$$\text{JA}(i) \leftarrow j$$

Если, наоборот, хотим исключить k, то полагаем

$$\text{JA}(i) \leftarrow \text{JA}(k)$$

Разумеется, если нужно включить элемент перед первым элементом или удалить первый элемент, то следует переопределить указатель входа. Список из одного элемента, скажем 5, при использовании расширенной схемы выглядит следующим образом:

позиция =	1	2	3	4	5	6	7	8	9
JA =	x	x	x	x	5	x	x	x	x
IP = 5									

Пустой список можно опознать по нулевому или отрицательному значению указателя входа.

Одно из главных применений расширенной схемы — это хранение нескольких *непересекающихся* списков, т. е. списков, не имеющих общих элементов. В большинстве случаев все списки имеют одинаковый размах, но чтобы наше рассуждение стало более общим, будем считать n максимальным из размахов отдель-

ных списков. Все списки можно хранить в одном и том же массиве (скажем, JA), длины *n* или большей. Необходим еще массив указателей входа (скажем, IP). Достаточно проиллюстрировать предлагаемый способ хранения примером. Рассмотрим три непересекающихся целых списка:

1-й список: 2, 8, 6, 3
2-й список: 5, 4, 9
3-й список: 7, 1, 10

Они могут быть размещены как кольцевые связные списки в одном массиве длины 10:

```
позиция =  1  2  3  4  5  6  7  8  9 10
    JA  = 10  8  2  9  4  3  1  6  5  7
    IP  =  2  5  7
```

При использовании этого типа хранения легко выполняются такие операции, как разбиение списка на два или конкатенация двух списков с образованием одного нового. В качестве упражнения читатель может попробовать написать соответствующие процедуры.

1.4. ПРЕДСТАВЛЕНИЕ И ХРАНЕНИЕ ГРАФОВ

На рис. 1.1 (а) изображен граф. Он состоит из множества *вершин,* в данном случае пяти, и множества *ребер,* в данном примере семи. Более точно, граф $G = (U, E)$ состоит из множества вершин U и множества ребер E; ребро можно отождествить с парой (u, v) различных вершин из V. Например, на рис. 1.1 *(а)* 3 есть вершина, а (3, 5) — ребро, изображаемое отрезком, соединяющим вершины 3 и 5. Графы очень важны для технологии разреженных матриц, поскольку существует соответствие между матрицами и графами, сохраняющее некоторые свойства. Конечно, графы имеют и много других полезных приложений. Более подробно графы будут рассмотрены в гл. 4 и 5.

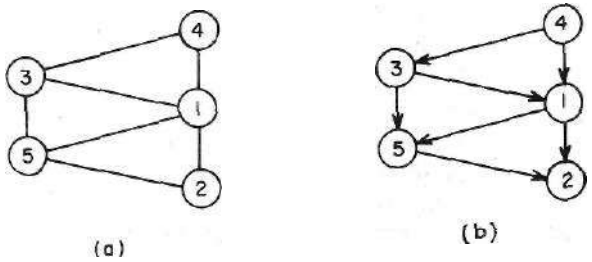


Рис. 1.1. Представление неориентированного графа (а) и ориентированного графа, или орграфа (Ь). Оба помечены.

Если пары *(u, v)* и *(v, u)* не различаются, то говорят, что ребра отождествляются с неупорядоченными парами, а граф называют *неориентированным.* Если же пары, представляющие ребра, упорядочены, то граф называют *ориентированным* графом, или *орграфом.* Если в орграфе *(u, v)* $\in E$, то говорят, что *v смежна* с *u.* Если *(u, v)* — ребро неориентированного графа, то говорят, что *v* смежна с *u* и *u* смежна с *v.* Неориентированный граф можно рассматривать как орграф, в котором, если (и, *v)* — ребро, то *(v, u)* также ребро. Чтобы изобразить граф с *n* вершинами, удобно *пометить* его, установив соответствие между вершинами и целыми числами 1, 2, ..., *n.* На рис. 1.1 показаны примеры неориентированного графа и орграфа; оба они помечены.

В памяти машины граф можно представить, храня для каждой вершины список вершин, смежных с ней. Такое множество списков называется *структурой смежности* графа [Tarjan, 1972]. Если списки хранятся в компактной форме, скажем в массиве LIST, то нужен еще массив указателей (скажем, IP) начала каждого списка. Для орграфа (b) на рис. 1.1 структура смежности такова:

```
позиция = 1 2 3 4 5 6 7 8
   LIST = 2 5 1 5 1 3 2
     IP = 1 3 3 5 7 8
```

Список смежности, например, для вершины 4 начинается с 5-й позиции массива LIST, что следует из значения 5 указателя IP (4), и кончается в 6-й позиции, потому что следующий список, т. е. список для вершины 5, начинается с позиции IP (5) = 7. Таким образом, с вершиной 4 смежны вершины 1 и 3. Заметим, что список для вершины 2 пуст, поскольку значения указателей IP (2) и IP (3) совпадают (оба равны 3). В конец массива IP включен дополнительный указатель; его смысл — первая свободная позиция массива LIST. Этот прием удобен при программировании.

Неориентированный граф рис. 1.1 (a) представляет следующая структура смежности:

```
позиция = 1 2 3  4  5  6 7 8 9 10 11 12 13 14 15
   LIST = 2 3 4  5  1  5 1 4 5  1  3  1  2  3
     IP = 1 5 7 10 12 15
```

Заметим, что каждое ребро представлено дважды. Компактное хранение списков смежности идентично строчной схеме хранения портрета разреженной матрицы, ассоциированной с данным графом. Эта схема будет рассмотрена в § 1.8.

При использовании этого типа хранения может оказаться удобным связать обе копии ребра с помощью дополнительного массива (скажем, LINK). Так, если одно и то же ребро хранится в позициях *i* и *j* массива LIST, то LINK (*i*) = *j* и LINK (*j*)= *i.*

Недостаток компактного хранения структуры смежности в том, что трудно включать и исключать ребра. Эта трудность преодолевается, если хранить структуру смежности как множество связных списков, линейных или кольцевых, вместе с массивом IP указателей входа в каждый список [Rheinboldt, Mesztenyi, 1973]. Среди списков обычно есть пересекающиеся, поэтому нельзя применить технику хранения, обсуждавшуюся в § 1.3, и нужен дополнительный массив NEXT. Используя кольцевые списки, орграф рис. 1.1 (b) можно представить следующим образом (несущественные для нас позиции помечены символом *x):*

позиция =	1	2	3	4	5	6	7	8	9
LIST =	3	5	1	2	*x*	5	1	*x*	2
NEXT =	3	7	1	6	*x*	4	2	*x*	9

IP = 4 0 7 3 9

Связное представление неориентированного графа на рис. 1.1 (a) может выглядеть так:

позиция =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LIST =	5	*x*	5	1	4	*x*	3	5	2	2	1	1	4	3	1	3	*x*
NEXT =	4	*x*	13	1	14	*x*	11	5	12	8	7	16	15	10	3	9	*x*

IP = 8 4 13 11 9

Заметим, что каждое ребро снова хранится дважды. Например, ребро (1,5) хранится в позициях 8 и 12, сначала как (1, 5), потом как (5, 1). Как указано выше, для связывания обеих копий каждого ребра можно использовать массив LINK. Если из неориентированного графа приходится удалять ребра, то могут пригодиться двунаправленные списки, описанные в § 1.2. Во всех случаях свободные позиции могут быть связаны между собой в целях последующего использования.

Еще одна распространенная схема хранения — это *таблица связей* [George, 1977, p. 64]. Если граф имеет n вершин и m — максимальная *степень* вершины (т. е. число смежных с ней вершин), то таблица связей представляет собой массив из n строк и m столбцов; при этом в i-й строке хранится список смежности вершины i. Для графа на рис. 1.1 (b) таблица связей состоит из 5 строк и двух столбцов:

	1	2
1	2	5
2	0	0
3	1	5
4	1	3
5	2	0

Наконец, для представления графа можно использовать *матрицу смежности* [Aho et al., 1976]. Для графа с n вершинами матрица смежности — это квадратная матрица A порядка n, определяемая следующим образом: $a_{ij} = 1$ тогда и только тогда, когда (i, j) — ребро данного графа; в противном случае $a_{ij} = 0$. Матрица смежности для графа на рис. 1.1 (b) имеет вид

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Матрица смежности неориентированного графа симметрична. Этот тип хранения обычно весьма неэкономичен, если матрица полностью записывается в память машины. Все же он может быть удобен для алгоритмов, которым часто требуется информация о наличии или отсутствии каких-то ребер. Матрицу смежности можно хранить и как разреженную матрицу, например, по строкам, не запоминая числовые значения ее элементов (см. § 1.8). В этом случае получится в точности компактная схема хранения структуры смежности, описанная в начале данного параграфа.

Клика — это граф, в котором всякая пара вершин соединена ребром. Ясно, что для задания клики достаточно хранить список ссылок на ее вершины, и нет необходимости запоминать какую-либо информацию об ее ребрах. Это свойство нашло важное применение в гауссовом исключении и будет еще обсуждаться в гл. 4.

1.5. ДИАГОНАЛЬНАЯ СХЕМА ХРАНЕНИЯ ЛЕНТОЧНЫХ МАТРИЦ

С ленточными матрицами связана простейшая и наиболее широко применяемая стратегия использования нулевых элементов матрицы. Матрицу A называют ленточной, если все ее ненулевые элементы заключены внутри ленты, образованной диагоналями, параллельными главной диагонали. Таким образом, $a_{ij} = 0$, если $|i - j| > \beta$, и $a_{k,\,k-\beta} \neq 0$, либо $a_{k,\,k+\beta} \neq 0$ хотя бы для одного значения k. Здесь β — *полуширина,* а $2\beta + 1$ — *ширина ленты. Лентой* матрицы A называется множество элементов, для которых $|i - j| \leq \beta$. Другими словами, для всякой строки i ленте принадлежат все элементы со столбцовыми индексами от $i - \beta$ до $i + \beta$, т. е. $2\beta + 1$ элементов. Это число может быть много меньше порядка матрицы.

Если матрица симметрична, то достаточно хранить ее *полуленту.* Верхняя полулента состоит из элементов, находящихся в верхней части ленты, т. е. таких, что $0 < j - i \leq \beta$; нижняя

полулента состоит из элементов нижней части ленты, т. е. таких, что $0 < i - j \leq \beta$; в обоих случаях в строке β элементов. Некоторые авторы называют лентой то, что мы зовем полулентой, и определяют ширину ленты как число β [Cuthill, 1972; George, 1977] или $\beta + 1$ [Smith et al., 1976]. Мы будем использовать в дальнейшем определения ленты, полуленты, ширины и полуширины ленты, приведенные выше.

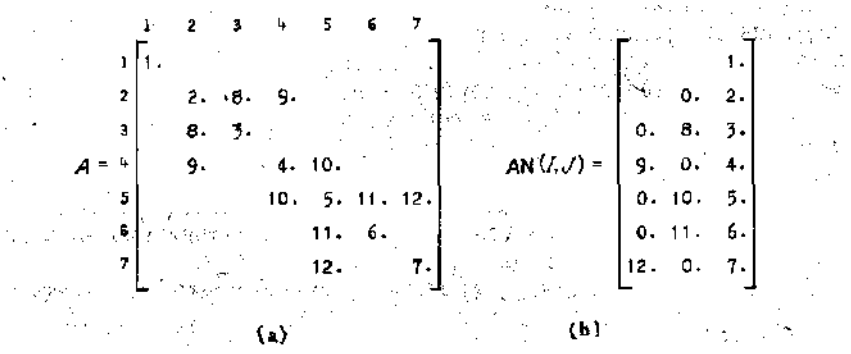


Рис. 1.2. Симметричная ленточная матрица 7×7 с шириной ленты 5 (a); диагональная схема ее хранения в прямоугольном массиве AN (I, J) с размерами 7×3 (b).

Диагональная схема хранения симметричной ленточной матрицы A посредством массива AN (I, J) иллюстрируется примером на рис. 1.2. Для матрицы порядка n и полуширины ленты β массив имеет размеры n x $(\beta + 1)$. Главная диагональ хранится в последнем столбце, а нижние кодиагонали — в остальных столбцах со сдвигом на одну позицию вниз при каждом смещении влево. В нашем примере $n = 7$, $\beta = 2$ и для массива нужна 21 ячейка памяти. Для несимметричной матрицы A необходим массив размером n x $(2\beta + 1)$; нижняя полулента и главная диагональ хранятся прежним образом, а верхние кодиагонали — в правой части массива со сдвигом на одну позицию вверх при каждом смещении вправо. Диагональная схема удобна, если $\beta << n$. Она является схемой прямого доступа в том смысле, что имеется простое взаимно однозначное соответствие между положением элемента в матрице A и его положением в массиве: a_{ij} хранится компонентой AN $(i, j - i + \beta + 1)$.

Ленточные матрицы имеют то важное свойство, что ширина ленты зависит от порядка, в каком расположены строки и столбцы. Поэтому можно искать перестановки строк и столбцов, приводящие к уменьшению ширины ленты. Малая ширина ленты означает снижение запросов к памяти; в вычислениях, связанных с матрицами, она обычно уменьшает и работу. В случае симме-

тричной матрицы ценное свойство симметрии будет сохранено, если для столбцов и строк используются одинаковые перестановки. Алгоритмы, уменьшающие ширину ленты, обсуждаются в гл. 4.

Если система линейных уравнений имеет ленточную матрицу коэффициентов и решается посредством гауссова исключения, причем главные элементы выбираются на главной диагонали, то вся арифметика ограничена лентой, вне которой никаких новых ненулевых элементов не возникает. Гауссово исключение можно проводить «на месте», поскольку для любого ненулевого элемента, если он появится, уже зарезервирована позиция в схеме хранения.

Собственные значения и собственные векторы ленточной матрицы, более того, собственные значения и собственные векторы обобщенной спектральной задачи с двумя ленточными матрицами одинаковой ширины, можно вычислить, не используя дополнительную память [1]). Это утверждение будет разъяснено в гл. 6.

1.6. ПРОФИЛЬНАЯ СХЕМА ХРАНЕНИЯ СИММЕТРИЧНЫХ МАТРИЦ

Ленточная матрица высокого порядка может иметь широкую ленту и большое число нулей внутри ее. Для такой матрицы диагональная схема может быть очень неэкономной. Дженнингс [Jennings, 1966] предложил более эффективную схему хранения симметричных матриц, и она вследствие своей простоты стала весьма популярной. Называется она *профильной схемой* [2]), или *схемой переменной ленты.* Для каждой строки i симметричной матрицы A положим

$$\beta_i = i - j_{min}\ (i)$$

где $j_{min}\ (i)$ — минимальный столбцовый индекс строки i, для которого $a_{ij} \neq 0$. Таким образом, первый ненулевой элемент строки i находится на β_i позиций левее главной диагонали. Определенная в § 1.5 полуширина ленты β есть просто $\max_i (\beta_i)$.

Оболочка матрицы A — это множество элементов a_{ij}, для которых $0 < i - j < \beta_i$ В строке i оболочке принадлежат все элементы со столбцовыми индексами от $j_{min}\ (i)$ до $i - 1$, всего β_i элементов. Диагональные элементы не входят в оболочку. *Профиль* матрицы A определяется как число элементов в оболочке:

$$\text{profile}\ (A) = \sum_i \beta_i.$$

---

[1]) Имеется в виду: помимо памяти, нужной для хранения самих собственных значений и векторов. — *Прим. перев.*

[2]) В оригинале — envelope scheme, т. е. буквально «оболочечная схема». Вследствие разъясняемой ниже тесной связи между понятиями оболочки и профиля мы сочли возможным заменить буквальный перевод более коротким термином. — ***Прим. перев.***

При использовании схемы Дженнингса все элементы оболочки, упорядоченные по строкам, хранятся, включая нули, в одномерном массиве, скажем AN. Диагональный элемент данной строки помещается в ее конец [1]). Длина массива AN равна сумме профиля и порядка A. Необходим еще массив указателей, назовем его IA; элементы этого массива суть указатели расположения диагональных элементов в AN. Так, при $i > 1$ элементы строки i находятся в позициях от IA (i — 1) + 1 до IA (i). Единственный элемент a_{11} 1-й строки хранится в AN (1). Элементы имеют последовательные легко вычисляемые столбцовые индексы. Например, для матрицы на рис. 1.2 (а) профиль равен 7, а профильная схема выглядит так:

позиция =	1	2	3	4	5	6	7	8	9	10	11	12	13	14
AN =	1.	2.	8.	3.	9.	0.	4.	10.	5.	11.	6.	12.	0.	7.
IA =	1	2	4	7	9	11	14							

Возможен вариант схемы Дженнингса с хранением оболочки по столбцам. Поскольку теперь столбцы матрицы сохраняют свои длины, эту схему часто называют вертикальной [2]). Упомянем еще одно полезное понятие, используемое при конструировании схем хранения симметричных матриц. Рассмотрим строку i матрицы A. Будем говорить, что столбец j, $j > i$, *активен* в этой строке, если он содержит ненулевой элемент в строке i или выше ее. Пусть ω_i — число столбцов, активных в строке i. Тогда $\max_i \omega_i$ называют *волновым фронтом* или *шириной фронта* A (см., например, [Cuthill, 1972]). В примере на рис. 1.2 (а) столбец 4 активен в строке 3, а ширина фронта A равна 2.

Как и в случае ленточных матриц, профиль обычно меняется при перестановках строк и столбцов. Меньший профиль означает меньшую память и меньшее число операций в вычислениях, выполняемых с матрицей, поэтому алгоритмы минимизации профиля играют важную роль в технологии разреженных матриц. Они обсуждаются в § 4.6.

Рассматривая элементы, заключенные в оболочке, можно заметить, что в гауссовом исключении значение β_i для каждой строки не изменится, если главные элементы выбирать на главной диагонали. Новые ненулевые элементы не могут возникнуть в позициях, внешних для оболочки. Они могут появиться внутри оболочки, где уже заготовлено место для них; отсюда следует, что исключение можно проводить при статической схеме хранения. Это важное свойство было исходной мотивировкой для схемы

[1]) Возможно хранение диагональных элементов отдельным массивом — см. [George, Liu, 1981]. — *Прим. перев.*

[2]) В оригинале — skyline storage. — *Прим. перев.*

Дженнингса, но ту же схему можно с выгодой применять и в других случаях. Например, она очень хорошо приспособлена для итерационных алгоритмов, где требуется эффективность при вычислении произведений A на векторы, но сама A не меняется. К этой категории относятся алгоритмы Ланцоша и сопряженных градиентов. В данном контексте понятие оболочки можно очевидным образом распространить на несимметричные матрицы.

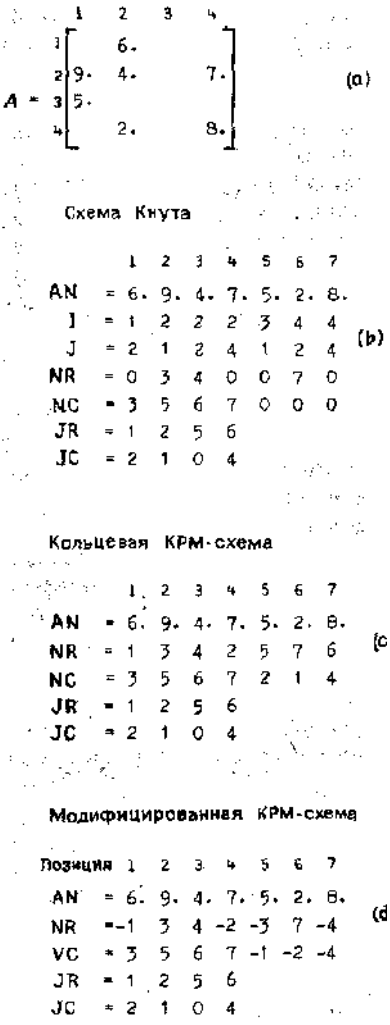
Схема переменной ленты строчно-ориентирована в том смысле, что любая строка матрицы сканируется эффективно; в то же время всякая попытка сканировать столбцы будет неэффективной. Кроме того, схема является статической, потому что включение нового элемента, лежащего вне оболочки, требует изменения всей структуры (если только не используются записи переменной длины).

1.7. СВЯЗНЫЕ СХЕМЫ РАЗРЕЖЕННОГО ХРАНЕНИЯ

Схемы хранения матриц, описанные в § 1.5, 1.6, дают прекрасные результаты во многих важных практических приложениях. Разреженная матрица A, которую можно переупорядочить так, чтобы она имела узкую ленту или малую оболочку, требует гораздо меньшей памяти, чем если бы она хранилась двумерным массивом, как это делают для плотных матриц. Хотя многие нули, находящиеся внутри ленты или оболочки, все же будут храниться и участвовать в операциях (или по крайней мере в проверках на нулевое значение), простота, свойственная этим схемам и связанному с ними программированию, может значительно перевесить это неудобство. Сравнение двух методов хранения показывает, что при использовании профильной схемы хранится меньшее число нулей; цена, которую приходится платить за это уменьшение, — необходимость иметь массив IA, содержащий лишь вспомогательную информацию, и вследствие этого усложнение программы. Дополнительная память называется *накладной.*

И трудности программирования, и накладная память возрастают параллельно с усложнением схемы хранения. Высокосложные схемы требуют подлинно мастерского программирования, иначе их потенциальные преимущества будут утрачены. С этой точки зрения простая схема может быть более подходящей для задачи малого или среднего размера. С другой стороны, возможность применения высокосложной схемы может определять, разрешима ли вообще большая задача на данной машине.

В этом и последующих параграфах будут изучены схемы, в которых нули вообще не хранятся, либо хранятся только «некоторые» нули. Рассмотрим разреженную матрицу A; без потери общности и для большей ясности предположим, что A очень велика. Ее ненулевые элементы, которых очень мало в сравнении с числом нулей, рассеяны по всей матрице, образуя то, что на-



```
        1   2   3   4
    1 |     6.          |
    2 | 9.  4.      7.  |          (a)
A = 3 | 5.              |
    4 |     2.      8.  |
```

Схема Кнута

```
          1   2   3   4   5   6   7
AN    = 6.  9.  4.  7.  5.  2.  8.
I     = 1   2   2   2   3   4   4      (b)
J     = 2   1   3   4   1   2   4
NR    = 0   3   4   0   0   7   0
NC    = 3   5   6   7   0   0   0
JR    = 1   2   5   6
JC    = 2   1   0   4
```

Кольцевая КРМ-схема

```
          1   2   3   4   5   6   7
AN    = 6.  9.  4.  7.  5.  2.  8.
NR    = 1   3   4   2   5   7   6      (c
NC    = 3   5   6   7   2   1   4
JR    = 1   2   5   6
JC    = 2   1   0   4
```

Модифицированная КРМ-схема

```
Позиция 1   2   3   4   5   6   7
AN    = 6.  9.  4.  7.  5.  2.  8.
NR    =-1   3   4  -2  -3   7  -4      (d
VC    = 3   5   6   7  -1  -2  -4
JR    = 1   2   5   6
JC    = 2   1   0   4
```

Рис. 1.3. Три схемы хранения разреженных матриц, использующие связные списки для строк и столбцов.

зывают *портретом* матрицы или *шаблоном нулей—ненулей.* Мы опишем прежде всего схему хранения разреженной матрицы, предложенную Кнутом [Knuth, 1968]. Ненулевые элементы хранятся в компактной форме и в произвольном порядке в одномерном массиве, скажем AN. Информация о положении ненулевых элементов может храниться двумя дополнительными параллельными одномерными массивами, скажем I и J; здесь для каждого ненулевого элемента содержатся его строчный и столбцовый индексы. Итак, для каждого $a_{i\,j} \neq \neq 0$ в памяти находится тройка $(a_{i\,j}, i, j)$. Далее, чтобы можно было легко отыскивать элементы произвольной строки или столбца матрицы, необходимы еще пара указателей для каждой тройки, а также *указатели входа* для *строк* и *столбцов,* сообщающие начало каждого строчного или столбцового списка. Пусть NR («next nonzero element in the same row» — «следующий ненулевой элемент той же строки») — массив, хранящий строчные указатели, а NC («next nonzero element in the same column» — «следующий ненулевой элемент того же столбца») — массив столбцовых указателей. Пять массивов AN, I, J, NR и NC имеют одинаковую длину, и их одноименные позиции соответствуют друг другу. Пусть JR и JC — массивы, содержащие указатели входа для строк и столбцов, расположенные в соответствии с порядком строк и столбцов матрицы. На рис. 1.3 (b) показан пример матрицы *A,* хранимой согласно схеме Кнутач Если нам нужны, например, элементы 2-го столбца, то, заметив, что JC (2) = 1, начнем с 1-й позиции массивов AN, I и NC. Видим, что AN (1) = 6 и I (1) = 1, т. е. эле-

мент 6. находится во 2-м столбце, 1-й строке. Поскольку NC (1) = = 3, то затем мы переходим в позицию 3 и находим элемент 4. во 2-й строке. Так как NC (3) = 6, мы переходим в позицию 6 и находим элемент 2 в 4-й строке. Больше элементов во 2-м столбце нет, потому что NC (6) = 0. Заметим, что JC (3) = 0; это означает, что 3-й столбец пуст. При пользовании данной схемой элемент $a_{i\,j}$ можно найти, лишь входя в список i-й строки и просматривая его до тех пор, пока не будет найден столбцовый индекс j, либо то же самое с переменой ролей строки и столбца. Обратная задача имеет очевидное решение: если задан номер позиции k элемента AN *(k),* то его строчный и столбцовый индексы суть I *(k) и* J *(k).*

Схема Кнута требует пяти ячеек памяти для каждого ненулевого элемента A плюс к этому указатели входа для строк и столбцов. Вследствие большой накладной памяти такая схема весьма неэкономна. Достоинства ее в том, что в любом месте можно включить или исключить элемент, и можно эффективно сканировать строки и столбцы. Для этих целей используется описанная в § 1.2 техника обработки связных списков. Записи, имеющие фиксированную длину, могут быть размещены в произвольных областях физической памяти и затем связаны между собой; свободные записи также связываются для последующего использования. Схема идеально приспособлена для случаев, когда A строится каким-то алгоритмом, где нельзя предсказать конечное число и позиции ненулевых элементов. Наиболее важный пример такого алгоритма — гауссово исключение для матриц общего вида.

Рейнболдт и Местеньи [Rheinboldt and Mesztenyi, 1973] предложили модификацию схемы Кнута, сохраняющую ее ценные свойства, но использующую значительно меньше накладной памяти. Мы будем называть эту модификацию схемой Кнута—Рейнболдта—Местеньи или *кольцевой КРМ-схемой.* Связные списки строк и столбцов закольцовываются, а начальные позиции списков включаются в указатели входа. Списки, ассоциированные со строками (столбцами), попарно не пересекаются и потому могут быть совместно хранимы одним массивом NR (для столбцов NC), как это описано в § 1.3. Пример приведен на рис. 1.3 (с). Эта схема более плотная по сравнению со схемой Кнута. Однако нужно уяснить, что если приходится просматривать элементы некоторой строки (или столбца), то мы не получаем никакой информации о столбцовых (строчных) индексах этих элементов. Рассмотрим теперь, как можно найти элемент $a_{i\,j}$. Сначала сканируется *i-я* строка и определяется множество S_i позиций элементов этой строки в массиве AN:

$$S_i = \{p \mid \text{AN } (p) \text{ есть элемент } i\text{-й строки}\}.$$

Подразумевается, что повторения отсутствуют, т. е. каждой паре $p,\ q$ строчного и столбцового индексов соответствует самое боль-

шее одна позиция в каждом из массивов AN, NR и NC. Далее сканированием j-го столбцового списка определяется множество S_j позиций массива AN, где хранятся элементы j-го столбца. Наконец, находим $S_{i\ j} = S_i \cap S_j$ (символ $\cap$ означает пересечение; элемент принадлежит $S_{i\ j}$ тогда и только тогда, когда он принадлежит и S_i, и S_j). $S_{i\ j}$ либо пусто, либо содержит только один элемент. Если $S_{i\ j}$ пусто, то $a_{i\ j} = 0$. Если $S_{i\ j}$ содержит одно число, скажем, k, то $a_{i\ j} =$ AN *(k)*. Обратная задача, т. е. по заданной позиции k в AN найти соответствующие строчный и столбцовый индексы i и j, не имеет иного решения как просмотр всей матрицы.

Вариант КРМ-схемы [Duff, 1977, p. 517], позволяющий находить нужные индексы, показан на рис. 1.3 (d). Указатели входа для строк и столбцов по существу *включаются* в соответствующие кольцевые списки. Есть несколько способов добиться этого; способ, иллюстрируемый рис. 1.3 (d), использует отрицательные числа для ссылок на указатели входа строк и столбцов. В этой схеме достаточно просматривать ряд (ряд матрицы — это строка либо столбец) до тех пор, пока не будет найдена отрицательная ссылка. Абсолютная ее величина есть номер ряда; в то же время она отсылает к соответствующему указателю входа, так что просмотр ряда при необходимости может продолжаться. Предположим, например, что мы хотим найти строчный индекс элемента AN(3). Так как NR(3) = 4 и NR(4) == —2, то искомый индекс равен 2. Кроме того, JR(2) = 2, а потому AN(2) содержит следующий элемент 2-й строки. Чтобы найти в массиве AN элемент $a_{i\ j}$, можно применить описанную выше процедуру. Другая возможность состоит в том, чтобы просмотреть i-ю строку и найти столбцовый индекс каждого элемента, сканируя соответствующие столбцовые списки, пока не встретится j. Подпрограмма MA18 из библиотеки Харуэлла [Curtis, Reid, 1971a] использует многие из этих приемов.

Еще один вариант схемы Кнута использовался Ларкумом [Larcombe, 1971] для хранения симметричных положительно определенных матриц, элементами которых могут быть как числа, так и подматрицы. Называется эта схема *вперед по строке — назад по столбцу*. Хранятся только диагональ и верхний треугольник матрицы; нужен лишь один массив указателей входа, которые отсылают к диагональным элементам. От каждого диагонального элемента начинаются два списка: один описывает часть соответствующей строки вправо от диагонали и проходится в прямом направлении, другой описывает часть соответствующего столбца над диагональю и проходится в обратном направлении, т. е. снизу вверх. Это объясняет название схемы. Пример показан на рис. 1.4. Идею схемы легко распространить на матрицы общего вида. Если существует диагональ из ненулевых элементов (так называемая трансверсаль; см. § 5.7), то от каждого диагональ-

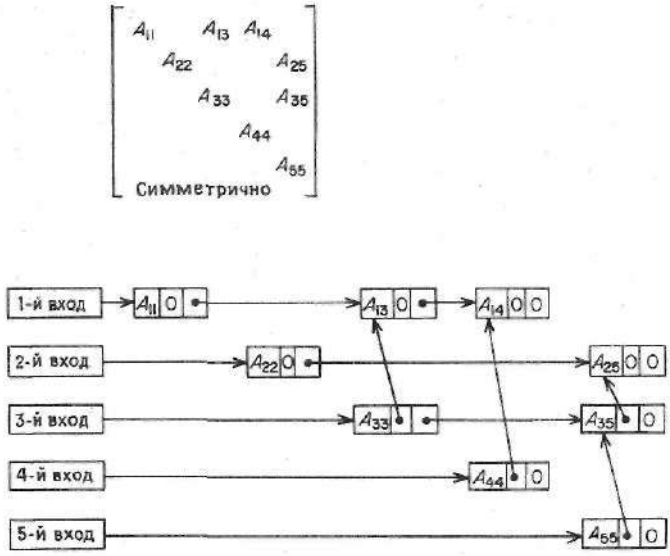


Рис. 1.4. Модификация Ларкума схемы Кнута для хранения симметричных матриц с ненулевыми диагональными элементами.

ного элемента могут начинаться четыре списка. С другой стороны, хватило бы и двух неупорядоченных списков: как объясняется в § 1.9, при работе с разреженными матрицами упорядоченные представления не всегда желательны.

1.8. РАЗРЕЖЕННЫЙ СТРОЧНЫЙ ФОРМАТ

Разреженный строчный формат [Chang, 1969; Gustavson, 1972], который будет сейчас описан,—это одна из наиболее широко используемых схем хранения разреженных матриц. Эта схема предъявляет минимальные требования к памяти и в то же время оказывается очень удобной для нескольких важных операций над разреженными матрицами: сложения, умножения, перестановок строк и столбцов, транспонирования, решения линейных систем с разреженными матрицами коэффициентов как прямыми, так и итерационными методами и т. д. Значения ненулевых элементов матрицы и соответствующие столбцовые индексы хранятся в этой схеме по строкам в двух массивах; назовем их соответственно AN и JA. Используется также массив указателей (скажем, IA), отмечающих позиции массивов AN и JA, с которых начинается описание очередной строки. Дополнительная компонента в IA содержит указатель первой свободной позиции в JA и AN. Здесь уместно привести пример. Рассмотрим матрицу

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{bmatrix} 0 & 0 & 1. & 3. & 0 & 0 & 0 & 5. & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7. & 0 & 1. & 0 & 0 \end{bmatrix} \end{array}$$

A представляется следующим образом:

```
позиция = 1  2  3  4  5  6
     IA = 1  4  4  6
     JA = 3  4  8  6  8      RR (C) O
     AN = 1. 3. 5. 7. 1.
```

Описание 1-й строки A начинается с позиции IA (1) = 1 массивов AN и JA. Поскольку описание 2-й строки начинается с IA (2) = = 4, это означает, что 1-й строке в AN и J A отвечают позиции 1, 2 и 3. Вообще в нашем примере

IA (1) = 1 первая строка начинается с JA (1) и AN (1).

IA (2) = 4 вторая строка начинается с JA (4) и AN (4).

IA (3) = 4 третья строка начинается с JA (4) и AN (4). Поскольку это те же позиции, с которых начинается 2-я строка, это означает, что. 2-я строка пуста.

IA (4) = 6 это первая свободная позиция в JA и AN. Таким образом, описание 3-й строки заканчивается в позиции 6 — 1 = 5 массивов JA и AN.

В общем случае описание r-й строки A хранится в позициях с IA (r) до IA (r + 1) — 1 массивов JA и AN, за исключением равенства IA (r + 1) = IA (r), означающего, что r-я строка пуста. Если матрица A имеет m строк, то IA содержит m + 1 позиций.

Данный способ представления называют *полным,* поскольку представлена вся матрица A, и *упорядоченным,* поскольку элементы каждой строки хранятся в соответствии с возрастанием столбцовых индексов. Таким образом, это строчное представление, полное и упорядоченное, или сокращенно RR (C) O.[1])

Массивы IA и JA представляют портрет A, задаваемый как множество списков смежности ассоциированного с A графа. Если в алгоритме разграничены этапы символической обработки и численной обработки (см. § 1.12), то массивы IA и JA заполняются на первом этапе, а массив AN — на втором.

Густавсон [Gustavson, 1972] предложил еще вариант строчной схемы хранения, приспособленный для приложений, где приходится выполнять операции и над строками, и над столб-

[1]) От английского словосочетания «Row-wise Representation Complete and Ordered». — ***Прим. перев.***

цами. A хранят по строкам, как описано выше; кроме того, определяют портрет A^T и также хранят по строкам. Строчное представление портрета A^T равносильно столбцовому представлению портрета A. Его можно получить транспонированием строчного портрета A (гл. 7). Эта схема применялась при решении задач линейного программирования [Reid, 1976].

Для несимметричных матриц была предложена и другая, гораздо более простая строчная схема [Key, 1973]. Ненулевые элементы хранят в двумерном массиве с размерами n x m, где n — порядок матрицы, m — максимальное число ненулевых элементов в строке. Эта схема допускает простую обработку; недостаток ее в том, что nr может быть неизвестно заранее, а может оказаться слишком большим.

1.9. УПОРЯДОЧЕННЫЕ И НЕУПОРЯДОЧЕННЫЕ ПРЕДСТАВЛЕНИЯ

Представления разреженных матриц необязательно должны быть упорядочены в том смысле, что, хотя упорядоченность строк поддерживается, внутри каждой строки элементы могут храниться в произвольном порядке. Для матрицы A нашего примера вполне можно было бы использовать и строчное представление, полное, но неупорядоченное (RR (C) U [1]), например, такое:

```
позиция = 1  2  3  4  5
   IA = 1  4  4  6
   JA = 8  3  4  8  6    RR(C)U
   AN = 5. 1. 3. 1. 7.
```

Неупорядоченные представления могут быть очень удобны. Результаты большинства матричных операций получаются неупорядоченными, и упорядочение их стоило бы больших затрат машинного времени. В то же время, за немногими исключениями, алгоритмы для разреженных матриц не требуют, чтобы представления были упорядоченными.

Используются также столбцовые представления, хотя они могут рассматриваться как строчные представления транспонированных матриц. Столбцовая схема для матрицы A нашего примера может выглядеть так:

```
позиция = 1  2  3  4  5 6 7 8 9 10 11
  IAT = 1  1  1  2  3 3 4 4 6  6  6
  JAT = 1  1  3  1  3                  CR(C)O [2])
  ANT = 1. 3. 7. 5. 1.
```

[1]) От английского словосочетания «Row-wise Representation Complete and Unordered». — *Прим. перев.*

[2]) От английского словосочетания «Column-wise Representation Complete and Ordered» — столбцовое представление, полное и упорядоченное. — ***Прим. перев.***

Одновременно это и схема RR (C) O для матрицы A^T. Заметим, что в массиве IAT 11 позиций, поскольку в A 10 столбцов.

Если заданная матрица симметрична, достаточно хранить лишь ее диагональ и верхний треугольник. Рассмотрим следующий пример:

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 2. & 0 & 0 & 1. \\ 0 & 1. & 0 & 1. \\ 0 & 0 & 3. & 0 \\ 1. & 1. & 0 & 3. \end{bmatrix} \end{array}$$

Для матрицы B строчное представление диагонали и верхнего треугольника (упорядоченное) таково:

```
позиция = 1  2  3  4  5  6 7
     IB = 1  3  5  6  7
     JB = 1  4  2  4  3  4          RR(DU)O[1]
     BN = 2. 1. 1. 1. 3. 3.
```

Если большинство диагональных элементов заданной симметричной матрицы отличны от нуля (как в случае симметричных положительно определенных матриц [2]), то они могут храниться в отдельном массиве BD, а разреженным форматом представляется только верхний треугольник B:

```
позиция = 1  2  3  4  5
     IB = 1  2  3  3  3
     JB = 4  4           RR(U)O[3]
     BN = 1. 1.
     BD = 2. 1. 3. 3.
```

Это очень плотное и удобное представление.

1.10. СЖАТИЕ ПО ШЕРМАНУ

В этом параграфе мы обсудим компактную схему хранения, предложенную Шерманом [Sherman, 1975]; она является вариантом описанного в § 1.8 разреженного формата. Схема Шермана

---

[1]) От английского словосочетания «Row-wise Representation, Diagonal and Upper, Ordered». — *Прим. перге,*

[2]) Где *все* диагональные элементы положительны. — *Прим. перев.*

[3]) От английского словосочетания «Row-wise Representation, Upper, Ordered». — *Прим. ruрее.*

используется для хранения разреженных треугольных матриц, вычисляемых в методе Гаусса. Чтобы объяснить схему, мы должны сперва определить *нижнюю треугольную матрицу* как матрицу, в которой ненулевые элементы могут стоять только в нижнем треугольнике и на главной диагонали, и *верхнюю треугольную матрицу* как матрицу с ненулевыми элементами только в верхнем треугольнике и на главной диагонали. Итак,

нижняя треугольная матрица: $a_{ij} = 0$ при $j > i$,
верхняя треугольная матрица: $a_{ij} = 0$ при $j < i$.

Более подробно треугольные матрицы обсуждаются в гл. 9. Наиболее важным их приложением является *треугольная факторизация*[1)] матрицы A:

$$A = LDU,$$

где L — нижняя треугольная, U — верхняя треугольная, D — диагональная матрицы. Треугольное разложение будет рассмотрено в гл. 2. Если A — разреженная матрица, то L и U, скорей всего, тоже будут разрежены.

L, D и U получаются из A посредством некоторого процесса исключения, например гауссова исключения. Часто случается, что некоторые множества строк в U или L имеют сходную структуру. В частности, нередко будет так, что строки U с номерами i и j, $i < j$, имеют идентичную структуру справа от позиции j. Чтобы избежать повторной записи одинаковых последовательностей столбцовых индексов, Шерман предложил компактную схему хранения этих индексов, требующую дополнительного массива указателей. Идею схемы иллюстрирует следующий пример. Рассмотрим верхнюю треугольную матрицу:

$U =$

	1	2	3	4	5	6	7	8	9	10
1	1.		11.				12.			13.
2		2.				14.			15.	
3			3.				16.			17.
4				4.					18.	
5					5.		19.			20.
6						6.		21.	22.	23.
7							7.	24.	25.	
8								8.	26.	
9									9.	27.
10										10.

[1]) Или треугольное разложение. — *Прим. перев.*

Эта матрица будет храниться так:

```
позиция = 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
UD =      1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
UN =      11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27.
IU =      1  4  6  8  9  11 14 16 17 18 18
JU =      3  7  10 6  9  8  9  10 8  9  10
IJ =      1  4  2  5  2  6  9  10 11
```

Диагональные элементы U хранятся в массиве UD, а внедиагональные элементы по строкам — в массиве UN. Массив IU содержит указатели для UN, определяемые обычным образом. Дополнительная компонента IU (11) указывает первую свободную позицию в UN, в данном случае позицию 18. Столбцовые индексы, отвечающие находящимся в UN ненулевым элементам, хранятся в JU в компактной форме. Теперь между UN и JU нет прямого соответствия. Столбцовые индексы выбираются из JU с помощью дополнительного массива указателей IJ. Несколько строк описываются в JU одним и тем же набором столбцовых индексов. Например, строки 1, 3 и 5 имеют одинаковую структуру справа от столбца 5; строку 1 обслуживают индексы 3, 7, 10; в то же время индексы 7 и 10 обслуживают и строки 3, 5. Таким образом, IJ (1) = 1, IJ (3) = IJ (5) = 2. Если мы хотим выбрать из представления полную строку, скажем 1-ю, то действуем обычным образом. Численные значения элементов находятся в UN в позициях от IU (1) = 1 до IU (2) — 1 = 3. Столбцовые индексы находятся в JU, начиная с позиции IJ (1) = 1. Поскольку уже известно, что нужны три столбцовых индекса, мы можем последовательно выбрать их из JU.

Схему хранения можно было бы сжать еще сильнее, если бы мы заметили, что строка 7 имеет столбцовые индексы 8 и 9, являющиеся одновременно первыми индексами строки 6. Обычно такое расположение индексов не используют, поскольку при небольшой экономии памяти затраты вычислительной работы здесь велики. Алгоритм исключения систематически порождает наборы строк, имеющих одинаковую структуру справа от некоторого столбца, тем самым предоставляя непосредственно информацию, необходимую для построения компактной схемы. Еще большее сжатие схемы потребовало бы дорогостоящего поиска. Сравнительные данные об обычной и компактной схемах приведены в [George, Liu, 1981, гл. 9]. Систематическая процедура выявления изоструктурных строк будет описана в § 4.8.

1.11. ХРАНЕНИЕ БЛОЧНЫХ МАТРИЦ

Идея разбиения большой матрицы на подматрицы или блоки возникает естественным образом. С блоками можно обращаться, как если бы они были элементами матрицы, и блочная матрица

превращается в матрицу из матриц. Разбиение на блоки играет важную роль в технологии разреженных матриц, потому что многие алгоритмы, сконструированные первоначально для числовых матриц, можно обобщить на случай матриц из матриц. Большая гибкость, связанная с понятием разбиения, может давать определенные вычислительные преимущества. С другой стороны, разбиение можно рассматривать просто как инструмент управления данными, помогающий организовать обмен информацией между оперативной памятью и внешними запоминающими устройствами.

Разбиения строк и столбцов не обязаны быть одинаковыми, но если они одинаковы, то диагональные блоки квадратные. Хранение блочной матрицы подразумевает хранение множества ее подматриц. Мы опишем прежде всего *неявную схему хранения* [George, 1977], предназначенную главным образом для симметричных матриц с квадратными диагональными блоками. Диагональные блоки рассматриваются так, как если бы они составляли единую матрицу, и хранятся в соответствии с профильной схемой Дженнингса из § 1.6. Численные значения элементов хранятся по строкам в массиве AN, а указатели на диагональные элементы каждой строки — в целом массиве IA. Поддиагональные блоки, рассматриваемые так, как если бы они составляли единую матрицу, хранятся посредством разреженного строчного формата из § 1.8. Ненулевые элементы размещаются по строкам в вещественном массиве AP, а соответствующие столбцовые индексы — в параллельном целом массиве JA. Указатели начала строк в AP и JA хранятся в массиве IP; в последний (для удобства программирования) включен добавочный указатель первой свободной позиции в AP и JA. Само разбиение определяется номерами первых строк каждого блока, хранимыми в LP, где тоже имеется добавочная компонента. Массив LP можно использовать совместно как с IA, так и с IP. Эта гибридная схема хранения с успехом применялась для гауссова исключения в комбинации с такими алгоритмами упорядочения, как древовидное разбиение и сечения (которые будут описаны соответственно в § 4.9 и 4.10).

Пример симметричной блочной матрицы A показан на рис. 1.5 (a). Строки и столбцы A сгруппированы в подмножества (1, 2, 3, 4), (5, 6, 7) и (8, 9, 10). В результате A разделилась на 9 блоков. Диагональные блоки квадратные, потому что строки и столбцы разбивались одинаковым образом. На рис. 1.5 (b) A представлена как 3×3 матрица из матриц; на рис. 1.5 (c) показана соответствующая неявная схема хранения.

Другая схема, называемая *блок внутри блочного* столбца, предложена Джорджем [George, 1977]; она особенно подходит для методов сечений из гл. 4. Диагональные блоки хранятся как прежде, а поддиагональные блоки трактуются как блочные столбцы. В результате сечений ненулевые элементы каждого блочного

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \left[\begin{array}{cccc|cccccc} 1. & & 3. & & 17. & & & & & \\ & 2. & 4. & 6. & & & & 21. & & \\ 3. & 4. & 5. & & & 18. & & & & 24. \\ \hline & 6. & & 7. & & 19. & 20. & & & \\ 17. & & & & 8. & & 10. & & 23. & \\ & & 18. & 19. & & 9. & & 22. & & \\ & & & 20. & 10. & & 11. & & & 25. \\ \hline & 21. & & & & 22. & & 12. & 13. & 15. \\ & & & & 23. & & & 13. & 14. & \\ & & 24. & & & & 25. & 15. & & 16. \end{array}\right] \end{array} \qquad (a)$$

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \qquad (b)$$

Позиция :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
AN=	1.	2.	3.	4.	5.	6.	0.	7.	8.	9.	10.	0.	11.	12.	13.	14.	15.
IA =	1	2	5	8	9	10	13	14	16	19							
AP=	17.	18.	19.	20.	21.	22.	23.	24.	25.								
JA=	1	3	4	4	2	6	5	3	7								
IP =	1	1	1	1	1	2	4	5	7	8	10						
LP=	1	5	8	11													

(c)

Рис. 1.5. Блочная матрица и неявная схема ее хранения.

столбца группируются в блоки. Эти блоки хранятся последовательно в вещественном массиве вместе с информацией о начале каждого блока (это номер первой его строки) и о блочном столбце, которому принадлежит данный блок.

Гиперматричная схема получается, если A хранится как матрица, элементами которой снова являются матрицы [Argyris, 1960, 1964; Von Fuchs et al., 1972]. Уже давно понято, насколько полезна эта схема, когда многие из блоков нулевые [Swift, I960]. Для хранения A можно использовать любой из методов, обсуждаемых в данной главе; отличие будет только в том, что вместо численных значений элементов нужно хранить информацию о расположении и размерах подматриц. Сами подматрицы, элементами которых будут уже числа, хранятся в соответствии с некоторым стандартным форматом. Особый интерес представляет случай, когда A, матрица из подматриц, разрежена. Тогда для. A можно применить разреженное представление, сохраняемое в оперативной памяти, в то время как подматрицы, так же как в некотором разреженном представлении, размещаются в периферийной памяти и вносятся в оперативную, только когда этого требует выполнение алгоритма. Этот вариант называется *сверхразреженной схемой* [Pissanetzky, 1979]. Все гиперматричные схемы позволяют обработку очень больших задач при умеренных запросах к памяти. Их главное достоинство состоит в легкости, с какой пользователь может уменьшить память или время (ценой увеличения другой из этих характеристик), для чего достаточно, сообразуясь с имеющейся памятью, задать более мелкое или более грубое разбиение. Эта техника успешно использовалась в гауссовом исключении и при решении спектральных задач. Она предполагает, что для разреженных матриц (прямоугольных, квадратных и треугольных) и векторов эффективно реализованы такие операции, как сложение, умножение, перестановки, а также факторизация квадратных диагональных блоков. Эти операции подробно обсуждаются в гл. 7 и 9.

1.12. СИМВОЛИЧЕСКАЯ ОБРАБОТКА И ДИНАМИЧЕСКИЕ СХЕМЫ ХРАНЕНИЯ

Алгоритм для разреженных матриц может порождать новые ненулевые элементы и изменять значения тех, что уже имеются в данной матрице; или же он может пользоваться этой матрицей, не меняя ее, как обстоит дело в случае итерационных алгоритмов, которым нужны лишь произведения матрицы с векторами. Множество новых ненулевых элементов, добавляемых к уже существующей разреженной матрице, называется *заполнением.* Алгоритм может строить и совершенно новую матрицу, но концептуально это равносильно генерированию ненулевых элементов для

первоначально нулевой матрицы. В этом параграфе речь будет идти об алгоритмах, создающих заполнение, и о том, как можно хранить это заполнение.

Прежде чем начнется выполнение такого алгоритма, нужно убедиться, что в памяти есть место для новых элементов. Должны быть также установлены законы управления памятью, определяющие внутреннее представление данных, или *структуру данных;* от них зависит, где и как будет храниться каждый новый элемент. Для структуры данных имеется выбор: она может быть сформирована до того, как начнется численная обработка, либо строиться параллельно с обработкой в соответствии с потоком вычисляемых элементов.

Структура данных, заготовленная до начала численной обработки, называется *статической* структурой. Ее формирование требует знания числа ненулевых элементов и их положения в матрице еще до того, как они будут реально вычислены. Ряд матричных алгоритмов позволяет предсказание необходимой информации. В эту категорию попадают сложение и умножение матриц, а также гауссово исключение, когда последовательность главных элементов известна a priori, как будет в случае симметричных положительно определенных матриц. Всякий такой алгоритм естественно распадается на две части: *символическую часть,* где выполняется *символическая обработка (или распределение памяти,* или *формирование структуры данных),* и *численную часть,* где выполняются реальные вычисления (или *численная обработка).* Результат символического этапа — статическая структура данных; результат численного этапа — значения ненулевых элементов, размещенные в ячейках памяти, которые были заготовлены йа символическом этапе.

Некоторые статические схемы в некоторых конкретных приложениях могут не требовать явного символического этапа. Ленточные и профильные схемы были сконструированы таким образом, что гауссово исключение с выбором главных элементов на диагонали может порождать ненулевые элементы лишь внутри ленты или оболочки; следовательно, эти схемы и являются нужными статическими структурами. То же справедливо при приведении ленточной матрицы к трехдиагональному виду в ходе решения спектральных задач (§ 6.6); но к профильной схеме хранения это не относится. Ясно, что сложение двух матриц с одинаковой лентой или оболочкой сохраняет ленту или оболочку, но умножение двух ленточных матриц расширяет ленту. Если гауссово исключение проводится для ленточной матрицы и при выборе главного элемента допускаются только перестановки строк, то нижняя полулента сохраняется, а верхняя полулента может в худшем случае удвоить свою ширину [Martin, Wilkinson, 1967]. Тьюарсон [Tewarson, 1971] предложил ряд статических схем, называемых

желательными формами, для использования в гауссовом исключении симметричных или несимметричных матриц.

С другой стороны, использование метода типа алгоритма минимизации ленты или профиля должно рассматриваться как подлинно символический этап; ведь такие алгоритмы только распределяют память, имея в виду ее уменьшение и соответствующее уменьшение работы, но не находят никаких численных результатов. К той же категории относятся некоторые другие методы — сечения, древовидное разбиение, блочная триангуляризация, — описываемые в гл. 4 и 5. Если матрица хранится в разреженном строчном формате (§ 1.8), то необходимым символическим этапом будет вычисление IA, JA, т. е. портрета матрицы; оно будет обсуждаться в связи с конкретными алгоритмами: умножения, транспонирования, треугольного разложения и т. д.

Статические схемы поощряют модульность, поскольку символический и численный этапы выполняются раздельно и, следовательно, могут быть оптимизированы независимо. Далее, в случае прямых методов для линейных систем сейчас существуют эффективные процедуры, позволяющие выполнять символический этап гораздо быстрее, чем численный. Они используют фиксированную память, лишь незначительно превосходящую ту, что требуется для хранения матрицы, и тем не менее дают существенную для пользователя информацию: количество памяти и вычислительной работы, необходимые для численного этапа. Еще одно важное достоинство модульности выявляется в приложениях, требующих повторного использования данного алгоритма при различных численных значениях. Примерами могут служить итерационные методы, нуждающиеся в решении многих линейных систем с одной и той же матрицей и различными правыми частями, или с матрицами, имеющими (при различных численных значениях) одинаковый портрет. В таких случаях символический этап может быть проведен только один раз, и во всех последующих вычислениях может использоваться одна и та же статическая структура данных.

К несчастью, статические структуры данных могут применяться не во всех случаях. Самым заметным исключением является метод Гаусса с выбором главного элемента для матриц общего вида; он будет рассмотрен в § 3.4. Чтобы не допустить сильного роста ошибок, главные элементы выбираются в ходе исключения сообразно с их численными значениями; при этом используются такие стратегии выбора главного элемента, как полный выбор, частичный выбор или выбор по барьеру. Выбор главных элементов равносилен перестановкам строк и столбцов, которые в свою очередь влияют на расположение получающегося заполнения. Следствием этого является непредсказуемость портрета окончательной матрицы, и решение о том, где и как разместить каждый новый не-

нулевой элемент, должно приниматься, когда этот элемент уже вычислен и готов к записи в память. Такая процедура называется *динамическим распределением* памяти и порождает *динамическую* структуру данных.

Уже отмечалось, что если гауссово исключение проводить для ленточной матрицы общего вида и при выборе главного элемента пользоваться только перестановками строк, то верхняя полулента удваивается, а нижняя не меняется. Строго говоря, эта схема статическая, потому что может быть сформирована заранее. Она имеет и недостаток, присущий каждой статической схеме: в процессе исключения нельзя сэкономить память и приходится хранить и обрабатывать (или по крайней мере подвергать проверкам) большое число нулей. Подлинно динамические структуры требуют схем, не зависящих от статического по существу характера машинной памяти. Это сильное требование обычно удовлетворяется посредством эффективной системы связок и указателей. Схема Кнута и КРМ-схема идеально приспособлены для данной цели (§ 1.7). Каждый ненулевой элемент матрицы имеет внутреннее представление в виде записи фиксированной длины, которая хранится в произвольном месте машинной памяти, а затем связывается с соответствующими списками, строчным и столбцовым. Элементы могут быть включены или исключены в любой позиции матрицы без перемещения прочих данных, и, когда алгоритм этого потребует, можно просматривать как строки, так и столбцы.

Для динамического хранения разреженных матриц в [Houbak, 1981] было предложено использовать *записи переменной длины.* Запись понимается как двумерный массив специального типа, в который можно динамически добавлять строки и столбцами которого могут быть простые переменные различных типов. Неиспользуемые строки распознаются автоматически и при необходимости схема хранения сжимается. Не нужно хранить указатели и связки; отпадает нужда в косвенной адресации. Связные списки формируются, если присвоить ссылочной переменной, хранимой в строке, значение адреса следующей строки. Стоимость этого в терминах накладной памяти равносильна использованию целой переменной на строку. Эта схема поддерживается современными трансляторами с таких языков, как Паскаль и Алгол ω, но не со стандартного Фортрана; ее применение требует от транслятора способности обрабатывать связки и указатели. Программа, опирающаяся на эту структуру данных, выдерживает сравнение (по крайней мере для матриц малых или умеренных порядков) с двумя другими программами, использующими упорядоченные списочные схемы и написанными соответственно на Алголе *w* и Фортране [Barker et al., 1976; Thomsen, Zlatev, 1978].

1.13. СЛИЯНИЕ РАЗРЕЖЕННЫХ ЦЕЛЫХ СПИСКОВ

Теперь, когда основные схемы хранения, используемые в технологии разреженных матриц, изучены, мы начинаем анализ некоторых основных процедур, применяемых для обработки списков с целыми или вещественными элементами. Эти процедуры очень часто встречаются в составе матричных алгоритмов, особенно если матрицы представлены в разреженном формате. Мы будем пользоваться символами Фортрана прямо в тексте, чтобы избежать неоправданного введения алгебраических символов. Применимы обычные соглашения относительно типов INTEGER и REAL. Первая процедура, которую мы рассмотрим, предназначена для слияния разреженных целых списков.

Слияние есть операция ИЛИ с включением. Путем слияния двух или большего числа списков получается новый разреженный список. Целое число принадлежит результирующему списку тогда и только тогда, когда оно принадлежит хотя бы одному из заданных списков; при этом повторения не допускаются. Операция слияния целых списков очень важна в технологии разреженных матриц, поскольку постоянно используется для формирования списка столбцовых индексов каждой строки новой матрицы, полученной в результате выполнения алгебраических операций над исходной матрицей (или матрицами); особенно это относится к случаю, когда применяются разреженные форматы. Примерами могут быть сложение, умножение и треугольное разложение разреженных матриц. Следующий пример иллюстрирует понятие слияния. Пусть даны три списка:

список A: 2, 7, 3, 5
список B: 3, 11, 5
список C: 7, 2

Результирующий список (скажем, M) имеет вид:

список M: 2, 7, 3, 5, 11

Результирующий список M получается вписыванием каждого целого числа из каждого заданного списка при условии, что это число не было вписано ранее. Чтобы эффективно определять, не вписано ли уже данное число, мы используем расширенный массив, или *массив переключателей;* назовем его IX. Сразу после того, как в рассматриваемый список M было добавлено число I, в позицию I массива IX записывается условленное число—*переключатель.* Обратно, прежде чем добавить число K к списку M, мы проверяем, будет ли значение IX (K) равно переключателю, и только если не равно, K добавляется. Первый из заданных списков можно вписать в M полностью, *включив* соответствующие переключатели, но без предварительной проверки их включенности.

Напомним, что по предположению внутри каждого списка нет повторений. Проверка переключателей должна начаться при обработке второго заданного списка. Обратимся снова к нашему примеру. Если массив переключателей имел нулевое начальное состояние, а в качестве переключателя используется 1, то после вписывания первого списка мы получим следующее:

список M: 2, 7, 3, 5
позиция в IX: 1 2 3 4 5 6 7 8 9 10 11 12...
значение в IX: 0 1 1 0 1 0 1 0 0 0 0 0 ...

Во втором списке есть число 3. Так как IX (3) = 1, значению переключателя, то мы узнаем, что 3 уже включено в M и не должно включаться еще. раз. Заметим, что не будь эта информация доступна непосредственно, нам пришлось бы просматривать все элементы M — операция весьма неэффективная. Далее находим число 11. Поскольку IX (11) $\neq$ 1, то 11 добавляется в массив M, а IX (11) полагается равным 1. Ни одно из оставшихся чисел 5, 7, 2 не будет добавлено к M. Результирующий список, как правило, неупорядочен, даже если заданные списки были упорядоченными.

Когда списки не пересекаются, слияние превращается в простую конкатенацию. Для двух или более непересекающихся связных списков конкатенацию можно осуществить, переопределяя соответствующие связки и устраняя ставшие излишними указатели входа. В более общем контексте Тарьян [Tarjan, 1975] рассмотрел эффективность программ, обрабатывающих семейство непересекающихся множеств с целью вычисления их суммы или определения, какому множеству принадлежит данный элемент.

1.14. МЕТОД ПЕРЕМЕННОГО ПЕРЕКЛЮЧАТЕЛЯ

Всякий раз, как начинается операция слияния, введенный только что массив переключателей IX не должен содержать значение переключателя ни в одной из своих позиций. Этого можно добиться, полагая IX (I) = 0 (I = 1, ..., N) до перехода к слиянию и используя впоследствии положительное число в качестве переключателя. Однако оператор IX (I) =0 выполняется N раз, и если нужно произвести только одно слияние, то выгода от применения разреженных методов теряется.

Однако в технологии разреженных матриц слияния используются для построения списков столбцовых индексов, например N строк матрицы порядка N. Для этого требуется N различных слияний, и все они должны быть выполнены с помощью одного и того же массива переключателей IX длины N. В этом случае мы только однажды засылаем 0 в N позиций IX, а затем производим N слия-

ний, беря каждый раз *новое* значение переключателя [Gustavson, 1976b]. Можно, например, при первом слиянии использовать в качестве переключателя 1, при втором — 2, и т. д. Таким образом, когда начинается первое слияние, все позиции IX содержат 0. В начале второго слияния все позиции IX содержат либо 0, либо 1, что не препятствует использованию 2 как переключателя, и т. д. Теперь N исполнений оператора IX (*I*) = 0 приходятся на *N* операций слияния; в среднем только одно присваивание IX (*I*) = 0 на каждое слияние. Это приемлемо, если число ненулевых элементов *N* x *N* матрицы имеет хотя бы порядок *N*, что обычно выполняется на практике.

Альтернатива методу переменного переключателя состоит в том, чтобы каждый раз возвращать IX к нулевому состоянию. Когда слияние закончено, целые числа, хранящиеся в результирующем списке, указывают позиции в IX, куда были записаны ненулевые числа. Если число *M* элементов результирующего списка значительно меньше *N*, то для переопределения IX требуется только *M* операций. Этот метод может быть полезен в некоторых случаях, например если ради экономии памяти в качестве массива переключателей взят логический массив. Технику переменного переключателя здесь применить нельзя, потому что элементы логического массива могут принимать только два значения.

Метод переменного переключателя называют еще методом *фазового счетчика* [Gustavson, 1976 c].

1.15. СЛОЖЕНИЕ РАЗРЕЖЕННЫХ ВЕКТОРОВ С ИСПОЛЬЗОВАНИЕМ РАСШИРЕННОГО ВЕЩЕСТВЕННОГО НАКОПИТЕЛЯ

Вектор размерности *N* называют разреженным, если только несколько из его *N* элементов отличны от нуля. Разреженный вектор — это разреженная матрица, состоящая только из одной строки (вектор-строка) или одного столбца (вектор-столбец). В обоих случаях в памяти машины численные значения ненулевых элементов хранятся посредством вещественного массива, скажем AN, а соответствующие им индексы — в целом массиве, скажем JA. Оба массива имеют одинаковую длину, гораздо меньшую, чем *N*; поэтому данный тип хранения называют *компактным* или *упакованным.* Необходимо еще хранить информацию о числе ненулевых элементов вектора. Это число можно хранить непосредственно как значение целой переменной и использовать как указатель конца для массивов JA и AN. Иначе можно записать в JA сразу вслед за последней занятой позицией 0 в качестве признака конца целого списка.

Численные значения ненулевых элементов разреженного вектора можно хранить и в вещественном массиве длины N (скажем, X), как если бы вектор был полным — так называемая *расширенная* форма хранения. Однако индексы по-прежнему хранятся в массиве JA. Этот способ применяется для временного хранения, обычно на время работы подпрограммы, выполняющей над вектором какие-либо алгебраические операции. Наличие массива JA позволяет алгоритму оперировать только с ненулевыми элементами, вследствие чего количество операций будет гораздо меньше, чем N. Заметим, что если бы вектор рассматривался как полный, то массив JA можно было бы опустить. Но в этом случае алгоритму пришлось бы обрабатывать каждый из N элементов X, хотя бы чтобы выяснить, равно или нет нулю значение элемента, а тогда число операций было бы пропорционально N.

Операцию сложения двух или большего числа разреженных векторов лучше всего проиллюстрировать с помощью примера. Рассмотрим два разреженных вектора a и b размерности N, которые хранятся в компактном неупорядоченном формате: a — посредством массивов JA, AN; b — посредством массивов JB, BN. Мы хотим вычислить вектор $c = a + b$ и разместить его в массивах JC, CN. Пусть

```
JA =  10   3    7     4
AN =  0.2  0.3  0.4  —0.7
JB =   5   4    10
BN =  0.6  0.7  0.5
```

Если мы попытаемся вычислить $c_1 = a_1 + b_1$, то скоро поймем, что для этого необходимо просмотреть массивы JA и JB, а затем выполнить соответствующие операции над вещественными числами, хранимыми в AN, и BN. В самом деле, чтобы определить a_1, мы просматриваем JA и обнаруживаем, что ни одно из находящихся в нем чисел не равно 1; следовательно, $a_1 = 0$. Таким же образом путем просмотра JB убеждаемся, что $b_1 = 0$.; значит, $c_1 = a_1 + b_1 = 0$. Тот же вывод получаем для c_2, в то время как для c_3 находим: $a_3 = 0.3$ и $b_3 = 0$, поэтому $c_3 = 0.3$, и т. д. Для *каждого* элемента c приходится просматривать *все* элементы JA и все элементы JB. Эта процедура очень неэффективна, поскольку требует выполнения элементарных операций в количестве, приблизительно пропорциональном *квадрату* числа ненулевых элементов в a и b.[1])

Эффективная процедура получается, если разбить алгоритм на символический и численный этапы. В этом случае, независимо от числа ненулевых элементов, потребуется фиксированное и

[1]) В той форме, как описана процедура, ее трудоемкость пропорциональна произведению N (| *а* | + | *в* |). — *Прим. перев.*

небольшое число просмотров JA и JB. Это означает, что общее число элементарных операций будет *линейной* функцией от числа ненулевых элементов.

Символический этап состоит в определении позиций ненулевых элементов *c* путем слияния списков JA и JB; при этом используется расширенный массив переключателей IX длины *N,* как это объяснено в § 1.13. Массиву переключателей нужно придать нулевое начальное состояние. Результатом символического этапа будет список

JC = 10 3 7 4 5

Теперь, когда мы знаем позиции ненулевых элементов (или структуру) *c,* мы определим их численные значения в численном этапе алгоритма. Опять-таки это делается не прямолинейно. Мы не можем вычислять c_{10}, первую ненулевую компоненту вектора *c,* указываемую массивом JC, пытаясь найти в JA и JB число 10. Вместо этого мы воспользуемся расширенной формой хранения векторов в массиве (скажем, X) размерности *N,* называемом *расширенным вещественным накопителем.* Прежде всего мы зашлем 0. во все нужные позиции X, т. е. те, что указаны в JC. Получим:

позиция: 1 2 3 4 5 6 7 8 9 10 11 12 ...
значение X: *x x* 0. 0. 0. *x* 0. *x x* 0. *x x* ...

Символом *x* отмечены несущественные для нас позиции. Теперь мы *загружаем* AN в X. Имеем:

позиция: 1 2 3 4 5 6 7 8 9 10 11 ...
значение X: *x x* 0.3 —0.7 0. *x* 0. 4*x x* 0.2 *x* ...

Далее *прибавляем* BN к X:

позиция: 1 2 3 4 5 6 7 8 9 10 11 ...
значение X: *x x* 0.3 0. 0.6 *x* 0.4 *x x* 0.7 *x* …

Наконец, мы *выбираем* из X нужные числа, чтобы сформировать CN. С помощью JC определяем, где хранятся в X нужные числа; например, так как JC (1) = 10, находим X (10) = 0.7 и полагаем CN (1) = X (10). Окончательным результатом будет:

JC = 10 3 7 4 5
CN = 0.7 0.3 0.4 0 0.6

Для его получения потребовалось число операций, пропорциональное числу ненулевых элементов в *a* и *b*, если *не считать N* операций на засылку нулей в массив переключателей IX перед слиянием JA и JB. С этого момента мы сосредоточим внимание на интересующем нас случае алгебры разреженных матриц. Боль-

шинство матричных операций требуют выполнять сложение разреженных векторов не один, а O *(N)* раз, где O *(N)* означает «число порядка *N»;* дело в том, что каждая строка матрицы — это разреженный вектор. Если использовать метод переменного переключателя из § 1.14, то нулевое состояние присваивается массиву IX только один раз; таким образом, на O *(M)* векторных сложений приходится *N* операций засылки, что вполне приемлемо.

Численный пример был умышленно подобран так, чтобы получить $c_4 = 0$. Взаимное уничтожение вещественных чисел происходит редко, и такая возможность обычно не принимается в расчет. Это означает, что некоторые из «ненулевых» элементов матрицы могут быть в действительности нулями, и алгоритм должен быть готов примириться с этим обстоятельством. Однако в некоторых случаях взаимные уничтожения будут частыми, например, для матриц, целиком состоящих из 1 и —1. В подобных случаях, возможно, стоит конструировать алгоритм так, чтобы он препятствовал записи нулей в результат.

Любое число разреженных векторов можно сложить за один символический и один численный этап. Таким же образом можно построить разность или линейную комбинацию разреженных векторов. Результаты будут неупорядоченными, даже если заданные векторы упорядочены. Однако численный этап сохраняет упорядоченность, поскольку ненулевые элементы выбираются из X и записываются в CN в том самом порядке, в каком соответствующие Индексы хранятся в JC. Следовательно, если результаты нужно упорядочить, то достаточно до выполнения численного алгоритма упорядочить массив JC.

1.16. СЛОЖЕНИЕ РАЗРЕЖЕННЫХ ВЕКТОРОВ С ИСПОЛЬЗОВАНИЕМ РАСШИРЕННОГО ЦЕЛОГО МАССИВА УКАЗАТЕЛЕЙ

Имеется другой алгоритм для численного этапа операции сложения разреженных векторов, в котором в качестве накопителя используется сам результирующий вектор. Мы снова обратимся к примеру из § 1.15 и предположим, что массив JC уже определен на символическом этапе. Теперь мы хотим найти численные значения для результирующего вектора CN. Так как мы собираемся использовать CN в качестве накопителя, то переводим его в нулевое начальное состояние. Имеем:

```
JC =  10  3  7  4  5
CN =   0. 0. 0. 0. 0.
```

Теперь мы заполним *расширенный массив указателей* IP, начальное состояние которого — нулевое. Мы просматриваем JC и

помещаем в IP указатели на позиции CN, где будут храниться ненулевые элементы. Например, в первой позиции JC находим 10; это говорит о том, что компонента c_{10} будет храниться в 1-й позиции CN. Поэтому полагаем IP (10) = 1. Аналогично, полагаем IP (3) == 2, и т. д. После того, как массив JC будет исчерпан, получим

позиция: 1 2 3 4 5 6 7 8 9 10 11 ...
значение IP: 0 0 2 4 5 0 3 0 0 1 0 ...

Далее мы сканируем массивы JA и AN, определяя с помощью IP позиции CN, где должны накапливаться значения ненулевых элементов. Например, в 1-й позиции массивов JA и AN находим соответственно 10 и 0.2; в позиции 10 массива IP находим 1; следовательно, нужно прибавить 0.2 к содержимому 1-й позиции CN. Эта процедура повторяется со всеми элементами в JA, AN и JB, BN. Наконец, с помощью JC возвращаем массив IP к нулевому состоянию, подготовив его к новому использованию. Посредством этого алгоритма можно эффективно складывать любое число разреженных векторов, сводя к минимуму перемещение данных. Численный этап сохраняет порядок, поскольку не изменяет JC.

Заметим, что верный результат был бы получен и в том случае, если бы после перевода CN в нулевое состояние элементы из AN были загружены в CN непосредственно без какого-либо обращения к IP. Так можно делать потому, что при слиянии двух целых списков JA и JB результирующий список JC содержит первый из данных списков (в нашем случае JA) в неизменном виде. Эта процедура более эффективна, чем описанная выше, и может использоваться для первого из двух или большего числа слагаемых. Нужно только позаботиться, чтобы в символическом и численном этапах векторы задавались в одинаковом порядке.

1.17. СКАЛЯРНОЕ УМНОЖЕНИЕ ДВУХ РАЗРЕЖЕННЫХ ВЕКТОРОВ С ИСПОЛЬЗОВАНИЕМ МАССИВА УКАЗАТЕЛЕЙ

В этом параграфе мы рассмотрим следующую задачу: два разреженных вектора a и b размерности N хранятся в массивах JA, AN и JB, BN; оба представления — компактные и неупорядоченные. Мы хотим вычислить скалярное произведение этих векторов:

$$h = \sum_{i=1}^{N} a_i b_i.$$

Снова обратимся к примеру из § 1.15. Прямолинейная попытка вычислить скалярное произведение привела бы к многочисленным

просмотрам массива JB. Действительно, начнем с первого ненулевого элемента, хранящегося в JA, AN, т. е. с $a_{10} = 0.2$. Это число нужно умножить на b_{10}. Чтобы найти b_{10}, мы должны просмотреть весь массив JB и выяснить, что b_{10} хранится в BN (3) и равен 0.5. Теперь вычисляется и запоминается произведение $a_{10}b_{10}$. Следующий ненулевой элемент — a_3 — 0.3, и мы снова просматриваем JB только для того, чтобы обнаружить: $b_3 = 0$. Для каждого элемента в a нам приходится просматривать JB. Эта процедура очень неэффективна, поскольку число выполняемых проверок пропорционально произведению числа ненулевых элементов a и числа ненулевых элементов b.

Нижеследующий алгоритм требует только одного просмотра a и b плюс фиксированное небольшое число элементарных операций для каждого ненулевого элемента векторов. Так как конечным результатом будет единственное вещественное число h, то алгоритм не имеет символического этапа. Для хранения указателей позиций ненулевых элементов в AN используется расширенный целый массив IP. Этот массив с нулевым начальным состоянием заполняется путем одного просмотра JA. Для примера из § 1.15 результат таков:

позиция: 1 2 3 4 5 6 7 8 9 10 11 ...
значение IP: 0 0 2 4 0 0 3 0 0 1 0 ...

Этот массив сообщает, что a_3 хранится во 2-й позиции AN, a_4 — в 4-й позиции, и т. д. Он сообщает также, что $a_1 = 0.$, $a_2 = 0.$, и т. д.

Далее мы просматриваем ненулевые элементы b, используя IP, чтобы найти соответствующие элементы a, при необходимости вычисляем произведения и накапливаем результаты в ячейке h. Например, в первой позиции JB, BN находим $b_5 = 0.6$. Таким образом, поскольку IP (5) = 0, то $a_5 = 0.$, и произведение a_5b_5 вычислять не нужно. Затем мы находим b_4 — 0.7; поскольку IP (4) = 4, то значение a_4 хранится в 4-й позиции AN, и мы вычисляем произведение a_4b_4. Эта процедура продолжается, пока не будет исчерпан массив JB.

Особенно интересно применение этого алгоритма в ситуации, когда вектор a нужно скалярно умножить на несколько векторов. В этом случае массив IP заполняется только один раз и затем используется для вычисления всех требуемых скалярных произведений. Указанная ситуация возникает при необходимости перемножить разреженную матрицу и разреженный вектор; более подробно она будет рассмотрена в гл. 7.

Глава 2

Линейные алгебраические уравнения

2.1. ВВЕДЕНИЕ

В этой и последующих главах мы будем обсуждать алгоритмы, предназначенные для отыскания решения χ системы

$$Ax=b, \tag{2.1}$$

где A — невырожденная квадратная вещественная матрица размеров $n \times n$, а b — заданный вектор. Эти алгоритмы можно разделить на два класса: прямые методы и итерационные методы. Прямые методы основаны на гауссовом исключении: в результате выполнения последовательности шагов уравнения или неизвестные системы модифицируются до тех пор, пока не будет найдено решение. В итерационных методах обычно выбирается начальное приближение для x, которое затем улучшается, пока не будет достигнута достаточная точность. В каждом частном случае оба этих подхода имеют как преимущества, так и недостатки, и трудно установить общие правила для определения того, какой из них окажется более подходящим.

Здесь мы ограничим рассмотрение случаем прямых методов для решения уравнения (2.1) с вещественной невырожденной квад-

ратной матрицей порядка n. Комплексная система линейных уравнений может быть преобразована в вещественную систему в два раза большего порядка, как показано в § 2.2. Решение уравнения (2.1) может быть записано в виде $x = A^{-1}b$, но явное использование этой формы записи производится только в случае небольших значений n (скажем, $n \sim 100$ и меньше). Матрица A^{-1} часто является плотной, даже если A — разреженная матрица, и ее явное вычисление ведет к потере преимуществ, свойственных разреженному случаю. Для больших систем указанные преимущества извлекаются из того факта, что нет необходимости получать A^{-1} в явном виде; достаточно выразить A^{-1} через произведение матриц, которые простым способом умножаются на вектор. Конечно, такая процедура оказывается более удобной, так как матричные сомножители являются разреженными и при хранении все они могут быть помещены в значительно меньший объем памяти, чем одна матрица A^{-1}; кроме того, произведение матричных сомножителей на вектор b требует меньше операций, чем вычисление произведения $A^{-1}b$. Обычным способом решения уравнения (2.1) прямым методом является вычисление *треугольного разложения* матрицы A:

$$A = LU, \tag{2.2}$$

где U — верхняя треугольная матрица (ее нижняя треугольная часть содержит только нулевые элементы) с диагональными элементами, равными единице, а L — нижняя треугольная (ее верхняя треугольная часть содержит только нулевые элементы). В этом случае $A^{-1} = U^{-1}L^{-1}$ и $x = U^{-1}\omega$, где $\omega = L^{-1}b$. Таким образом, решение уравнения (2.1) определяется в результате решения линейной системы

$$L\omega = b \tag{2.3}$$

относительно ω, а затем — линейной системы

$$Ux = \omega \tag{2.4}$$

относительно x. Если матрица A разрежена, то разреженными будут также и матрицы L и U, хотя степень их разреженности обычно меньше по сравнению с матрицей A. Системы типа (2.3) и (2.4), решение которых не представляет труда, изучаются в § 2.13.

Разложение (2.2) единственно, если оно существует и матрица A невырождена [Wilkinson, 1965]. Часто из матрицы L выносится в качестве множителя диагональная матрица D, так что $L = L'D$, где L' является нижней треугольной матрицей, все диагональные элементы которой равны единице. Уравнение (2.2) в этом случае принимает следующий вид:

$$A = L'DU. \tag{2.5}$$

Если матрица симметричная, то $L' = U^T$. Возможны и другие разложения матрицы A. Важно изучать различные возможности, так как они обеспечивают большую гибкость при разработке алгоритмов и схем хранения. Именно эту задачу мы и ставим перед собой в данной главе.

Выполняя арифметические операции, вычислительные машины вносят в них погрешности округления. Большие вычислительные задачи, такие, как решение системы (2.1) высокого порядка, требуют тщательного анализа погрешностей округления и использования специальной техники для того, чтобы гарантировать получение осмысленного конечного результата с заданной точностью. В алгоритмах гауссова исключения чрезмерное накопление ошибок обычно предотвращается путем масштабирования исходной матрицы, выбора подходящих главных элементов в процессе исключения и контроля за ростом элементов последовательных редуцированных матриц. Эти вопросы изучаются в гл. 3.

Структура расположения ненулевых элементов (портрет) матрицы A зависит от происхождения системы (2.1); так, матрицы возникающие в конечноэлементных задачах, и матрицы, связанные с моделированием электрических сетей, имеют совершенно различные портреты (см., например, [Duff, 1981a, p. 17]). Вследствие этого существуют два класса алгоритмов: те, которые используют частные свойства матрицы A и поэтому предназначены для специфических приложений, и те, которые работают одинаково хорошо для произвольной разреженной матрицы. Обычно чем больше налагается ограничений, тем более эффективными получаются алгоритмы. Разработка алгоритмов для специальных приложений привлекает значительный интерес, и исследования в этой области проводятся в настоящее время весьма активно.

Структура матрицы системы линейных уравнений может быть изменена как путем введения новой нумерации неизвестных, так и за счет переупорядочения уравнений. В то время как эти преобразования весьма мало отражаются на эффективности большинства итерационных методов [1]), для прямых методов они оказываются очень важными. Причина этого заключается в том, что в процессе исключения образуются новые ненулевые коэффициенты и матрица системы становится менее разреженной: она претерпевает *заполнение*. Хорошо известно, что степень заполнения весьма сильно зависит от принятого способа упорядочения неизвестных и уравнений. Поскольку требуемые для решения системы объемы памяти и вычислений, а также точность окончательного результата зависят в свою очередь от степени заполнения, целе-

[1]) Имеются в виду симметричные перестановки строк и столбцов и простейшие методы типа Якоби, Гаусса—Зейделя и последовательной верхней релаксации. — *Прим. перев.*

сообразно рассматривать такие упорядочения, которые уменьшают заполнение настолько, насколько это возможно. Исследования в этой области позволили внести в прямые методы решения систем уравнений ряд важных усовершенствований, которые изучаются в гл. 4 и 5.

Другим прямым методом решения линейных уравнений является *ортогональное разложение.* Мы обсуждаем технику ортогонализации в гл. 6 в связи с решением задач на собственные значения. Ортогонализация обладает лучшей численной устойчивостью по сравнению с гауссовым исключением, но приводит к существенно большему заполнению. Методы, основанные на ортогонализации, не рекомендуется применять при решении разреженных систем общего вида [Duff, 1974b; Duff, Reid, 1975]. Однако они могут быть с успехом применены в некоторых случаях, например, при решении линейных задач методом наименьших квадратов и в линейном программировании [Gill, Murray, 1976], а также в AQ-алгоритме [Borland, 1981] для случая, когда матрица коэффициентов может быть приведена к почти нижней хессенберговой форме путем перестановок.

Методы исключения и факторизации могут быть применены не только к матрицам, элементами которых являются числа, но и к матрицам, составленным из подматриц; методы, получающиеся в этом случае, называют *блочными методами.* Они алгебраически эквивалентны обычному исключению, с той лишь разницей, что такие операции, как сложение, умножение и деление единицы на число заменяются на матричное сложение, умножение и обращение матриц. Однако блочные методы отличаются с точки зрения разреженности и численной устойчивости; эти вопросы изучаются в гл. 3, 4 и 5.

2.2. НЕКОТОРЫЕ ОПРЕДЕЛЕНИЯ И СВОЙСТВА

Матрицу A называют *симметричной,* если $A^{\mathrm{T}} = A$, где индекс т означает операцию транспонирования; таким образом, для симметричной матрицы A имеем $a_{i\,j} = a_{j\,i}$ для всех i и j. В противном случае матрица A называется *несимметричной.* Говорят, что симметричная матрица A *положительно определена,* если

$$y^T A y > 0 \tag{2.6}$$

для любого вектора y, имеющего хотя бы одну ненулевую компоненту. Если найдутся два вектора y и z, для которых выполнены неравенства

$$y^T A y > 0, \quad z^T A z < 0, \tag{2.7}$$

то матрицу A называют *незнакоопределенной.* С вычислительной точки зрения симметрия и положительная определенность явля-

ются важными свойствами. Полезно изучить эти свойства несколько более детально. Сводка результатов дается ниже.

(1) Если A и B — симметричные положительно определенные матрицы, то матрица $A + B$ также симметрична и положительно определена. Это свойство оказывается полезным в той ситуации, когда матрица большого размера представляется в виде суммы матриц меньшего размера[1]), положительная определенность которых уже установлена. Заметим, что для матриц малого размера проверить положительную определенность значительно легче, чем для больших матриц. Важный пример дается в гл. 8.

(2) Симметричная положительно определенная матрица имеет η собственных значений, являющихся вещественными положительными числами.

(3) Определитель симметричной положительно определенной матрицы положителен.

(4) *Подматрицей*[2]) матрицы A называется матрица, полученная вычеркиванием некоторых строк и столбцов. Главная подматрица получается, когда номера вычеркнутых строк совпадают с номерами вычеркнутых столбцов. Если матрица A — — симметричная положительно определенная, то любая ее главная подматрица также симметрична и положительно определена и поэтому имеет положительный определитель. В частности, все диагональные элементы матрицы A положительны:

$$A_{ii} > 0 \quad \text{для всех } i.$$

(5) Для положительно определенной матрицы A рассмотрим определитель произвольной главной подматрицы второго порядка, который, согласно свойству (4), должен быть положительным:

$$\operatorname{Det} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} = a_{ii}a_{jj} - a_{ij}^2 > 0.$$

Отсюда получаем оценку

$$| a_{ij} | \leq (a_{ii}a_{jj})^{1/2} \leq \max (a_{ii}, a_{jj}).$$

В частности, если a — значение наибольшего диагонального элемента матрицы A, то любой элемент матрицы A по модулю не превосходит этого значения

$$|a_{ij}| \leq a \quad \text{для всех } i \text{ и } j.$$

[1]) С точки зрения формального определения суммы матриц складывать можно только матрицы одинакового порядка. Говоря о сложении матриц с разными размерами, автор имеет в виду ситуацию, характерную для метода конечных элементов: матрица глобальной задачи «собирается» из малых матриц, описывающих отдельные элементы. — *Прим. перев.*

[2]) В оригинале — «minor», однако минором в математической литературе на русском языке называют не подматрицу, а определитель подматрицы. — *Прим. перев.*

(6) Необходимым и достаточным условием того, чтобы симметричная матрица A была положительно определенной, является положительность всех ее ведущих главных подматриц [Wilkinson, 1965].

(7) Если матрица A — симметричная положительно определенная, то существует и единственно *разложение Холецкого*

$$A = U^T U,$$

где U — верхняя треугольная матрица с положительными диагональными элементами [Stewart, 1973]. Кроме того, матрицу U можно переписать в виде $U = D'U'$, где D' — диагональная матрица с положительными диагональными элементами, а U' — верхняя треугольная матрица, все диагональные элементы которой равны единице. В этом случае

$$A = U'^T D U',$$

где $D = D'^2$, также является единственной треугольной факторизацией указанного вида [Martin, Wilkinson, 1965].

(8) Невырожденная квадратная матрица R, такая, что $R^T = R^{-1}$, называется *ортогональной*. Если матрица A симметрична и положительно определена, то матрица $B = R^T A R$ также является симметричной и положительно определенной [1]).

(9) Если C — произвольная матрица, квадратная или прямоугольная, то матрица $A = C^T C$ является симметричной неотрицательно определенной [2]). Очевидно, что матрица A симметрична. Для того чтобы показать ее неотрицательную определенность, рассмотрим произвольный ненулевой вектор x соответствующей размерности. Тогда $x^T A x = x^T C^T C x = (Cx)^T Cx \geq 0$. Кроме того, если столбцы матрицы C линейно независимы, то $Cx \neq 0$ и матрица A положительно определена.

(10) Если матрица A незнакоопределена, то треугольное разложение может не существовать, а вырожденная матрица A может иметь бесконечное число треугольных разложений [Wilkinson, 1965, p. 224].

(11) Задача определения того, будет ли данная матрица A положительно определена, редко возникает на практике. Мы обычно знаем, что матрица A является симметричной положительно определенной по построению, как правило, в силу свойств (1), (8) и (9). Тогда мы можем использовать преимущества этой

[1]) Для этого матрица R, вообще говоря, не обязана быть ортогональной; достаточно потребовать ее невырожденности. — *Прим.. перев.*

[2]) Матрица A называется неотрицательно определенной, если $x^T A x \geq 0$ для всех x. — *Прим. перев.*

ситуации, вытекающие из свойства (2) или свойства (7). Теоретическое значение остальных свойств велико, но практическое их приложение ограниченно.

Ранг r произвольной матрицы A равен наибольшему порядку ее подматрицы, имеющей ненулевой определитель. Матрица порядка n называется *вырожденной,* если $r < n$. Величина $n—r$ называется *дефектом* матрицы. Симметричная положительно определенная матрица имеет ранг $r = n$. Любая матрица ранга единица представима в виде xy^T, где x и y — некоторые ненулевые векторы-столбцы; в этом случае каждая строка матрицы пропорциональна вектору y^T, а каждый столбец пропорционален вектору x.

Говорят, что матрица A имеет *диагональное преобладание,* если

$$|a_{ii}| \geqslant \sum_{j \neq i} |a_{ij}| \tag{2.8}$$

для всех i, причем хотя бы для одного i выполнено строгое неравенство [1]). Матрица A имеет строгое диагональное преобладание, если

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \tag{2.9}$$

для всех i. Можно показать, что если A — матрица со строгим диагональным преобладанием и все ее диагональные элементы положительны, т. е.

$$a_{ii} > \sum_{j \neq i} |a_{ij}| \tag{2.10}$$

для всех i, то все собственные значения матрицы A положительны и она является положительно определенной [2]). Простое доказательство этого факта, основанное на теореме, принадлежащей Гершгорину [Gerschgorin, 1931] обсуждается в § 6.3 (см. также [Wilkinson, 1965, p. 71]).

Мы будем рассматривать только системы вида (2.1) с вещественными матрицей A и правой частью b; вследствие этого решение x также будет вещественным. Комплексная система вида

$$(A + iB)(x + iy) = b + ic, \tag{2.11}$$

[1]) Если не потребовать неразложимости матрицы A, то последнее условие ничего нового, вообще говоря, не дает; например, не гарантируется невырожденность A.—*Прим. перев.*

[2]) Необходимо также потребовать, чтобы матрица A была симметричной. — ***Прим. перев.***

где A, B, x, y, b и c — вещественны, эквивалентна следующей вещественной системе уравнений удвоенного порядка:

$$\begin{bmatrix} A & -B \\ B & A \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}. \qquad (2.12)$$

Комплексная матрица M называется *эрмитовой,* если $M = M^*$, где знак «*» означает комплексное сопряжение. Матрица, комплексно сопряженная к $A + iB$, есть $A^T - iB^T$; поэтому матрица $A + iB$ является эрмитовой в том случае, если $A^T = A$ и $B^T = -B$. В случае когда матрица $A + iB$ эрмитова, система (2.12) симметричная. Решая систему (2.12) вместо (2.11), мы остаемся в поле вещественных чисел, что представляется выгодным с вычислительной точки зрения в том случае, если комплексная арифметика на данной вычислительной машине реализована неэффективно.

2.3. ЭЛЕМЕНТАРНЫЕ МАТРИЦЫ И ТРЕУГОЛЬНЫЕ МАТРИЦЫ

Некоторые результаты потребуются в дальнейшем изложении в этой и других главах, и поэтому для удобства они собраны и изложены ниже.

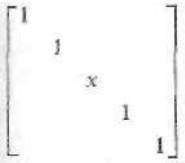
Элементарной матрицей называется матрица E вида $E = I + B$, где I — единичная матрица, а B — матрица ранга единица. Так как любая матрица ранга единица представима в виде $B = xy^T$, где x и y — два ненулевых вектора-столбца (см. § 2.2), то общий вид элементарной матрицы дается формулой

$$E = I + xy^T. \qquad (2.13)$$

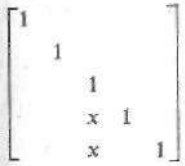
а) Квадратная матрица L называется *нижней треугольной,* если ее нулевые элементы расположены только на диагонали или ниже диагонали: $l_{ij} = 0$ при $i < j$ и по крайней мере один элемент $l_{ij} \neq 0$ при $i > j$. Нижняя треугольная матрица называется *нижней треугольной с единичной диагональю,* если все ее диагональные элементы равны единице: $l_{ii} = 1$ для всех i.

б) Квадратная матрица U называется *верхней треугольной,* если ее нулевые элементы расположены только на диагонали или выше диагонали: $u_{ij} = 0$ при $i > j$ и по крайней мере один элемент $u_{ij} \neq 0$ при $i < j$. Верхняя треугольная матрица называется *верхней треугольной матрицей с единичной диагональю,* если все ее диагональные элементы равны единице: $u_{ii} = 1$ для всех i.

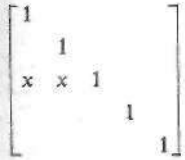
в) .Ниже приведены примеры и определения некоторых элементарных матриц специального вида, которые потребуются позднее. Символом «x» отмечены позиции элементов, которые могут быть отличны от нуля, а позиции, отвечающие нулевым элементам, оставлены незаполненными.



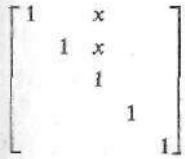
Диагональная элементарная матрица D_k.. Диагональная матрица с элементами $(D_k)_{ii} = 1$ для всех $i \neq k$ и $(D_k)_{kk} \neq 0$.



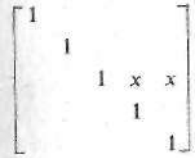
Нижняя столбцовая элементарная матрица L_k^C. Внедиагональные нулевые элементы расположены только в k-м столбце под диагональю.



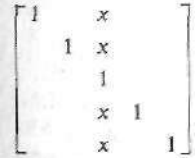
Левая строчная элементарная матрица L_k^R Внедиагональные нулевые элементы расположены только в *k-й* строке левее диагонали.



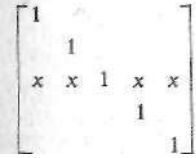
Верхняя столбцовая элементарная матрица U_k^C Внедиагональные ненулевые элементы расположены только в k-м столбце над диагональю.



Правая строчная элементарная матрица U_k^R Внедиагональные ненулевые элементы расположены только в *k-й* строке справа от диагонали.



Полная столбцовая элементарная матрица T_k^C Внедиагональные ненулевые элементы расположены только в k-м столбце.



Полная строчная элементарная матрица T_k^R Внедиагональные ненулевые. элементы расположены только в *k-й* строке.

Следует отличать названия этих матриц от таких терминов, как «столбцовая матрица» *(nX* 1- матрица), которая обычно называется «вектором-столбцом». Конечно, любая из перечисленных выше элементарных матриц может быть представлена в памяти компьютера только той своей строкой, которая действительно используется.

2-4. НЕКОТОРЫЕ СВОЙСТВА ЭЛЕМЕНТАРНЫХ МАТРИЦ

(а) В некоторых случаях элементарные матрицы перестановочны с диагональными элементарными матрицами, а именно,

$$L_i^C D_j = D_j L_i^C \text{ при } i > j, \quad U_i^C D_j = D_j U_i^C \text{ при } i < j,$$
$$L_i^R D_j = D_j L_i^R \text{ при } i < j, \quad U_i^R D_j = D_j U_i^R \text{ при } i > j.$$

(б) Нижние и верхние столбцовые матрицы перестановочны в следующих случаях:

$$L_i^C U_j^C = U_j^C L_i^C \text{ при } i \geqslant j, \quad L_i^R U_j^R = U_j^R L_i^R \text{ при } i \leqslant j.$$

Эти произведения получаются суперпозицией внедиагональных элементов [1]).

(в) Полные столбцовые или строчные матрицы можно представить в виде произведений соответственно столбцовых или строчных треугольных матриц:

$$L_k^C U_k^C = U_k^C L_k^C = T_k^C, \qquad L_k^R U_k^R = U_k^R L_k^R = T_k^R,$$

где произведения получаются путем суперпозиции внедиагональных элементов сомножителей.

(г) Матрица, обратная к любой из элементарных матриц, за исключением D_k, получается путем изменения алгебраических знаков внедиагональных элементов. Обращение матрицы D_k сводится к вычислению величины, обратной $(D_k)_{kk}$..

(д) Произведения матрицы следующего вида получаются просто путем суперпозиции внедиагональных элементов сомножителей:

$$L_i^C L_j^C \text{ при } i < j, \; L_i^R L_j^R \text{ при } i < j, \; U_i^C U_j^C \text{ при } i > j, \; U_i^R U_j^R \text{ при } i > j.$$

2.5. НЕКОТОРЫЕ СВОЙСТВА ТРЕУГОЛЬНЫХ МАТРИЦ

(а) Произведение двух нижних треугольных матриц также будет нижней треугольной матрицей. Поэтому произведение любого числа нижних треугольных матриц представляет собой нижнюю треугольную матрицу. Если все матричные сомножители

[x]) Говоря здесь и далее о «суперпозиции внедиагональных элементов», автор имеет в виду, что если $L = I + xy^T$ и $U = I + uv^T$, то в указанных случаях $LU = I + xy^T + uv^T$, причем сложения ненулевых элементов матриц xy^T и uv^T не происходит, так как эти элементы стоят в разных позициях. — *Прим. перев.*

являются треугольными с единичной диагональю, то произведение будет треугольной матрицей с единичной диагональю. Аналогичными свойствами обладают верхние треугольные матрицы.

(б) Согласно свойству 2.4 (д), любая нижняя треугольная матрица L с единичной диагональю может быть записана в виде произведения n — 1 элементарных матриц нижнего столбцового или левого строчного типа:

$$L = L_1^C L_2^C \ldots L_{n-1}^C, \quad L = L_2^R L_3^R \ldots L_n^R,$$

где ненулевые внедиагональные элементы матрицы L_i^C те же, что и в i-м столбце матрицы L, а ненулевые внедиагональные элементы матрицы L_i^R те же, что и в i-й строке L. Следующие примеры иллюстрируют этот факт в случае $n = 4$:

$$L = \begin{bmatrix} 1 & & & \\ a & 1 & & \\ b & c & 1 & \\ d & e & f & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ a & 1 & & \\ b & & 1 & \\ d & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & c & 1 & \\ & e & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & f & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & & & \\ a & 1 & & \\ b & c & 1 & \\ d & e & f & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ a & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ b & c & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ d & e & f & 1 \end{bmatrix}$$

Исходя из этих соотношений, мы можем рассматривать L как *таблицу множителей* [Tinney, Walker 1967], представляющую множество матриц L_i^C и множество матриц L_i^R, хранимых в компактной форме. Заметим, что эти множители неперестановочны.

(в) Нетрудно получить матрицу, обратную к нижней треугольной матрице с единичной диагональю. Согласно свойству 2.4 (г), матрицы $(L_i^C)^{-1}$ и $(L_i^R)^{-1}$ совпадают соответственно с матрицами, у которых изменены алгебраические знаки внедиагональных элементов. Вычисление матрицы L^{-1} производится непосредственно с использованием любого из двух разложений и включает только умножения и сложения. Вследствие этого свойства и свойства 2.5 (а) мы видим, что матрица L^{-1} также является нижней треугольной матрицей с единичной диагональю.

(г) Произведение L^{-1} на другую матрицу (или вектор) можно вычислить, используя непосредственно матрицу L, без явного вычисления элементов матрицы L^{-1}. Для этого данная матрица (или вектор) умножается на взятые в соответствующем порядке множители $(L_i^C)^{-1}$ или $(L_i^R)^{-1}$, на которые разложена матрица L^{-1}. Эти множители, согласно свойству 2.4 (г), получаются непосредственно из столбцов или строк матрицы L изменением знаков вне-

диагональных элементов. В качестве примера приведем два способа умножения матрицы, обратной к матрице L из п. 2.5 (б), на вектор-столбец:

$$\begin{bmatrix} 1 & & & \\ a & 1 & & \\ b & c & 1 & \\ d & e & f & 1 \end{bmatrix}^{-1} \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} =$$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -f & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -c & 1 & \\ & -e & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ -a & 1 & & \\ -b & & 1 & \\ -d & & & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & \\ a & 1 & & \\ b & c & 1 & \\ d & e & f & 1 \end{bmatrix}^{-1} \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix} =$$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ -d & -e & -f & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ -b & -c & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ -a & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \\ s \end{bmatrix}$$

Важным следствием этого свойства является то, что не требуется дополнительной памяти для хранения матрицы L^{-1} в памяти компьютера. Достаточно хранить матрицу L.

(д) Верхняя треугольная матрица O с единичной диагональю может быть записана в виде произведения n — 1 элементарных матриц верхнего столбцового или правого строчного типа:

$$U = U_n^C U_{n-1}^C \ldots U_2^C, \quad U = U_{n-1}^R U_{n-2}^R \ldots U_1^R,$$

где ненулевые внедиагональные элементы матрицы U_i^C те же, что и в i-м столбце матрицы U, а ненулевые внедиагональные элементы матрицы U_i^R те же, что и в i-й строке матрицы U. Аналогично 2.5 (б), U может рассматриваться как таблица множителей.

(е) Матрица, обратная к верхней треугольной матрице с единичной диагональю, может быть вычислена двумя способами:

$$U^{-1} = (U_2^C)^{-1} \ldots (U_{n-1}^C)^{-1} (U_n^C)^{-1},$$
$$U^{-1} = (U_1^R)^{-1} \ldots (U_{n-2}^R)^{-1} (U_{n-1}^R)^{-1}.$$

Матрица U^{-1} также является верхней треугольной с единичной диагональю и ее вычисление использует ту же таблицу множителей, которая представляет матрицу U, но с измененными зна-

ками внедиагональных элементов, аналогично тому как это было показано в 2.5 (в) для матрицы L.

(ж) Произведение U^{-1} на другую матрицу или вектор с использованием элементов матрицы U можно вычислить при помощи процедуры, аналогичной той, которая описана в 2.5 (г) для матрицы L. Имеет место то же важное следствие, что и в 2.5 (г): для представления U^{-1} не требуется дополнительной памяти.

2.6. МАТРИЦЫ ПЕРЕСТАНОВОК

Матрицей перестановки P называется квадратная матрица порядка n, такая, что каждая строка и каждый столбец содержат один элемент, равный единице, а остальные их элементы равны нулю.

Простейшей матрицей перестановки является единичная матрица I. Легко проверить, что произведение любой матрицы перестановки P на транспонированную к ней P^T равно единичной матрице I. Таким образом,

$$P^{-1} = P^T,$$

и матрица P является ортогональной. Если матрицу, имеющую n строк, умножить слева на P, то ее строки переупорядочиваются. Аналогично, умножение матрицы с η столбцами на P справа приводит к перестановке столбцов этой матрицы.

Матрицу P можно хранить в памяти ЭВМ в виде вектора с целочисленными компонентами: целое число в i-й позиции полагается равным столбцовому индексу единичного элемента i-й строки матрицы P.

2.7. ГАУССОВО ИСКЛЮЧЕНИЕ ПО СТОЛБЦАМ

Гауссово исключение — хорошо известная процедура, предназначенная для решения систем линейных уравнений. Исключение по столбцам, являющееся наиболее популярной версией этого алгоритма, мы изложим в первую очередь. С другой стороны, для разреженной матрицы, хранящейся в строчном формате, более эффективным оказывается алгоритм исключения по строкам. Эта версия будет описана в следующем параграфе.

Для простоты будем предполагать, что матрица A линейной системы (2.1) уже подготовлена для исключения, в том смысле, что диагональные элементы могут быть непосредственно использованы в качестве главных в том порядке, в каком они расположены. В общем случае главные элементы выбираются среди ненулевых элементов матрицы A, причем элемент, выбранный в качестве главного, переводится на главную диагональ посредством перестановки строк и столбцов. Выбор главных элементов осуществляют таким образом, чтобы в процессе исключения сохранялась

разреженность и обеспечивалась численная устойчивость. Вопрос выбора главных элементов или *упорядочения* изучается в гл. 4 и 5; в настоящей главе этот вопрос не рассматривается.

Рассмотрим систему (2.1). Гауссово исключение по столбцам состоит из n шагов. В результате выполнения k-го шага исключаются все ненулевые элементы матрицы, расположенные в k-м столбце под диагональю. На первом шаге ненулевые элементы первого столбца матрицы $A = A^{(1)}$ исключаются посредством поэлементного вычитания первой строки, умноженной на соответствующие скаляры, из всех остальных строк, имеющих ненулевой элемент в первом столбце. Элемент a_{11}, находящийся в строке, которая вычитается из остальных строк (в рассматриваемом случае это первая строка) и в столбце, поддиагональные элементы которого исключаются (в рассматриваемом случае это первый столбец), называется *главным элементом',* предполагается, что он отличен от нуля. Перед исключением первая строка нормируется путем деления всех ее ненулевых элементов на главный элемент. После исключения получается матрица $A^{(2)}$, для которой $a_{i1}^{(2)} = 0$ при $i > 1$ и $a_{11}^{(2)} = 1$.

На втором шаге в качестве главного выбирается элемент $a_{22}^{(2)}$. При этом вновь предполагается, что $a_{22}^{(2)} \neq 0$. Вторая строка нормируется, и все ненулевые элементы второго столбца, расположенные ниже диагонали, исключаются при помощи вычитания из соответствующих строк нормированной второй строки, умноженной на подходящие множители. Заметим, что поскольку $a_{21}^{(2)} = 0$, то элементы первого столбца не изменят при этом своих значений. Получается матрица $A^{(3)}$, у которой $a_{i1}^{(3)} = 0$ при $i > 1$, $a_{i2}^{(3)} = 0$ при $i > 2$ и $a_{11}^{(3)} = a_{22}^{(3)} = 1$. Другими словами, в своих первых двух столбцах матрица $A^{(3)}$ является верхней треугольной с единичной диагональю.

В начале k-го шага матрица $A^{(k)}$ имеет нулевые элементы в первых $k - 1$ столбцах под диагональю и единичные элементы в первых $k - 1$ позициях на диагонали.

Следующий пример иллюстрирует структуру матрицы $A^{(k)}$ в случае $n = 6$, $k = 3$:

$$A^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & x & x & x & x & x \\ & 1 & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \end{bmatrix} \end{array} \tag{2.14}$$

На *k-м* шаге в качестве главного выбирается элемент $a_{kk}^{(k)}$. Затем *k-я* строка нормируется, и поддиагональные элементы в *k-м* столбце матрицы $A^{(k)}$ исключаются путем вычитания умноженной на подходящие скаляры нормированной *k*-й строки из всех строк, имеющих ненулевые элементы в *k-м* столбце под диагональю. Получается матрица $A^{(k+1)}$ с нулевыми элементами в первых *k* столбцах ниже диагонали и с единицами в первых *k* позициях на диагонали.

Этот процесс продолжается до тех пор, пока в конце n-го шага не получится матрица $A^{(n+1)}$, которая содержит нулевые элементы под диагональю и единичные элементы на диагонали, являясь, таким образом, верхней треугольной с единичной диагональю. Этот алгоритм можно записать в удобных обозначениях, используя элементарные матрицы D_k и L_k^C из § 2.3. Каждый шаг гауссова исключения по столбцам эквивалентен умножению матрицы $A^{(k)}$ слева на $(L_k^C)^{-1} D_k^{-1}$:

$$A^{(k+1)} = (L_k^C)^{-1} D_k^{-1} A^{(k)}, \tag{2.15}$$

где

$$\begin{aligned} (D_k)_{kk} &= a_{kk}^{(k)}, \\ (L_k^C)_{ik} &= a_{ik}^{(k)} \text{ при } i > k, \end{aligned} \tag{2.16}$$

и $A^{(1)} = A$. Напомним, что согласно свойству 2.4 (г), внедиагональные элементы $(L_k^C)^{-1}$ равны внедиагональным элементам L_k^C, взятым с обратным знаком, т. е. $-a_{ik}^{(k)}$

Таким образом, вводя для единообразия тривиальное умножение слева матрицы $D_n^{-1} A^{(n)}$ на $(L_n^C)^{-1} = I$ (т. е. на единичную матрицу) в формулу для $A^{(n+1)}$, мы получаем

$$(L_n^C)^{-1} D_n^{-1} \ \ldots \ (L_2^C)^{-1} D_2^{-1} (L_1^C)^{-1} D_1^{-1} A = U, \tag{2.17}$$

где $U = A^{(n+1)}$ — верхняя треугольная матрица с единичной диагональю, причем

$$u_{kj} = (D_k)_{kk}^{-1} a_{kj}^{(k)} \equiv a_{kj}^{(k+1)} \text{ при } j > k. \tag{2.18}$$

Отсюда получается факторизованная форма матрицы *A:*

$$A = LU, \tag{2.19}$$

где

$$L = D_1 L_1^C D_2 L_2^C \ \ldots \ D_n L_n^C \tag{2.20}$$

является нижней треугольной матрицей (согласно свойству 2.5 (а)) с ненулевыми диагональными элементами. Поскольку произведения треугольных матриц в (2.20) вычисляются посред-

ством суперпозиции, выражения для элементов матрицы L легко могут быть получены из уравнений (2.16):

$$l_{ik} = a_{ik}^{(k)} \text{ при } i \geqslant k. \tag{2.21}$$

Используя соотношения (2.18) и (2.21), можно получить формулу, показывающую, как выражается $A^{(k+1)}$ через $A^{(k)}$,

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}u_{kj}, \; i > k, \; j > k, \tag{2.22}$$

откуда можно вывести полные выражения для элементов матрицы $A^{(k)}$:

$$a_{ij}^{(k)} = a_{ij} - \sum_{m=1}^{k-1} l_{im}u_{mj}, \; i > k, \; j > k. \tag{2.23}$$

Элиминативная форма обратной матрицы получается из соотношения (2.19):

$$A^{-1} = U^{-1}L^{-1}. \tag{2.24}$$

На практике вычисление матриц U^{-1} и L^{-1} никогда не выполняется явно. Величины, получающиеся в процессе исключения, обычно записываются на место элементов матрицы A в виде таблицы, представляющей две различные матрицы и называемой иногда *таблицей множителей* [Tinney, Walker, 1967]:

$$\begin{matrix} (D_1)_{11}^{-1} & u_{12} & u_{13} & \cdots \\ (L_1^C)_{21} & (D_2)_{22}^{-1} & u_{23} & \cdots \\ (L_1^C)_{31} & (L_2^C)_{32} & (D_3)_{33}^{-1} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{matrix} \tag{2.25}$$

Сначала рассмотрим процесс формирования этой таблицы. В силу соотношений (2.16), ее нижний треугольник получается, если оставлять элементы матрицы $a_{ik}^{(k)}$ там, где они были перед исключением; эти элементы в точности те, которые должны быть исключены на k-м шаге. Диагональная часть таблицы получается, если на каждом шаге записывать на место диагонального элемента k-й строки величину, обратную к нему. Указанная обратная величина получается при нормировании k-й строки, так как вычисления выполняются быстрее, если определяется величина, обратная главному элементу, которая затем умножается на все ненулевые элементы k-й строки по сравнению со случаем, когда ненулевые элементы строки делятся на главный элемент. Кроме того, при использовании таблицы для обратной подстановки выгодно иметь непосредственный доступ к обращенным главным элементам. В верхнем треугольнике таблицы оставляются элементы матрицы U, получающиеся в процессе исключения, как

указано в соотношении (2.18). Численный пример, иллюстрирующий построение таблицы множителей, дан в § 2.15.

Теперь рассмотрим способ использования таблицы множителей. Обычно она применяется для решения системы (2.1) по формуле

$$x = A^{-1}b, \tag{2.26}$$

которую, используя (2.24), можно переписать в виде

$$x = U^{1}L^{-1}b. \tag{2.27}$$

Из соотношения (2.20) получаем

$$L^{-1} = (L_n^C)^{-1} D_n^{-1} \ldots (L_2^C)^{-1} D_2^{-1} (L_1^C)^{-1} D_1^{-1}. \tag{2.28}$$

Вся необходимая информация о матрицах D_k^{-1} содержится в таблице множителей. Доступ к матрицам $(L_k^C)^{-1}$ также осуществляется непосредственно через таблицу множителей, так как, согласно свойству 2.4 (г), внедиагональные элементы этих матриц равны внедиагональным элементам матриц L_k^C, взятым с обратным знаком. Таким образом, умножение матрицы L^{-1} на вектор можно легко осуществить с использованием таблицы; соответствующая процедура обычно называется *прямой подстановкой.*[1]) Для того чтобы получить *x,* необходимо умножить матрицу U^{-1} на вектор, полученный в результате прямой подстановки. Для этой цели представим U в виде произведения n верхних столбцовых элементарных матриц:

$$U = U_n^C \ldots U_2^C U_1^C. \tag{2.29}$$

Согласно свойству 2.4 (д), это произведение получается путем суперпозиции; таким образом, элементы матриц U_k^C — это те самые числа, которые хранятся в верхнем треугольнике таблицы (2.25). Отсюда имеем

$$U^{-1} = (U_1^C)^{-1} (U_2^C)^{-1} \ldots (U_n^C)^{-1}, \tag{2.30}$$

и произведение матрицы U'^{l} на вектор $b \sim^{X} b$ можно получить, используя тот факт, что внедиагональные элементы $(U_k^C)^{-1}$ равны элементам U_k^C, хранящимся в таблице, взятым с обратным знаком (свойство 2.4 (г)). Эту процедуру называют *обратной подстановкой.* Прямая и обратная подстановки более детально обсуждаются в § 2.13.

Описанная выше процедура может быть использована для умножения A^{-1} на произвольную матрицу. Здесь полезно напомнить, что множители, из которых построены L и U, хранятся

[1]) Или прямым ходом. — *Прим. перее.*

в таблице. Следовательно, произведение матрицы $A = LU$ на произвольный вектор или матрицу можно получить, используя эту таблицу. Оказывается даже возможным использовать таблицу множителей для решения специальных двухкомпонентных гибридных систем, где вектором неизвестных является $(b_1, b_2, \ldots, b_k, x_{k+1}, \ldots, x_n)^T$ вместо x. Такие системы рассматривались в работе [Tinney, Walker, 1967].

Если требуется решить систему $Ax = b$ только для одного вектора правой части b или для небольшого их количества, то формирование всей таблицы множителей можно не производить; можно просто расширить матрицу A, приписав к ней вектор-столбец b, выполнить над расширенной матрицей Все операции исключения и затем применить обратную подстановку к полученному вектору-столбцу.

Разложение вида (2.5) можно получить из уравнения (2.19) при $L' = LD^{-1}$, где L' получается нижней треугольной с единичной диагональю, если D — диагональная матрица с элементами, равными диагональным элементам матрицы L. Из уравнения (2.20) получаем

$$D = D_1 D_2 \ldots D_n, \qquad (2.31)$$

и диагональные элементы матрицы D являются числами, обратными к диагональным элементам таблицы (2.25). Этот результат особенно важен, если матрица A симметрична, так как в этом случае $L' = U^T$ и

$$A = U^T D U. \qquad (2.32)$$

Поэтому нет необходимости вычислять нижний треугольник таблицы (2.25), который можно получить транспонированием ее верхнего треугольника. Исключение производится только над элементами верхнего треугольника и главной диагонали, за счет чего экономится почти половина объема памяти и вычислений. Конечно, реализация соответствующей вычислительной процедуры встречается с некоторыми затруднениями, в основном из-за того, что элементы нижнего треугольника матрицы A, которые должны быть исключены, не хранятся в памяти ЭВМ. Эта ситуация детально изучается в § 7.23, где рассматриваются матрицы, хранящиеся в разреженном строчном формате.

2.8. ГАУССОВО ИСКЛЮЧЕНИЕ ПО СТРОКАМ

Для разреженных матриц, хранящихся в строчном формате, исключение по строкам намного более эффективно, чем исключение по столбцам. В обоих случаях получаются одинаковые численные результаты и затрачивается одно и то же количество арифметических операций. Повышение эффективности достигается

за счет того, что элементы матрицы используются в естественном порядке, т. е. в том же порядке, в каком они хранятся. Гауссово исключение для матрицы A по строкам выполняется в n шагов. В начале k-го шага имеется матрица $A^{(k)}$ содержащая в строках с первой по $(k-1)$-ю нулевые элементы левее диагонали и единицы в первых $(k-1)$ позициях на диагонали. Следующий пример иллюстрирует структуру матрицы $A^{(k)}$ в случае $n = 6$, $k = 4$:

$$A^{(4)} = \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & x & x & x & x & x \\ & 1 & x & x & x & x \\ & & 1 & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \end{bmatrix} \end{array} \qquad (2.33)$$

На k-*м* шаге производится исключение ненулевых элементов k-й строки, расположенных левее диагонали, путем вычитания взятых с соответствующими множителями первой строки, второй строки, ..., $(k-1)$-й строки в указанном порядке. Затем каждая строка нормируется путем деления всех ее элементов на диагональный элемент. Рассмотрим более подробно первые три шага этого алгоритма.

На первом шаге мы делим все элементы первой строки матрицы A на диагональный элемент, находящийся в позиции (1,1).

На втором шаге мы исключаем элемент, находящийся в позиции (2,1), путем вычитания первой строки, умноженной на этот элемент, из второй строки. Затем нормируем вторую строку делением всех ее элементов на диагональный элемент, находящийся в позиции (2,2).

На третьем шаге исключаем элемент, находящийся в позиции (3,1), вычитанием первой строки, умноженной на соответствующий скаляр, из третьей строки. Затем исключаем элемент, находящийся в позиции (3,2), вычитанием второй строки, умноженной на соответствующий скаляр, из третьей строки и, наконец, делим третью строку на диагональный элемент, находящийся в позиции (3,3), и т. д.

Нетрудно проверить, что описанная процедура дает в точности тот же результат, что и исключение по столбцам [Tinney, Walker, 1967], которое было описано в § 2.7. Например, в обоих алгоритмах все элементы первой строки, расположенные справа от диагонали, получаются в результате нормирования первой строки. Все элементы второй строки, расположенные справа от диагонали, в обоих случаях получаются вычитанием первой строки, умноженной на одни и те же скаляры, с последующим нормированием

полученной второй строки. Все элементы третьей строки, расположенные справа от диагонали, получаются в обоих случаях при помощи вычитания первой и второй строк (совпадающих в обоих случаях), умноженных на те же самые множители, из третьей строки с последующим нормированием третьей строки. Единственным различием между двумя алгоритмами является порядок, в котором производится исключение, однако в обоих случаях в итоге получается одна и та же верхняя треугольная матрица U, заданная уравнением (2.17).

Исключения, выполняемые на k-м шаге строчного алгоритма, эквивалентны умножению слева на строчную элементарную матрицу $\left(L_k^R\right)^{-1}$, в то время как каждое нормирование эквивалентно умножению слева на диагональную элементарную матрицу D_k^{-1}. Так как произведение всех $\left(L_k^R\right)^{-1}$ и D_k^{-1} является нижней треугольной матрицей L', мы можем формально представить гауссово исключение по строкам следующим образом:

$$L' A = U. \tag{2.34}$$

Сравнивая это выражение с уравнением (2.17), получаем

$$(L')^{-1} = L. \tag{2.35}$$

Таким образом, получено то же самое треугольное разложение $A = LU$, что и в столбцовом алгоритме. И опять хранение матриц L и U можно организовать в виде таблицы множителей (см. (2.25)), если оставлять элементы в нижнем треугольнике в их окончательном виде перед самым исключением, элементы в верхнем треугольнике оставлять в тех же позициях, где они получаются, а элементы на диагонали заменять на величины, обратные к ним, непосредственно перед нормированием.

2.9. ИСКЛЮЧЕНИЕ ГАУССА — ЖОРДАНА

Алгоритм исключения Гаусса—Жордана по столбцам аналогичен гауссову исключению по столбцам; главное отличие состоит в том, что перед k-м шагом матрица $A^{(k)}$ имеет нулевые элементы в столбцах от первого до k-го как ниже, так и выше диагонали. Следующий пример иллюстрирует структуру матрицы $A^{(k)}$ в случае $n = 6$, $k = 3$:

$$A^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 1 & & x & x & x & x \\ & 1 & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \\ & & x & x & x & x \end{bmatrix} \end{array} \tag{2.36}$$

Шаг с номером k алгоритма состоит в исключении ненулевых элементов k-го столбца матрицы $A^{(k)}$, *расположенных* как *выше,* так и *ниже диагонали.* Сначала нормируется k-я строка, для чего все ее элементы делятся на диагональный элемент, находящийся в позиции (k, k). Затем нормированная k-я строка умножается на подходящие скаляры и вычитается из всех строк, имеющих ненулевые элементы в k-м столбце как выше, так и ниже диагонали. Таким образом, получается матрица $A^{(k+1)}$ с нулями во внедиагональных позициях первых k столбцов. Описанный процесс продолжается до тех пор, пока в результате выполнения последнего шага не получится единичная матрица $A^{(n+1)} \equiv I$. Каждый шаг исключения Гаусса—Жордана по столбцам эквивалентен умножению слева сначала на D_k^{-1}, а затем на полную столбцовую элементарную матрицу $(T_k^C)^{-1}$:

$$A^{(k+1)} = (T_k^C)^{-1} D_k^{-1} A^{(k)}, \tag{2.37}$$

где $A^{(1)} = A$ и

$$(D_k)_{kk} = a_{kk}^{(k)}, \tag{2.38 а}$$

$$(T_k^C)_{ik} = a_{ik}^{(k)} \text{ при } i \neq k. \tag{2.38 б}$$

Таким образом, получаем

$$(T_n^C)^{-1} D_n^{-1} \ldots (T_2^C)^{-1} D_2^{-1} (T_1^C)^{-1} D_1^{-1} A = I. \tag{2.39}$$

Факторизованная форма матрицы A записывается как

$$A = D_1 T_1^C D_2 T_2^C \ldots D_n T_n^C, \tag{2.40}$$

а *факторизованная форма обратной матрицы* в терминах столбцовых матриц имеет вид

$$A^{-1} = (T_n^C)^{-1} D_n^{-1} \ldots (T_2^C)^{-1} D_2^{-1} (T_1^C)^{-1} D_1^{-1}. \tag{2.41}$$

Тесная связь этого разложения и элиминативной формы обратной матрицы (2.24) будет обсуждаться в § 2.10. Результаты, полученные в процессе исключения, обычно записываются в виде таблицы множителей:

$$\begin{matrix} (D_1)_{11}^{-1} & (T_2^C)_{12} & (T_3^C)_{13} \ldots \\ (T_1^C)_{21} & (D_2)_{22}^{-1} & (T_3^C)_{23} \ldots \\ (T_1^C)_{31} & (T_2^C)_{32} & (D_3)_{33}^{-1} \ldots \\ \vdots & \vdots & \vdots \quad \searrow \end{matrix} \tag{2.42}$$

Согласно уравнениям (2.38), эту таблицу можно сформировать, если оставлять каждый внедиагональный элемент $a_{ik}^{(k)}$ там, где он был получен. Диагональ получается как и в гауссовом

исключении запоминанием величин, обратных к диагональным элементам, используемым при нормировании каждой строки. Таким образом, нижний треугольник и диагональ получившейся таблицы идентичны нижнему треугольнику и диагонали таблицы множителей для метода Гаусса. Выражения (2.40) и (2.41) показывают, как использовать таблицу (2.42). Решая систему линейных уравнений по формуле $x = A^{-1}b$, достаточно использовать разложение (2.41), где матрицы $(T_k^C)^{-1}$ получаются взятием внедиагональных элементов k-го столбца таблицы (2.42) с обратными знаками (свойство 2.4 (г)). Доступ к матрицам D_k^{-1} осуществляется через таблицу непосредственно. Произведение матрицы A на любую матрицу или вектор также можно вычислить при помощи таблицы, используя для этого разложение (2.40).

Исключение Гаусса—Жордана может также производиться по строкам. Столбцовая версия использует сложение k-й строки, умноженной на некоторые числа, со всеми остальными строками для того, чтобы уничтожить ненулевые внедиагональные элементы k-го столбца. Концептуально этот процесс можно понимать как конструирование новых уравнений, являющихся линейными комбинациями исходных. С другой стороны, при исключении Гаусса—Жордана по строкам k-й *столбец,* умноженный на некоторые числа, складывается со всеми остальными столбцами таким образом, что внедиагональные элементы k-й *строки* становятся нулевыми. Этот процесс можно трактовать как конструирование новых *неизвестных,* которые являются линейными комбинациями исходных и удовлетворяют линейным уравнениям, некоторые коэффициенты которых нулевые. Другую интерпретацию можно дать, если забыть о системе линейных уравнений и рассматривать строчный алгоритм как процесс приведения к диагональному виду матрицы A^T, транспонированной к A, по столбцам. Сделав это, можно получить уравнение, аналогичное (2.41):

$$(A^T)^{-1} = (T_n'^C)^{-1}(D_n')^{-1} \ldots (T_2'^C)^{-1}(D_2')^{-1}(T_1'^C)^{-1}(D_1')^{-1}, \quad (2.43)$$

транспонируя которое и применяя соотношение $(A^T)^{-1} = (A^{-1})^T$, мы приходим к формуле

$$A^{-1} = (D_1')^{-1}(T_1'^R)^{-1}(D_2')^{-1}(T_2'^R)^{-1} \ldots (D_n')^{-1}(T_n'^R)^{-1}. \quad (2.44)$$

Полученное выражение (2.44) дает факторизованную формулу обратной матрицы в терминах элементарных строчных матриц. Исключение по строкам эквивалентно умножению матрицы A *справа* на множители, указанные в правой части (2.44). Элементы матриц в правой части (2.44), отличные от нуля и единицы, записываются в виде таблицы множителей обычным образом, и эта

таблица может быть использована как для решения системы линейных уравнений, так и для умножения матриц A или A^{T} на любую матрицу или вектор.

2.10. СВЯЗЬ МЕЖДУ ЭЛИМИНАТИВНОЙ ФОРМОЙ ОБРАТНОЙ МАТРИЦЫ И ФАКТОРИЗОВАННЫЙ ФОРМОЙ ОБРАТНОЙ МАТРИЦЫ

Из предыдущего параграфа должно быть ясно, что исключение Гаусса—Жордана по столбцам можно произвести столь же успешно, если исключить сначала все ненулевые элементы в нижнем треугольнике матрицы A, и только затем — все ненулевые элементы, расположенные в верхнем треугольнике. Фактически, если начинать с верхнего левого угла матрицы A, можно исключать верхние и нижние части столбцов в любом порядке, с тем лишь условием, что верхние части исключаются по порядку [1]), нижние части также исключаются по порядку, и верхняя часть любого столбца с номером k исключается *после* нижней части предыдущего столбца. Высказанное утверждение справедливо в силу того что $(k+1)$-я строка получается в окончательном виде непосредственно после исключения нижних частей столбцов от первого до k-го и нормирования $(k+1)$-й строки; $(k+1)$-я строка затем может быть использована либо немедленно, либо на более поздней стадии для исключения верхней части $(k+1)$-го столбца при условии, что верхние части столбцов с первого по k-й предварительно уже были исключены. Эти факты могут быть установлены формально, если использовать свойства элементарных матриц (см. § 2.4). Используем свойство 2.4 (в) для того, чтобы выразить T_k^C следующим образом:

$$T_k^C = L_k^C U_k^C, \tag{2.45}$$

где U_k^C строится по верхней части матрицы T_k^C, а L_k^C — по нижней части T_k^C, и поэтому матрица L_k^C та же, что и в уравнении (2.16); из формулы (2.39) получаем:

$$(U_n^C)^{-1}(L_n^C)^{-1} D_n^{-1} \ldots (U_2^C)^{-1}(L_2^C)^{-1} D_2^{-1} (U_1^C)^{-1}(L_1^C)^{-1} D_1^{-1} A = I. \tag{2.46}$$

Теперь в силу свойств 2.4 (а) и 2.4 (б) мы видим, что $(U_i^C)^{-1}$, являющаяся верхней столбцовой матрицей, перестановочна с любой матрицей D_j^{-1} или $(L_k^C)^{-1}$, если $i < j$ и $i < k$; таким образом, уравнение (2.46) можно записать в следующем виде:

$$[(U_n^C)^{-1} \ldots (U_2^C)^{-1}(U_1^C)^{-1}]\,[(L_n^C)^{-1} D_n^{-1} \ldots (L_2^C)^{-1} D_2^{-1} (L_1^C)^{-1} D_1^{-1} A] = I. \tag{2.47}$$

[1]) Имеется в виду естественный порядок 1, 2, ... *п.—Прим. перев.*

Выражение, стоящее в уравнении (2.47) в правых квадратных скобках, равно U в силу уравнения (2.17). Следовательно,

$$(U_n^C)^{-1} \ldots (U_2^C)^{-1} (U_1^C)^{-1} = U^{-1}. \qquad (2.48)$$

Уравнение (2.48) дает факторизацию матрицы U^{-1} в терминах верхних столбцовых элементарных матриц $(U_k^C)^{-1}$; так как произведения в уравнении (2.48) вычисляются путем суперпозиции (по свойству 2.4 (д)), то если хранить нетривиальные столбцы матриц $(U_k^C)^{-1}$ в виде таблицы, результат будет идентичен записи матрицы U^{-1}. Заметим теперь, что верхний треугольник таблицы множителей, получающийся в методе Гаусса—Жордана, формируется из нетривиальных столбцов матриц U_k^C, как показано в (2.38) и (2.45); эти столбцы отличаются от соответствующих столбцов матриц $(U_k^C)^{-1}$ только алгебраическими знаками внедиагональных элементов (по свойству 2.4 (г)). Поэтому мы приходим к заключению, что верхний треугольник таблицы Гаусса—Жордана идентичен верхнему треугольнику матрицы U^{-1} с обращенными знаками внедиагональных элементов. Кроме того, сравнивая соотношение (2.38 (б)) при $i > k$ с уравнениями (2.16), мы находим, что нижний треугольник таблицы Гаусса—Жордана совпадает с нижним треугольником матрицы L, хранимой в таблице множителей для элиминативной формы обратной матрицы. Эти факты проиллюстрированы на числовых примерах в § 2.15. Поставим теперь вопрос о том, какая из двух таблиц множителей является более разреженной: таблица (2.25) для элиминативной формы обратной матрицы или таблица (2.42) для факторизованной формы? Ясно, что фактически производится сравнение разреженности матриц. Этот вопрос исследовался в работе [Brayton, Gustavson, Willoughby, 1970], где установлено, что если все диагональные элементы матрицы U отличны от нуля, то матрица U^{-1} всегда требует для хранения больше памяти, чем U [1]). Это одна из причин, по которым для решения линейных уравнений почти всегда используется именно гауссово исключение в любой из его форм.

2.11. РАЗЛОЖЕНИЕ ХОЛЕЦКОГО СИММЕТРИЧНОЙ ПОЛОЖИТЕЛЬНО ОПРЕДЕЛЕННОЙ МАТРИЦЫ

Если матрица A — симметричная положительно определенная, то существует и единственно разложение вида

$$A = U^T U, \qquad (2.49)$$

[1]) Имеется в виду объем памяти, необходимый для хранения ненулевых элементов матрицы. — *Прим. перев.*

где U — верхняя треугольная матрица с положительными диагональными элементами (свойство 7 из § 2.2). В этом параграфе мы выясним, как факторизация (2.49) может быть получена непосредственно. Мы также выведем некоторые оценки, которые играют важную роль при анализе ошибок округления в симметричном исключении. Запишем заданную матрицу в блочной форме:

$$A = \begin{bmatrix} \alpha & b^T \\ b & B \end{bmatrix} \tag{2.50}$$

где $\alpha \equiv \alpha_{11} > 0$ и B — симметричная матрица порядка $n - 1$. Теперь матрицу $A \equiv A^{(1)}$ можно факторизовать следующим образом:

$$A = \begin{bmatrix} \alpha^{1/2} & 0^T \\ b/\alpha^{1/2} & I \end{bmatrix} \begin{bmatrix} 1 & 0^T \\ 0 & A^{(2)} \end{bmatrix} \begin{bmatrix} \alpha^{1/2} & b^T/\alpha^{1/2} \\ 0 & I \end{bmatrix}, \tag{2.51}$$

где 0 — нулевой вектор-столбец, а

$$A^{(2)} = B - \frac{b}{\alpha^{1/2}} \frac{b^T}{\alpha^{1/2}} \tag{2.52}$$

— матрица порядка $n - 1$.

Ясно, что матрица $A^{(2)}$ симметрична; можно также доказать, что она положительно определена. Для доказательства мы, используя разбиение (2.50) матрицы A и разбиение

$$y = \begin{bmatrix} y_1 \\ z \end{bmatrix} \tag{2.53}$$

вектора y, где z — некоторый ненулевой вектор размерности $n - 1$, запишем неравенство (2.6) в виде

$$y^T A y = \alpha y_1^2 + 2y_1 z^T b + z^T B z > 0. \tag{2.54}$$

Если положить $y_1 = -z^T b/\alpha$ и использовать уравнение (2.52), то получим

$$z^T A^{(2)} z > 0, \tag{2.55}$$

и так как z — произвольный вектор, то положительная определенность матрицы $A^{(2)}$ доказана.

Так как матрица $A^{(2)}$ симметричная и положительно определенная, она может быть факторизована тем же образом, что и матрица A (см. уравнение (2.51)). В результате получается симметричная положительно определенная матрица $A^{(3)}$ порядка $n - 2$, которая в свою очередь также может быть факторизована. Указанная процедура повторяется до тех пор, пока не будет получена матрица $A^{(n)}$ порядка 1. Все матрицы в последователь-

ности $A^{(1)} \equiv A$, $A^{(2)}$, ..., $A^{(\eta)}$ являются симметричными положительно определенными, и порядок матрицы $A^{(k)}$ равен $n - k + 1$. После окончания этой процедуры для матрицы Л получается следующее выражение:

$$A = L_1'^C L_2'^C \dots L_n'^C U_n'^R \dots U_2'^R U_1'^R, \tag{2.56}$$

где $L_i'^C$ и U_i^R — матрицы того же типа, что и элементарные матрицы, определенные в § 2.3, но отличающиеся тем, что их диагональные элементы положительны и, вообще говоря, отличны от единицы, а матрицы $L_i'^C$ равны $(U_i'^R)^T$. Детали оставляются читателю, который может непосредственно убедиться в том, что окончательный вид разложения (2.49) получается простой суперпозицией матриц.

Практическая реализация алгоритма Холецкого для разреженных матриц является, однако, весьма отличной от вышеописанной процедуры. Она будет обсуждаться позднее в § 2.12. Изложение материала в настоящем параграфе больше подходит для теоретических целей, в частности для вывода оценок величины элементов матриц $A^{(k)}$. Эти оценки в свою очередь необходимы для анализа ошибок округления в алгоритме Холецкого, который будет приведен в гл. 3, и для обоснования наиболее важного свойства данного метода: в случае симметричных положительно определенных матриц выбор главных элементов на главной диагонали обеспечивает численную устойчивость.

Найдем теперь оценки для модулей элементов матриц $A^{(k)}$. Согласно свойству 5 § 2.2, мы знаем, что в силу положительной определенности $A^{(k)}$ выполняется неравенство

$$\left|a_{ij}^{(k)}\right| \leqslant a_k \text{ для всех } i, j, \tag{2.57}$$

где a_k — наибольший диагональный элемент $A^{(k)}$. Теперь в силу соотношения (2.52) очевидно, что диагональные элементы каждой матрицы $A^{(k)}$, положительные в силу свойства 4 § 2.2, получаются путем вычитания неотрицательных чисел из соответствующих диагональных элементов $A^{(k-1)}$, также положительных. Таким образом, как показано в [Wilkinson, 1961]:

$$a_1 \geqslant a_2 \geqslant \dots \geqslant a_n, \tag{2.58}$$

где a_k — наибольший диагональный элемент матрицы $A^{(k)}$. Окончательно мы можем заключить, что $a \equiv a_1$ является верхней границей для модулей элементов всех матриц $A^{(k)}$:

$$\left|a_{ij}^{(k)}\right| \leqslant a \text{ для всех } i, j, k\ ^{1)}. \tag{2.59}$$

---

[1]) Должно быть выполнено условие $1 \leq i \leq n - k + 1$, $1 \leq i \leq n - k + 1$. — *Прим. перев.*

Неравенство (2.59) показывает, что любой элемент редуцированной матрицы $A^{(k)}$ по модулю не превосходит значения a наибольшего диагонального элемента исходной матрицы A.

2.12. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ РАЗЛОЖЕНИЯ ХОЛЕЦКОГО

На практике, так как матрица A симметрична, в памяти хранится только ее диагональ и верхний треугольник. Матрицы $A^{(k)}$ никогда не строятся явным образом. Верхний треугольный множитель U строится непосредственно в отдельно отведенной области памяти, а не записывается на место элементов A, поскольку U обычно является менее разреженной, чем матрица A, и требует введения другой структуры данных. Хотя мы не можем говорить о методе Холецкого как об «исключении», поскольку фактически мы ничего не исключаем, мы увидим, что этот алгоритм почти идентичен гауссову исключению.

Анализ уравнения (2.51) показывает, что, как только вычислен квадратный корень из $a_{11}^{(1)}$ и остальные элементы первой строки разделены на это значение, первая строка матрицы U уже получена. Затем мы можем вычислить элементы первой строки матрицы $A^{(2)}$, используя уравнение (2.52), и немедленно получить корень из $a_{11}^{(2)}$, после чего разделить остальные элементы первой строки $A^{(2)}$ на это значение; это дает нам вторую строку матрицы U. Заметим, что элементы строки $b^{\mathrm{T}}/\alpha^{1/2}$ уже были вычислены и хранятся в качестве первой строки матрицы U; таким образом, при использовании уравнения (2.52) необходимы умножения и вычитания. Заметим еще, что этот шаг в точности аналогичен гауссову исключению элемента a_{21} с использованием элемента a_{11} как главного, хотя a_{11} в этом случае не равен единице.

Теперь мы переходим к третьей строке матрицы A и используем уравнение (2.52), чтобы получить вторую строку матрицы $A^{(2)}$. Этот шаг аналогичен исключению элемента a_{31}. Затем опять для третьей строки матрицы A мы выполняем вычисления, необходимые для получения первой строки матрицы $A^{(3)}$; этот шаг аналогичен исключению элемента a_{32}. Затем вычисляется квадратный корень из только что полученного диагонального элемента и все остальные элементы третьей строки делятся на это значение. В результате получается третья строка матрицы U в своем окончательном виде. Действуя описанным образом и далее, мы получаем матрицу U в точности таким же образом, как если бы мы производили гауссово исключение; разница заключается лишь в том, что в конце каждого шага вместо запоминания величины, обратной к диагональному элементу, и умножения внедиагональных элементов строки на это значение, мы запоминаем величину,

обратную к квадратному корню из диагонального элемента, и умножаем внедиагональные элементы на это значение. Таблица множителей, представляющая разложение Холецкого (2.49), состоит из верхнего треугольника матрицы U и величин, обратных к диагональным элементам матрицы U. При этом элементы нижнего треугольника матрицы A не хранятся и не используются. Если обозначить через $D^{1/2}$ диагональную матрицу, элементы которой совпадают с (положительными) диагональными элементами матрицы U, можно записать $U = D^{1/2}U'$, где матрица U' — верхняя треугольная с единичной диагональю (в отличие от матрицы U); уравнение (2.49) тогда может быть переписано в следующем виде:

$$A = U'^{T}DU', \tag{2.60}$$

где $D = (D^{1/2})^2$ —диагональная матрица с положительными диагональными элементами. Теперь легко провести сравнение разложения Холецкого с гауссовым исключением. Соотношение (2.60), полученное из разложения Холецкого, идентично уравнению (2.5) для гауссового исключения в случае, когда матрица A симметрична. Диагональ таблицы множителей для разложения Холецкого задается матрицей $D^{-1/2}$, а верхний треугольник — матрицей U. Диагональ таблицы множителей для гауссова исключения задается матрицей D^{-1}, а верхний треугольник — матрицей U'.

Числовой пример, иллюстрирующий разложение Холецкого, дан в § 2.15, а алгоритм для вычисления разложения в форме (2.60), не требующей извлечения квадратных корней, для матрицы A, заданной в разреженном строчном формате, дается в записи на Фортране в гл. 9.

2.13. ПРЯМАЯ И ОБРАТНАЯ - ПОДСТАНОВКИ

Решение линейной системы (2.1) является несложной задачей, если задано треугольное разложение (2.2) матрицы A. Искомое решение получается решением уравнения (2.3) относительно w, а затем уравнения (2.4) относительно x. Для ясности изложения приведем эти уравнения еще раз:

$$L\omega = b, \tag{2.61}$$

$$Ux = \omega. \tag{2.62}$$

Если матрица A — симметричная положительно определенная и получено ее разложение Холецкого $A = U^{T}U$, то процедура остается той же самой, за исключением того, что вместо L используется матрица U^{T}. Если матрица A общего вида представлена в виде $A = LDU$, то необходимо выполнить три шага, второй из которых есть тривиальная процедура решения линейной системы с диагональной матрицей коэффициентов.

Решение системы (2.61) называется *прямой подстановкой.* Так как матрица L — нижняя треугольная, то первое уравнение имеет следующий вид:

$$l_{11}\omega_1 = b_1,$$

где $l_{11} \neq 0$ в силу того, что мы предполагаем невырожденность матрицы A. Отсюда получаем $\omega_1 = b_1 / l_{11}$. Теперь можно вычесть первый столбец матрицы L, умноженный на ω_1, из вектора b. Уравнение (2.61) преобразуется в треугольную систему порядка $(n - 1)$ с угловой главной подматрицей матрицы L в качестве матрицы коэффициентов. Теперь можно вычислить ω_2,, и приведенные рассуждения могут повторяться рекуррентно и далее, пока не будет завершено вычисление всех компонент вектора ω. Описанная процедура, конечно, эквивалентна умножению матрицы L^{-1}, заданной в (2.28), на вектор b. Если обозначить через n_L число внедиагональных ненулевых элементов матрицы L, то можно убедиться в том, что прямая подстановка требует n операций деления, n_L операций умножения и n_L операций сложения.

Если известен вектор w, то систему (2.62) можно решить при помощи *обратной подстановки.* Так как мы предполагаем, что матрица U имеет единичную диагональ, то последнее уравнение (2.62) имеет вид

$$x_n = w_n,$$

так что x_n известно. Вычитая произведение последнего столбца матрицы U на ω_n из вектора ω, мы приходим к треугольной системе порядка $(n - 1)$ с ведущей главной подматрицей матрицы U в качестве матрицы коэффициентов. Продолжая указанные преобразования рекуррентно, мы получаем решение x. Описанная процедура эквивалентна умножению матрицы U^{-1}, заданной уравнением (2.30), на до. Обозначив через n_U число внедиагональных ненулевых элементов матрицы U, можно показать, что обратная подстановка требует n_U умножений и n_U сложений. Алгоритмы прямой и обратной подстановки для разреженных матриц, хранящихся в строчном формате, даются в записи на Фортране в гл. 9.

2.14. О ВЫЧИСЛИТЕЛЬНЫХ ЗАТРАТАХ

Занимаясь технологией разреженных матриц, важно знать объем памяти, требуемой алгоритмом, и число операций, производимых при его выполнении. Располагая этой информацией, пользователь может оценить реальную стоимость вычислений, принимая во внимание также алгоритм, описывающий распределение накладных затрат, связанных с применением того или иного вычислительного оборудования. В случае вычисления треуголь-

ной факторизации и решения при помощи прямой и обратной подстановки основные теоремы были приведены в работе [Rose, 1972] для разреженных симметричных матриц, и в работе [Bunch, Rose, 1974] для разреженных матриц общего вида, в то время как эффект применения специальных схем хранения был рассмотрен в работе [Bunch, 1974 в] (с незначительными опечатками на стр. 851; соответствующий результат приведен в нашей табл. 2.1 в исправленном виде).

Упомянутые теоремы дают возможность оценить объем памяти и число операций в терминах следующих величин:

r_i^U = число внедиагональных ненулевых элементов в *i-й* строке матрицы *U;*

c_j^L = число внедиагональных ненулевых элементов в *j*-м столбце матрицы *L;*

n = порядок матрицы.

При подсчете операций число делений включается в число умножений, а число вычитаний — в число сложений. Сложения с нулями также учитываются при подсчете сложений, а возможные взаимные уничтожения ненулевых элементов не принимаются во внимание. Конечно, было бы желательно выразить эти величины в терминах характеристик, связанных непосредственно с *исходной* матрицей *A,* и уметь вычислять затраты до того, как начнется исключение; эта задача ставится для симметричных матриц в гл. 4.

Результаты, получающиеся в некоторых случаях, приведены в табл. 2.1 и 2.2. Если матрица A симметрична, то $r_i = c_i$; при этом как объем памяти, так и число операций уменьшаются почти в 2 раза, что было отмечено в конце § 2.7. Для плотных матриц все элементы рассматриваются как ненулевые, даже если их значение может быть равным нулю. Все элементы, расположенные внутри ленты в ленточной матрице, также рассматриваются как ненулевые. В обоих случаях можно опускать операции над нулевыми элементами, выполняя тем самым меньшее число операций, чем указанное в таблицах. Однако такой способ обработки требует дополнительной проверки для каждого элемента, и суммарная стоимость вычислений скорее увеличится, чем уменьшится, если доля нулевых элементов невелика.

Для полных матриц $r_i^U = c_i^L = n - i$. В этом случае результаты, приведенные в табл. 2.1 и 2.2, оказываются выраженными в терминах характеристик исходной матрицы (а именно, через порядок матрицы n).

Для ленточных матриц величины r_i^U и c_i^L также могут быть выражены в терминах свойств исходной матрицы. Обе полуширины ленты равны β, а ширина ленты равна $2\beta + 1$. Главные

Таблица 2.1. Объем памяти и число операций для факторизации. Суммирование производится от $i = 1$ до $i = n - 1$ включительно. Детальные пояснения можно найти в тексте

Матрица A	Число умножений и делений	Число сложений	Объем памяти
Заполненная симметричная	$\frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n$	$\frac{1}{6}n^3 - \frac{1}{6}n$	$\frac{1}{2}n^2 + \frac{1}{2}n$
Ленточная симметричная	$\frac{1}{2}\beta(\beta+3)n - \frac{1}{3}\beta^3 - \beta^2 - \frac{2}{3}\beta$	$\frac{1}{2}\beta(\beta+1)n - \frac{1}{3}\beta^3 - \frac{1}{2}\beta^2 - \frac{1}{6}\beta$	$(\beta+1)n - \frac{1}{2}\beta^2 - \frac{1}{2}\beta$
Разреженная симметричная	$\sum r_i^U (r_i^U + 3)/2$	$\sum r_i^U (r_i^U + 1)/2$	$n + \sum r_i^U$
Заполненная несимметричная	$\frac{1}{3}n^3 - \frac{1}{3}n$	$\frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n$	n^2
Ленточная несимметричная, диагональный выбор главного элемента	$(\beta+1)\beta n - \frac{2}{3}\beta^3 - \beta^2 - \frac{1}{3}\beta$	$\beta^2 n - \frac{2}{3}\beta^3 - \frac{1}{2}\beta^2 + \frac{1}{6}\beta$	$(2\beta+1)n - \beta^2 - \beta$
Ленточная несимметричная, частичный выбор главного элемента	$\leqslant (2\beta+1)\beta n - \frac{13}{6}\beta^3 - \frac{3}{2}\beta^2 - \frac{1}{3}\beta$	$\leqslant 2\beta^2 n - \frac{13}{6}\beta^3 - \beta^2 + \frac{1}{6}\beta$	$\leqslant (3\beta+1)n - \frac{5}{2}\beta^2 - \frac{3}{2}\beta$
Разреженная несимметричная	$\sum (r_i^U + 1) c_i^L$	$\sum r_i^U c_i^L$	$n + \sum (r_i^U + c_i^L)$

элементы выбираются на диагонали в порядке возрастания номера, если принят «диагональный выбор главного элемента» и заполнение может появиться только внутри ленты. Однако, если матрица несимметрична или незнакоопределена, обычно необходимым является выбор k-го главного элемента из k-го столбца редуцированной матрицы с последующей его переста-

Таблица 2.2. Число операций для прямой и обратной подстановки. Суммирование проводится от $i = 1$ до $i = n - 1$ включительно. Детальные пояснения можно найти в тексте

Матрица A	Число умножений	Число сложений
Заполненная симметричная или несимметричная	n^2	n^2-n
Ленточная симметричная или несимметричная с диагональным выбором главного элемента	$(2\beta+1)\,n-\beta^2-\beta$	$2\beta n-\beta^2-\beta$
Разреженная симметричная	$n+2\sum r_i^U$	$2\sum r_i^U$
Ленточная несимметричная, частичный выбор главного элемента	$\leqslant (3\beta+1)\,n-\frac{5}{2}\beta^2-\frac{3}{2}\beta$	$3\beta n-\frac{5}{2}\beta^2-\frac{3}{2}\beta$
Разреженная несимметричная	$n+\sum\left(r_i^U+c_i^L\right)$	$\sum\left(r_i^U+c_i^L\right)$

новкой в диагональную позицию посредством перестановки двух строк (см. § 1.12). Требуемая память в этом случае должна быть достаточна для хранения нижней полуленты ширины β и верхней полуленты ширины до 2β. Этот случай выделен в табл. 2.1 и 2.2 как «частичный выбор главного элемента», и приведенные там формулы дают лишь верхние оценки объема памяти и числа операций, так как реальная ширина верхней полуленты может не достигать значения 2^{β}.

В табл. 2.1 для матрицы A общего вида приводятся данные, отвечающие факторизации $A = LU$, а для симметричной матрицы A — данные, отвечающие факторизации $A = U^TDU$. Оценка объема памяти включает все те элементы, которые рассматриваются как ненулевые, но не включают накладные затраты памяти, подчас весьма ощутимые, если применяются связанные списки или разреженные форматы хранения. Во всех случаях дополнительно требуются η ячеек памяти для выполнения прямой или обратной подстановки. Результаты для факторизации матрицы и для решения системы даются отдельно друг от друга, так как в прикладных задачах может потребоваться выполнить факторизацию заданной матрицы только один раз и затем решить большое количество линейных систем с различными правыми частями.

2.15. ЧИСЛЕННЫЕ ПРИМЕРЫ

В этом параграфе при помощи численных примеров продемонстрированы различные формы факторизации матрицы, обсуждавшиеся на протяжении данной главы. Рассмотрим матрицу

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 2 & 3 & -1 \\ 1 & 1 & 4 \end{bmatrix}.$$

Используя гауссово исключение (любой его вариант — по строкам или по столбцам), получаем факторизацию вида

$$A = LU = \begin{bmatrix} 2 & & \\ 2 & 2 & \\ 1 & \frac{1}{2} & 4 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ & 1 & -1 \\ & & 1 \end{bmatrix};$$

соответствующая таблица множителей (2.25) имеет вид

$$\text{гауссова таблица} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 2 & \frac{1}{2} & -1 \\ 1 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

Матрица U^{-1} вычисляется, исходя из верхнего треугольника таблицы в соответствии с соотношением (2.30):

$$U^{-1} = \begin{bmatrix} 1 & -\frac{1}{2} & -1 \\ & 1 & 1 \\ & & 1 \end{bmatrix}.$$

Факторизацию вида $A = L'DU$, где L' — нижняя треугольная матрица с единичной диагональю, можно получить из факторизации $A = LU$, если записать нижний треугольный множитель в виде $L'D = L$ (см.. уравнение (2.5)). Результат имеет следуюший вид: .

$$A = \begin{bmatrix} 1 & & \\ 1 & 1 & \\ \frac{1}{2} & \frac{1}{4} & 1 \end{bmatrix} \begin{bmatrix} 2 & & \\ & 2 & \\ & & 4 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ & 1 & -1 \\ & & 1 \end{bmatrix}.$$

Если используется исключение Гаусса—Жордана по столбцам, то получается факторизация (2.40) вида $A = D_1T_1^CD_2T_2^CD_3T_3^C$, которую можно записать следующим образом:

$$A = \begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ 2 & 1 & \\ 1 & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & \frac{1}{2} & \\ & 1 & \\ & \frac{1}{2} & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & & 4 \end{bmatrix}\begin{bmatrix} 1 & & 1 \\ & 1 & -1 \\ & & 1 \end{bmatrix},$$

а соответствующая таблица множителей (2.42) имеет вид

таблица Гаусса—Жордана столбцовых множителей =

$$= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 1 \\ 2 & \frac{1}{2} & -1 \\ 1 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

В соответствии с замечаниями, высказанными в § 2.10, нижний треугольник этой таблицы идентичен нижнему треугольнику гауссова сомножителя L, в то время как верхний треугольник совпадает с нижним треугольником U^{-1}, элементы которого взяты с обратными знаками.

Выполнение исключения Гаусса—Жордана над матрицей A по строкам эквивалентно выполнению исключения Гаусса—Жордана над матрицей A^{T} (транспонированной к A) по столбцам. Получающаяся факторизация $A^{\mathrm{T}} = D_1'T_1'^CD_2'T_2'^CD_3'T_3'^C$ имеет вид

$$A^{\mathrm{T}} = \begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ 1 & 1 & \\ 1 & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & 1 \\ & & 1 \\ & -2 & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & & 4 \end{bmatrix}\begin{bmatrix} 1 & & \frac{1}{4} \\ & 1 & \frac{1}{4} \\ & & 1 \end{bmatrix},$$

откуда после транспонирования получаем соответствующую факторизацию матрицы $A = T_3'^RD_3'T_2'^RD_2'T_1'^RD_1'$:

$$A = \begin{bmatrix} 1 & & \\ & 1 & \\ \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & & 4 \end{bmatrix}\begin{bmatrix} 1 & & \\ 1 & 1 & -2 \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \\ & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix}$$

Таблица множителей в этом случае имеет вид:

таблица Гаусса—Жордана строчных множителей =

$$= \begin{bmatrix} \frac{1}{2} & 1 & 1 \\ 1 & \frac{1}{2} & -2 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

В заключение рассмотрим симметричную положительно-определенную матрицу

$$B = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 2 & 4 \end{bmatrix}.$$

В этом случае LU-факторизация имеет вид

$$B = \begin{bmatrix} 2 & & \\ 1 & \frac{5}{2} & \\ 1 & \frac{3}{2} & \frac{13}{5} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ & 1 & \frac{3}{5} \\ & & 1 \end{bmatrix},$$

в то время как $U^{\mathrm{T}}DU$-факторизация есть

$$B = \begin{bmatrix} 1 & & \\ \frac{1}{2} & 1 & \\ \frac{1}{2} & \frac{3}{5} & 1 \end{bmatrix} \begin{bmatrix} 2 & & \\ & \frac{5}{2} & \\ & & \frac{13}{5} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ & 1 & \frac{3}{5} \\ & & 1 \end{bmatrix}.$$

Разложение Холецкого $B = U'^{T}U'$, где U' — верхняя треугольная матрица, диагональные элементы которой могут быть отличны от единицы, можно получить либо извлечением квадратного корня из элементов матрицы D, либо путем выполнения трех шагов вида (2.51) (см. § 2.11), либо в соответствии с обычно используемой на практике реализацией этого алгоритма, которая обсуждалась в § 2.12. В любом случае результат выглядит следующим образом:

$$B = \begin{bmatrix} 2^{1/2} & & \\ \left(\frac{1}{2}\right)^{1/2} & \left(\frac{5}{2}\right)^{1/2} & \\ \left(\frac{1}{2}\right)^{1/2} & \left(\frac{9}{10}\right)^{1/2} & \left(\frac{13}{5}\right)^{1/2} \end{bmatrix} \begin{bmatrix} 2^{1/2} & \left(\frac{1}{2}\right)^{1/2} & \left(\frac{1}{2}\right)^{1/2} \\ & \left(\frac{5}{2}\right)^{1/2} & \left(\frac{9}{10}\right)^{1/2} \\ & & \left(\frac{13}{5}\right)^{1/2} \end{bmatrix}.$$

Матрица B имеет диагональное преобладание. В § 6.3 будут вычислены ее собственные значения.

Глава 3

Вычислительные ошибки в гауссовом исключении

3.1. ВВЕДЕНИЕ

Некоторые алгоритмы исключения, в частности гауссово исключение, были детально изучены в гл. 2. В случае когда операции, необходимые для решения системы

$$Ax = b, \tag{3.1}$$

выполняются в арифметике с плавающей запятой, результаты этих операций подвергаются воздействию округления или усечения. Возникающие ошибки возрастают в процессе выполнения алгоритма, и если для предотвращения чрезмерного их роста не предпринимать каких-либо мер, то они могут серьезно ухудшить точность окончательного решения или даже вызвать полную потерю верных значащих цифр. Рассмотрим следующий пример, где $-9 \leq a \leq 9$ и вычисления выполняются с использованием сумматора чисел с плавающей запятой, сохраняющего только одну цифру:

$$A = \begin{bmatrix} 0.1 & 1 \\ 1 & a \end{bmatrix}.$$

Если в качестве главного выбран элемент 0.1, то мы получаем

$$\mathrm{fl}\begin{bmatrix} 0.1 & 1 \\ 0 & a-10 \end{bmatrix} = \begin{bmatrix} 0.1 & 1 \\ 0 & -10 \end{bmatrix},$$

где символ fl означает «операция с плавающей запятой». Значение a как бы исчезает, и результат оказывается не зависящим от

него. Это произошло в результате плохого выбора главного элемента. Если же переставить строки матрицы A,

$$A' = \begin{bmatrix} 1 & a \\ 0.1 & 1 \end{bmatrix},$$

то результат окажется верным в пределах принятой точности:

$$fl\begin{bmatrix} 1 & a \\ 0 & 1 - 0.1a \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}.$$

В гл. 2 главные элементы выбирались последовательно вдоль диагонали матрицы, и вопрос о наличии какого-либо другого выбора, заслуживающего рассмотрения, не затрагивался. В действительности на k-м шаге исключения любой ненулевой элемент активной подматрицы $A^{(k)}$ может быть выбран в качестве главного. Напомним, что активная подматрица образована элементами матрицы $A^{(k)}$ с индексами $i, j \geq k$. Как показывает пример, мы можем либо выбирать в качестве главных те или иные элементы активной подматрицы, либо предварительно переупорядочить уравнения и неизвестные таким образом, чтобы главные элементы расположились последовательно вдоль диагонали. Эти процедуры эквивалентны математически, но лишь первая из них является приемлемой с вычислительной точки зрения, так как она не требует пересылок данных в памяти ЭВМ. В целях упрощения математического описания будем предполагать, что переупорядочение строк и столбцов матрицы A представляется при помощи матриц перестановок P и Q в виде произведения

$$A' = PAQ \tag{3.2}$$

и что над матрицей A' производится исключение с выбором главных элементов на диагонали; однако при реализации на ЭВМ фактическое переупорядочение не производится. Таким образом, вместо уравнения (3.1) на самом деле решается система

$$(PAQ)\,(Q^{\mathrm{T}}x) = (Pb), \quad . \tag{3.3}$$

где $QQ^{\mathrm{T}} = I$, так как Q является матрицей перестановки.

Свобода в выборе матриц P и Q может быть использована не только для обеспечения численной устойчивости и улучшения точности, но также и в целях сохранения разреженности треугольных сомножителей L и U. Ограничения, накладываемые требованиями устойчивости, рассматриваются в настоящей главе, а вопросы сохранения разреженности являются предметом гл. 4 и 5. Там же обсуждается проблема согласования указанных двух требований, являющаяся особенно трудной в случае, когда матрица A незнакоопределена.

В качестве общей характеристики «размера» вектора или матрицы обычно используются нормы. Для вещественных векторов x

с n компонентами будет использоваться следующая векторная норма:

$$p\text{-норма} \quad \|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \tag{3.4}$$

где $1 \leq p \leq \infty$.
В частности, для $p = 1$ или $p = \infty$:

$$1\text{-норма} \quad \|x\|_1 = \sum_{i=1}^{n} |x_i|, \tag{3.5}$$

$$\infty\text{-норма} \quad \|x\|_\infty = \max_{1 \leqslant i \leqslant n} |x_i|. \tag{3.6}$$

Для вещественной n x n-матрицы A будут использоваться следующие матричные нормы:

$$1\text{-норма} \quad \|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|, \tag{3.7}$$

$$\infty\text{-норма} \quad \|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|. \tag{3.8}$$

Для 1-нормы и ∞-нормы справедливы следующие неравенства:

$$\|x + y\| \leqslant \|x\| + \|y\|; \tag{3.9}$$

$$\|Ax\| \leqslant \|A\| \|x\|; \tag{3.10}$$

$$\|AB\| \leqslant \|A\| \|B\|. \tag{3.11}$$

Для любых двух векторов x и y выполняется неравенство Гёльдера

$$|x^T y| \leqslant \|x\|_p \|y\|_q, \tag{3.12}$$

где $\frac{1}{p} + \frac{1}{q} = 1$ и $1 \leq p, q \leq \infty$.

3.2. ОШИБКИ ОКРУГЛЕНИЯ В ОПЕРАЦИЯХ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Если число с плавающей запятой записывается в память ЭВМ, то сохраняется лишь некоторое фиксированное количество его старших значащих цифр (например, шесть шестнадцатеричных цифр, что эквивалентно 24 двоичным цифрам, в Фортране однократной точности). Когда над числами a и b, хранящимися в памяти ЭВМ, выполняется какая-либо операция, полученный результат перед записью в ячейку памяти необходимо предварительно подвергнуть округлению или усечению. Для указанного типа ошибок границы обычно устанавливаются следующим образом:

$$\mathrm{fl}\,(a \circ b) = (a \circ b)(1 + \varepsilon), \tag{3.13}$$

где символ $\circ$ обозначает одну из элементарных операций +, —, x или /, через $(a \circ b)$ обозначен точный результат операции, fl $(a \circ b)$ обозначает результат, полученный после выполнения операции с плавающей запятой и последующего усечения или округления (если это необходимо), и, кроме того, $|\varepsilon| \leq \varepsilon_M$, где ε_M — машинная точность; например, $\varepsilon_M = 2^{-t}$ для ЭВМ, использующей t-битовые числа и округляющей результат. Для описания точных вычислений будут использоваться обычные алгебраические обозначения, а для вычислений, использующих арифметику с плавающей запятой, — символ fl.

Хотя модель арифметики с плавающей запятой, представленная соотношением (3.13), является несколько упрощенной (см., напр., [Parlett, 1980, p. 23]), она широко использовалась [Wilkinson, 1965], и ее вполне достаточно для наших целей.

В следующем параграфе нам понадобится граница ошибки, возникающей при вычитании из числа a произведения двух других чисел l и u с использованием арифметики с плавающей запятой, в результате чего получается величина

$$b = \text{fl}\ (a - lu), \tag{3.14}$$

где a, l и u — точные значения, хранящиеся в памяти ЭВМ, а b — вычисленное значение. Ошибка e определяется соотношением

$$b = a - lu + e, \tag{3.15}$$

где предполагается, что операции над числами, хранящимися в памяти ЭВМ, выполняются точно. Предполагается, что величины a и b ограничены:

$$|a|, \quad |b| \leq \alpha_M. \tag{3.16}$$

Из соотношений (3.13) и (3.14) получаем

$$b = (a - lu(1 + \varepsilon_1))(1 + \varepsilon_2), \tag{3.17}$$

где

$$|\varepsilon_1|, \ |\varepsilon_2| \leq \varepsilon_M. \tag{3.18}$$

Так как наличие границ для l и u не предполагается, мы исключаем произведение lu из уравнения (3.15) при помощи соотношения (3.17), и выражаем e из полученного соотношения:

$$e = b\left(1 - \frac{1}{(1+\varepsilon_1)(1+\varepsilon_2)}\right) - a\frac{\varepsilon_1}{1+\varepsilon_1}. \tag{3.19}$$

Используя границы для a и b, заданные неравенством (3.16), получаем

$$|e| \leqslant \alpha_M \left(\left|1 - \frac{1}{(1+\varepsilon_1)(1+\varepsilon_2)}\right| + \left|\frac{\varepsilon_1}{1+\varepsilon_1}\right|\right). \tag{3.20}$$

Учитывая границы для ε_1 и ε_2, определяемые неравенствами (3.18), и предполагая, что $\varepsilon_M < 1$, мы получаем после некоторых преобразований оценку

$$|e| \leqslant \alpha_M \varepsilon_M \frac{1}{1-\varepsilon_M}\left(\frac{1}{1-\varepsilon_M} + 2\right). \tag{3.21}$$

Предположим теперь, что машинная точность не превышает $\varepsilon_M \leq 0.002$ (что эквивалентно 9 бит в представлении чисел). Это предположение обычно выполняется во всех стандартных ситуациях. В этом случае

$$\frac{1}{1-\varepsilon_M}\left(\frac{1}{1-\varepsilon_M} + 2\right) < 3.01,$$

и окончательная оценка для e имеет следующий вид:

$$|e| < 3.01\varepsilon_M\alpha_M. \tag{3.22}$$

Полезно отметить, что не удается существенно снизить погрешность, используя арифметику более высокой точности только при выполнении операций сложения, как это делается в некоторых программах, где скалярные произведения накапливаются в режиме двойной точности, в то время как умножения выполняются с однократной точностью. Если, например, взять $|\varepsilon_1| \leq \varepsilon_M$ и $|\varepsilon_2| \leq \varepsilon_M^2$ вместо (3.18), то получается оценка

$$|e| < 2.01\varepsilon_M\alpha_M. \tag{3.23}$$

Для повышения точности необходимо выполнять в двойной точности как сложение, так и умножение, а затем производить округление окончательного результата перед записью в память ЭВМ.

Результат, выражаемый неравенством (3.22), был получен в отсутствие каких-либо предположений об ограниченности величин l или u, хотя, несомненно, произведение lu ограничено величиной $2\alpha_M + |e|$ в силу соотношений (3.15) и (3.16). Методика, продемонстрированная выше, в сущности совпадает с той, которая была применена Ридом [Reid, 1971b]. В следующем параграфе при использовании оценки (3.22) для анализа k-го шага гауссова исключения по столбцам в качестве a и b будут фигурировать элементы редуцированных матриц $A^{(k)}$ и $A^{(k+1)}$ соответственно, в то время как l и u будут представлять собой элементы матриц L и U, a величина α_M будет являться верхней границей соответствующих элементов всех редуцированных матриц. Заметим теперь, что в силу (2.21) элементы матрицы L равны некоторым элементам матриц $A^{(k)}$, что в рассматриваемом случае означает справедливость оценки $|l| \leq \alpha_M$, и на самом деле величиной α_M ограничены все элементы матрицы L.

Однако этим свойством не обладают *u,* элементы матрицы *U,* полученные по формуле (2.18), которые могут быть велики в случае, когда мал главный элемент $a_{kk}^{(k)} = (d_k)_{kk} = l_{kk}$.

Существует другая формулировка [Bunch, 1974b], основанная на использовании границы σ для элементов *U* и границы $\alpha_м$ для элементов матриц *L* и $A^{(k)}$. Таким образом, в соотношении (3.15) $|a|, |b|, |l| \leq \alpha_м$ и $|u| \leq \sigma$. Исключая *a* из уравнений (3.15) и (3.17), получаем

$$e = b \frac{\varepsilon_2}{1+\varepsilon_2} - lue_1, \tag{3.24}$$

откуда, используя оценки (3.18) и $\varepsilon_м \leq 0.002$, легко получаем

$$|e| < (\sigma + 1.003)\varepsilon_м a_м. \tag{3.25}$$

Граница для величины $|e|$, определяемая неравенством (3.25), включает произведение $\sigma\alpha_м$, появление которого вызвано наличием произведения *1u* в уравнении (3.24). На практике обе величины σ и $\alpha_м$ могут быть велики (за исключением того случая, когда используется частичный выбор главного элемента по столбцам с выбором наибольшего элемента из *k*-й строки, что обеспечивает $\sigma = 1$, хотя граница $\alpha_м$ может стать слишком высокой). Фактически все стратегии, используемые для улучшения точности, пытаются устранить чрезмерный рост элементов матриц в процессе исключения. В этом контексте оценка, определяемая соотношением (3.22), менее ограничительна, и именно она будет использоваться в этой книге.

Уравнение (3.15), а также все остальные рассмотрения этого и следующего параграфов представляют собой разновидность обратного анализа ошибок Уилкинсона, широко используемого в литературе и заслужившего положительные отзывы многих авторов, см., например, [Parlett, 1980, p. 97].

3.3. ОШИБКИ ОКРУГЛЕНИЯ В РАЗРЕЖЕННОЙ ФАКТОРИЗАЦИИ

Рассмотрим элемент a_{ij} матрицы *A,* над которой производится шаг гауссова исключения по столбцам с выбором главных элементов непосредственно вдоль диагонали. Алгоритм исключения по столбцам взят в качестве примера; излагаемые ниже методы и результаты сохраняют силу также для исключения по строкам, и с небольшими изменениями — для разложения Холецкого. Подразумевается, что элемент $a_{ij}^{(k)}$ каждой из редуцированных матриц $A^{(k)}$, где $A^{(1)} \equiv A$, хранится в одной и той же ячейке памяти. На *k-м* шаге, где $k < \min(i, j)$, мы делим *k*-ю строку на главный элемент $a_{kk}^{(k)}$ и запоминаем величину $u_{kj} \equiv a_{kj}^{(k+1)} = a_{kj}^{(k)}/a_{kk}^{(k)}$ на месте, где был записан элемент $a_{kj}^{(k)}$. Операция,

выполняемая над элементом $a_{ij}^{(k)}$ на *k-м* шаге, при использовании арифметики с плавающей запятой записывается в следующем виде (см. рис. 3.1):

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}u_{kj} + e_{ij}^{(k)}, \tag{3.26}$$

где $l_{ik} = a_{ik}^{(k)}$ при $i \geq k$. Обозначения, использованные в формуле (3.26), те же, что и в формулах (2.18) и (2.21) гл. 2. Результаты, полученные в предыдущем параграфе, позволяют оценить ошибку $e_{ij}^{(k)}$, возникшую в вычислениях с плавающей запятой. Определим для этого величины $\alpha_{ij} — \max_k a_{ij}^{(k)}|$, так что

$$\left| a_{ij}^{(k)} \right| \leqslant \alpha_{ij} \tag{3.27}$$

при $k \leq \min(i, j)$. В частности, для $k = 1$ эти оценки справедливы для элементов исходной матрицы $A \equiv A^{(1)}$. Как уже отмечалось в предыдущем параграфе, оценка (3.27) определяет границы только для элементов матрицы L, но не для элементов U. Фактически, если $i \geq j = k$, то $l_{ik} = a_{ik}^{(k)}$ в силу (2.21) и значение $| l_{ik} |$ ограничено величиной α_{ik}. С другой стороны, если $k = i < j$, то $u_{kj} = a_{kj}^{(k)}/l_{kk}$ в силу (2.16) и (2.18), и поэтому произведение $|l_{kk}u_{kj}|$ ограничено величиной α_{kj}, однако для величины u_{kj}, которая может быть велика, границу получить не удается.

Если выполняется неравенство (3.27), то для уравнения (3.26) выполняются те же предположения, что и для уравнения (3.15), где $\alpha_м = \alpha_{ij}$ и мы можем получить оценку ошибки, используя (3.22):

$$\left| e_{ij}^{(k)} \right| \leqslant 3.01\varepsilon_M\alpha_{ij} \tag{3.28}$$

для $k < \min(i, j)$ при условии, что $\varepsilon_M < 0.002$.

Если некоторый элемент матрицы A равен нулю, и позиция, в которой он расположен, не подвергается заполнению в процессе исключения, то соответствующие элементы матрицы L или U, расположенные в той же позиции, будут в точности равны нулю. Более того, над этим элементом не производится никаких арифметических операций, и соответственно не возникают погрешности, определяемые уравнением (3.26). Мы будем отличать этот случай от другого случая, когда какой-либо элемент матриц L или U становится равным нулю в результате случайного взаимного унич-

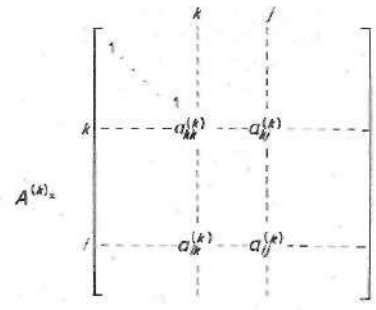


Рис. 3.1. Матрица $A^{(k)}$ непосредственно перед выполнением k-го шага гауссова исключения по столбцам.

тожения образующих его слагаемых; в этом случае погрешность присутствует, так как в соответствующей позиции происходило заполнение. Такой элемент матрицы будет рассматриваться как ненулевой. Случайное взаимное уничтожение происходит редко (за исключением некоторых специфических случаев), и вряд ли стоит пытаться явно учитывать его. Таким образом, ненулевым считается элемент, отличный от нуля либо равный нулю в результате взаимного уничтожения, в то время как нулевым считается тот элемент, который с самого начала был равен нулю и который не подвергался действию заполнения.

Заметим, что если хотя бы одно из значений l_{ik} или u_{kj} равно нулю, то уравнение (3.26) не определяет никакого изменения элемента $a_{ij}^{(k)}$, и, таким образом,

$$e_{ij}^{(k)} = 0. \tag{3.29}$$

Уравнение (3.26) выражает точное соотношение между вычисленными величинами; оно не отражает влияния общей ошибки, допущенной при вычислении $a_{ij}^{(k+1)}$. Действительно, для того чтобы оценить общую ошибку, необходимо было бы учесть ошибки, допущенные при вычислении величин $a_{ij}^{(k)}$, l_{ik} и u_{kj}. Ошибка $e_{ij}^{(k)}$, для которой выше были установлены границы, является не чем иным, как разностью между вычисленным значением $a_{ij}^{(k+1)}$ и тем значением $a_{ij}^{(k+1)}$, которое было бы получено, если бы над величинами $a_{ij}^{(k)}$, l_{ik} и u_{kj} выполнялись точные арифметические операции.

Важное свойство разреженного исключения описывается соотношением (3.29), которое указывает на то, что малое число операций, производимое над каждым элементом, приводит к не столь уж большому накоплению ошибки. Именно это свойство делает возможным решение очень больших разреженных систем прямыми методами без чрезмерного роста ошибок округления. Для того чтобы осуществить систематический анализ указанного свойства, определим матрицу с целочисленными элементами n_{ij}:

$$n_{ij} = \sum_{k=1}^{m} n_{ij}^{(k)}, \tag{3.30}$$

где

$n_{ij}^{(k)} = 1$, если и l_{ik}, и u_{kj} отличны от нуля;
$n_{ij}^{(k)} = 0$, в противном случае;
$m = \min(i, j)$.

Введем также следующие обозначения:

r_i^L — количество внедиагональных ненулевых элементов в i-й строке матрицы L;

c_i^L - количество в недиагональных ненулевых элементов в i-м столбце матрицы L;

r_i^U - количество внедиагональных ненулевых элементов в i-й строке матрицы U;

c_i^U - количество внедиагональных ненулевых элементов в i-м столбце матрицы U;

n_L - количество ненулевых элементов матрицы L, включая диагональ, $n_L = \sum_i r_i^L + n = \sum_i c_i^L + n$;

n_U - количество внедиагональных ненулевых элементов матрицы U, $n_U = \sum_i r_i = \sum_i c_i$.

Полезной оказывается следующая оценка:

$$n_{ij} \leqslant \min\left(r_i^L,\ c_j^U\right) + 1. \tag{3.31}$$

Так как $r_i^L \leqslant i - 1$ и $c_j^U \leqslant j - 1$, можно получить другую, более грубую оценку:

$$n_{ij} \leq \min \quad (i, \quad j), \tag{3.32}$$

которая может быть получена непосредственно из (3.30), если заметить, что $n_{ij}^{(k)} \leq 1$ и производится суммирование m таких членов. Формулы для n_{ij}, справедливые в случае полных и ленточных матриц, приведены в табл. 3.1, а числовые примеры для некоторых случаев показаны на рис. 3.2.

Продолжим теперь анализ уравнения (3.26). Каждый раз, когда на k-м шаге произведение $l_{ik}u_{kj}$ вычисляется и затем вычитается из элемента, расположенного в позиции (i, j), к этому элементу прибавляется ошибка $e_{ij}^{(k)}$. Если $j > i$, то (i, j)-й элемент изменяется последний раз на i — 1-м шаге, а на i-м шаге этот элемент делится на диагональный элемент l_{ii} и запоминается в качестве значения u_{ij}. Таким образом, операции, выполненные над элементом, для которого $j > i$, дают следующий результат:

$$\left(\left(a_{ij} - l_{i1}u_{1j} + e_{ij}^{(1)} - l_{i2}u_{2j} + e_{ij}^{(2)} - \ldots - l_{i,\,i-1}u_{i-1,\,j} + e_{ij}^{(i-1)}\right)/l_{ii}\right) \times \\ \times (1 + \varepsilon) = u_{ij}, \tag{3.33}$$

где множитель $(1 + \varepsilon)$ введен для того, чтобы учесть, согласно соотношению (3.13), ошибку, возникающую при выполнении деления на l_{ii}. Уравнение (3.33) можно переписать в виде

$$a_{ij} + \sum_{k=1}^{i-1} e_{ij}^{(k)} = \sum_{k=1}^{i} l_{ik}u_{kj} - l_{ii}u_{ij}\frac{\varepsilon}{1+\varepsilon}, \tag{3.34}$$

где в частном случае $i = 1$ суммирование в левой части опускается.

Таблица 3.1. Оценки норм матрица L, выражения для величин n_{ij} (см. (2.16)) и оценки норм матрицы ошибок E для факторизации $LU = A + E$, где все матрицы имеют порядок, равный n. Предполагается, что ширина ленты не превосходит n, если A — ленточная матрица

Матрица A	Границы норм L	Выражение для n_{ij}	Оценки ошибок факторизации
Разреженная	$\|L\| \leqslant \alpha_M \left(\max_j c_j^L + 1\right)$	$\sum_{k=1}^{m} n_{ij}^{(k)}$	$\|E\|_1 \leqslant 3.01\varepsilon_M\alpha_M \times \max_j \sum_{i=1}^{n} n_{ij}$
	$\|L\|_\infty \leqslant \alpha_M \times \left(\max_i r_i^L + 1\right)$	$m = \min(i, j)$	$\|E\|_\infty \leqslant 3.01\varepsilon_M\alpha_M \times \max_i \sum_{j=1}^{n} n_{ij}$
Заполненная	$\|L\|_1 \leqslant \alpha_M n$	$\min(i, j)$	$\|E\|_1, \|E\|_\infty \leqslant$
	$\|L\|_\infty \leqslant \alpha_M n$		$\leqslant \frac{3.01}{2}\varepsilon_M\alpha_M n(n+1)$
Ленточная $\beta\backslash\beta$	$\|L\|_1 \leqslant \alpha_M(\beta+1)$	$\max(0, \min(i, j,$	$\|E\|_1, \|E\|_\infty \leqslant$
	$\|L\|_\infty \leqslant \alpha_M(\beta+1)$	$i-j+\beta+1,$	$\leqslant 3.01\varepsilon_M\alpha_M(\beta+1)^2$
		$j-i+\beta+1))$	
Ленточная $\beta\backslash 2\beta$	$\|L\|_1 \leqslant \alpha_M(\beta+1)$	$\max(0, \min(i, j,$	$\|E\|_1, \|E\|_\infty \leqslant$
	$\|L\|_\infty \leqslant \alpha_M(\beta+1)$	$i-j+2\beta+1,$	$\leqslant 3.01\varepsilon_M\alpha_M \times$
		$j-i+\beta+1, \beta+1))$	$\times(\beta+1)(2\beta+1)$

Определим матрицу ошибок E с элементами e_{ij} следующим образом:

$$e_{ij} = \sum_{k=1}^{i-1} e_{ij}^{(k)} + l_{ii}u_{ij}\frac{\varepsilon}{1+\varepsilon}, \tag{3.35}$$

где $j > i$ (остальные элементы определены ниже соотношением (3.39)). Граница для элементов матрицы E может быть получена следующим образом. Используя неравенства $|\varepsilon| \leq |\varepsilon_M| \leq 0.002$ и тот факт, что $|l_{ii}u_{ij}| \leq \alpha_{ij}$ в силу (3.27), можно убедиться в справедливости неравенств

$$\left|l_{ii}u_{ij}\frac{\varepsilon}{1+\varepsilon}\right| < 1.003\varepsilon_M\alpha_{ij} < 3.01\varepsilon_M\alpha_{ij}. \tag{3.36}$$

Теперь если элемент u_{ij} является ненулевым, то число вычитаемых членов в уравнении (3.33) равно в точности $n_{ij} - 1$ (так как $l_{ii} \neq 0$ и поэтому в формуле (3.30) $n_{ij}^{(i)} = 1$). Сумма в формуле

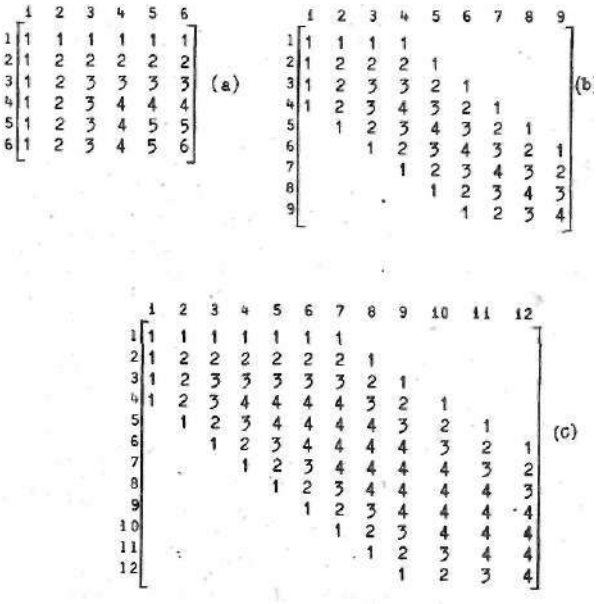


Рис. 3.2. Значения $n_{..}$, определенные соотношением (3.30): (a) для полной матрицы порядка $n = 6$; (*b*) для ленточной матрицы порядка $n = 9$ с полушириной ленты $\beta = 3$; (*c*) для ленточной матрицы порядка $n = 12$ с шириной нижней ленты $\beta = 3$ и шириной верхней ленты $2\beta = 6$.

(3.35) фактически содержит $n_{ij} - 1$ слагаемых, каждое из которых ограничено величиной $3.01\varepsilon_M a_{ij}$ в соответствии с оценкой (3.22). Если же элемент u_{ij} равен нулю, то заполнения не происходит и $n_{ij}^{(k)} = 0$ при $k < i$; кроме того, $n_{ij}^{(i)} = 0$ в силу $u_{ij} = 0$; таким образом, $n_{ij} = 0$. В любом случае при $j > i$ имеем оценку

$$| e_{ij} | \leq 3.01\varepsilon_M a_{ij} n_{ij}. \qquad (3.37)$$

Рассмотрение случая $j \leq i$ проводится аналогично. Операции, выполненные над элементом a_{ij}, приводят к следующему результату:

$$a_{ij} - l_{i1}u_{1j} + e_{ij}^{(1)} - l_{i2}u_{2j} + e_{ij}^{(2)} - \ldots - l_{i,\,j-1}u_{j-1,\,j} + e_{ij}^{(j-1)} = l_{ij}. \qquad (3.38)$$

Уравнение (3.38) выражает тот факт, что в качестве значения l_{ij} принимается результат выполнения последовательности операций типа (3.26). Для $j \leq i$ определим

$$e_{ij} = \sum_{k=1}^{j-1} e_{ij}^{(k)}. \qquad (3.39)$$

Если элемент l_{ij} ненулевой, то в силу того, что $u_{jj} = 1 \neq 0$ и $n_{ij}^{(j)} = 1$ в (3.30), сумма в (3.39) содержит $n_{ij} - 1$ членов. Если же элемент l_{ij} равен нулю, то в рассматриваемой позиции заполнения не происходит, так что $n_{ij} = 0$ и $e_{ij} = 0$. Таким образом, используя оценку (3.28), получаем, что для $j \leq i$ выполняется

$$|e_{ij}| \leqslant 3.01\varepsilon_M a_{ij} n_{ij}. \tag{3.40}$$

Наконец, подставляя (3.35) в (3.34) и (3.39) в (3 38), можно записать

$$LU = A + E, \tag{3.41}$$

где для всех i и j

$$|e_{ij}| \leqslant 3.01\varepsilon_M a_{ij} n_{ij}. \tag{3.42}$$

Несколько менее точная, но более удобная оценка получается, если использовать величину α_M, такую что

$$|a_{ij}^{(k)}| \leqslant \alpha_M \tag{3.43}$$

для всех i, j и $k \leq \min(i, j)$. В этом случае

$$|e_{ij}| \leqslant 3.01\varepsilon_M \alpha_M n_{ij}. \tag{3.44}$$

Глобальной оценкой «величины» матрицы ошибок E служит ее норма. Из (3.7) и (3.8) можно получить границы норм $\|E\|_1$ и $\|E\|_\infty$; для разреженных, полных и ленточных матриц формулы, задающие эти границы, приведены в табл. 3.1. Границы для норм матрицы L потребуются в § 3.4. Они могут быть легко получены из (3.43); соответствующие формулы для различных случаев, представляющих интерес, приведены также в табл. 3.1.

3.4. ОШИБКИ ОКРУГЛЕНИЯ В РАЗРЕЖЕННОЙ ПРЯМОЙ И ОБРАТНОЙ ПОДСТАНОВКЕ

Прямая и обратная подстановки рассматривались в § 2.13. Прямая подстановка, т. е. процесс решения системы $Lw = b$, может рассматриваться как алгоритм, выполняемый в n шагов, в результате чего последовательно вычисляются векторы $b^{(1)} \equiv b$, $b^{(2)}$, ... , $b^{(n)}$; при этом первые k компонент векторов $b^{(k+1)}$ и $b^{(k)}$ совпадают. На k-м шаге, $k = 1, 2, \ldots, n$, выполняются следующие вычисления:

$$\begin{aligned} w_k &= (b_k^{(k)}/l_{kk})(1 + \varepsilon), \\ b_i^{(k+1)} &= b_i^{(k)} - l_{ik} w_k + f_i^{(k)}; \quad i = k + 1, \ldots, n, \end{aligned} \tag{3.45}$$

где ε и $f_i^{(k)}$ — соответствующие вычислительные ошибки. Как обычно, предполагаем, что $|\varepsilon| \leq \varepsilon_M \leq 0.002$. Пусть $b_{Mi} = \max_k |b_i^{(k)}|$, так что

$$|b_i^{(k)}| \leqslant b_{Mi}; \quad 1 \leqslant i \leqslant n, \quad 1 \leqslant k \leqslant i. \tag{3.46}$$

Фактически оценка (3.46) справедлива для $1 \leq k \leq n$. Тогда из (3.22) получаем оценку

$$\left|f_i^{(k)}\right| \leqslant 3.01\varepsilon_M b_{Mi}; \quad k < i. \tag{3.47}$$

Никаких предположений относительно разреженности векторов b и w мы не делаем. Операции, выполненные над элементом b_i, приводят к соотношению

$$\left((b_i - l_{i1}w_1 + f_i^{(1)} - \ldots - l_{i,\,i-1}w_{i-1} + f_i^{(i-1)})/l_{ii}\right)(1+\varepsilon) = w_i, \tag{3.48}$$

которое можно переписать в виде

$$b_i + \sum_{k=1}^{i-1} f_i^{(k)} = \sum_{k=1}^{i} l_{ik}w_k - l_{ii}w_i \frac{\varepsilon}{1+\varepsilon}, \tag{3.49}$$

где при $i = 1$ суммирование в левой части должно быть опущено. Так как $|l_{ii}w_i| = |b_i^{(i)}| \leq b_{Mi}$[1]), получаем, что

$$\left|l_{ii}w_i \frac{\varepsilon}{1+\varepsilon}\right| \leqslant 1.003\varepsilon_M b_{Mi} \leqslant 3.01\varepsilon_M b_{Mi}. \tag{3.50}$$

Напомним, что через r_i^L обозначается количество внедиагональных ненулевых элементов в i-й строке матрицы L. Определим вектор ошибок δb с компонентами

$$\begin{aligned} \delta b_i &= \sum_{k=1}^{i-1} f_i^{(k)} + l_{ii}w_i \frac{\varepsilon}{1+\varepsilon}; \quad i > 1, \\ \delta b_1 &= l_{11}w_1 \frac{\varepsilon}{1+\varepsilon}. \end{aligned} \tag{3.51}$$

Тогда вычисленный результат w удовлетворяет точному соотношению

$$Lw = b + \delta b, \tag{3.52}$$

где, согласно (3.47) и (3.50), для компонент вектора δb справедлива следующая оценка:

$$|\delta b_i| \leqslant 3.01\varepsilon_M b_{Mi}(r_i^L + 1). \tag{3.53}$$

Менее точная, но более простая граница получается, если использовать b_M, величину наибольшего по модулю из элементов всех векторов $b^{(k)}$:

$$\left|b_i^{(k)}\right| \leqslant b_M; \; i = 1, 2, \ldots, n; \; k \leqslant i, \tag{3.54}$$

так что $b_{Mi} \leq b_M$. Отсюда получаем оценку

$$|\delta b_i| \leqslant 3.01\varepsilon_M b_M (r_i^L + 1). \tag{3.55}$$

Обратная подстановка заключается в решении системы $Ux = w$. Обратную подстановку можно рассматривать как алгоритм, вы-

[1]) Здесь автор пренебрегает множителем $(1 + \varepsilon)$ в (3.45). — *Прим. перев.*

полняемый за n шагов, в результате чего последовательно вычисляются векторы $w^{(n)} \equiv w$, $w^{(n-1)}$, ..., $w^{(2)}$, $w^{(1)}$; при этом, начиная с k-й и кончая n-й, компоненты векторов $w^{(k)}$ и $w^{(k-1)}$ совпадают. На k-м шаге, $k = n, n-1, \ldots, 1$, выполняются следующие вычисления:

$$x_k = w_k^{(k)},$$
$$w_i^{(k-1)} = w_i^{(k)} - u_{ik}x_k + g_i^{(k)};\ i = 1, \ldots, k-1, \qquad (3.56)$$

где $g_i^{(k)}$ — ошибка, возникшая в результате выполнения вычислений с плавающей запятой. Результат операций, выполненных над элементом w_i, $i < n$, можно записать в виде

$$w_i - u_{in}x_n + g_i^{(n)} - u_{i,\,n-1}x_{n-1} + g_i^{(n-1)} - \ldots - u_{i,\,i+1}x_{i+1} + g_i^{(i+1)} = x_i, \qquad (3.57)$$

или

$$w_i + \sum_{k=i+1}^{n} g_i^{(k)} = \sum_{k=i}^{n} u_{ik}x_k;\ i < n. \qquad (3.58)$$

Таким образом, если определить вектор ошибок δw:

$$\delta w_i = \sum_{k=i+1}^{n} g_i^{(k)};\ i < n,$$
$$\delta w_n = 0, \qquad (3.59)$$

то можно записать следующее точное соотношение между вычисленными величинами:

$$Ux = w + \delta w. \qquad (3.60)$$

Для того чтобы получить границы для δw, положим $w_{Mi} = \max_k |w_i^{(k)}|$, так что

$$|w_i^{(k)}| \leqslant w_{Mi};\ 1 \leqslant i \leqslant n;\ 1 \leqslant k \leqslant n. \qquad (3.61)$$

В частности, для $k = i$ получаем $w_i^{(i)} = x_i$, так что $|x_i| \leq w_{Mi}$. Пусть также w_M равно наибольшему из чисел w_{Mi}, т. е.

$$|w_i^{(k)}| \leqslant w_M;\ i = 1, 2, \ldots, n;\ k \geqslant i. \qquad (3.62)$$

Тогда, используя (3.22), получаем

$$|g_i^{(k)}| \leqslant 3.01\varepsilon_M w_{Mi};\ k > i, \qquad (3.63)$$

и

$$|\delta w_i| \leqslant 3.01\varepsilon_M w_{Mi} r_i^U, \qquad (3.64)$$

где r_i^U — количество внедиагональных ненулевых элементов в i-й строке матрицы U. Кроме того, используя (3.62), получаем

$$|\delta w_i| \leqslant 3.01\varepsilon_M w_M r_i^U. \qquad (3.65)$$

Наконец, рассмотрим невязку

$$r = Ax - b \tag{3.66}$$

для решения x системы (3.1), вычисленного с использованием арифметики с плавающей запятой. Из уравнений (3.41), (3.52) и (3.60) получаем

$$r = -Ex + L\delta w + \delta b. \tag{3.67}$$

Взяв 1-норму или ∞-норму, имеем оценку

$$\|r\| \leqslant \|E\|\|x\| + \|L\|\|\delta w\| + \|\delta b\|. \tag{3.68}$$

Из (3.62) получаем следующие границы для норм вектора x:

$$\begin{aligned} \|x\|_1 &\leqslant n w_M, \\ \|x\|_\infty &\leqslant w_M. \end{aligned} \tag{3.69}$$

Границы для норм матриц E и L приведены в табл. 3.1. Границы норм векторов ошибок δw и δb получаются из неравенств

Таблица 3.2. Значения некоторых параметров и оценки норм δb для δw для прямой и обратной подстановок

Матрица	Параметры, определенные в параграфе 3.4	Прямая подстановка	Обратная подстановка
Разреженная	См. параграф 3.4	$\|\delta b\|_1 \leqslant 3.01\varepsilon_M b_M n_L$ $\|\delta b\|_\infty \leqslant 3.01\varepsilon_M b_M \times$ $\times \left(\max_i r_i^L + 1\right)$	$\|\delta w\|_1 \leqslant 3.01\varepsilon_M w_M n_U'$ $\|\delta w\|_\infty \leqslant 3.01\varepsilon_M w_M \times$ $\times \max_i r_i^U$
Заполненная	$r_i^L = i - 1$ $r_i^U = n - i$ $n_L = n(n+1)/2$ $n_U' = n(n-1)/2$	$\|\delta b\|_1 \leqslant \frac{3.01}{2}\varepsilon_M \times$ $\times b_M n(n+1)$ $\|\delta b\|_\infty \leqslant 3.01\varepsilon_M b_M n$	$\|\delta w\|_1 \leqslant \frac{3.01}{2}\varepsilon_M \times$ $\times w_M n(n-1)$ $\|\delta w\|_\infty \leqslant 3.01\varepsilon_M \times$ $\times w_M (n-1)$
Ленточная $\beta\backslash\beta$	$r_i^L = \min(i-1, \beta)$ $r_i^U = \min(n-i, \beta)$ $n_L = (n - \beta/2)(\beta+1)$ $n_U' = (n - \beta/2 - 1/2)\beta$	$\|\delta b\|_1 \leqslant 3.01\varepsilon_M b_M \times$ $\times \left(n - \frac{\beta}{2}\right)(\beta+1)$ $\|\delta b\|_\infty \leqslant 3.01\varepsilon_M b_M \times$ $\times (\beta+1)$	$\|\delta w\|_1 \leqslant 3.01\varepsilon_M w_M \times$ $\times \left(n - \frac{\beta}{2} - \frac{1}{2}\right)\beta$ $\|\delta w\|_\infty \leqslant 3.01\varepsilon_M w_M \beta$
Ленточная $\beta\backslash 2\beta$	$r_i^L = \min(i-1, \beta)$ $r_i^U = \min(n-i, 2\beta)$ $n_L = (n - \beta/2)(\beta+1)$ $n_U' = (2n - 2\beta - 1)\beta$	$\|\delta b\|_1 \leqslant 3.01\varepsilon_M b_M \times$ $\times \left(n - \frac{\beta}{2}\right)(\beta+1)$ $\|\delta b\|_\infty \leqslant 3.01\varepsilon_M b_M \times$ $\times (\beta+1)$	$\|\delta w\|_1 \leqslant 3.01\varepsilon_M w_M \times$ $\times (2n - 2\beta - 1)\beta$ $\|\delta w\|_\infty \leqslant 6.02\varepsilon_M w_M \beta$

(3.65)' и (3.55J соответственно, и выражения для них можно найти в табл. 3.2. Таким образом, оценку для норма невязки $\| r \|$ можно найти из (3.68).

Можно дать другую интерпретацию вектора невязки *r*. Пусть $\bar{x}$ — точное решение системы (3.1); тогда $A\bar{x} = b$ и

$$r = A\ (x - \bar{x}). \tag{3.70}$$

Следовательно,

$$\| x - \bar{x} \| = \| A^{-1} r \| \leqslant \| A^{-1} \| \| r \|, \tag{3.71}$$

и если доступна оценка для нормы $\| A^{-1} \|$, то можно найти границу нормы погрешности решения $\| x - \bar{x} \|$

3.5. УПРАВЛЕНИЕ ОШИБКАМИ ОКРУГЛЕНИЯ

В уравнении (3.41) предполагается, что A — заданная матрица, точно записанная в памяти ЭВМ, а L и U — вычисленные треугольные матрицы. Уравнение (3.41) выражает тот факт, что точное произведение матриц L и U равно не A, а $A + E$, где E — матрица ошибок. Оценка (3.42) дает границы элементов матрицы E в терминах параметров ε_M, α_{ij} и n_{ij}. Воздействуя на эти параметры, мы можем улучшить точность решения. Напротив, неудачный выбор этих параметров может привести к неточному результату или даже к полной потере верных значащих цифр в решении, если некоторые элементы матрицы E окажутся много больше соответствующих элементов матрицы A.

Рассмотрим сначала параметр a_{ij},, определенный в (3.27). Чтобы значения α_{ij} оставались достаточно малыми, необходимо предотвратить чрезмерный рост элементов последовательных редуцированных матриц $A^{(k)}$. Это обычно достигается при помощи соответствующего *масштабирования* исходной матрицы A и последующего *выбора главных элементов* в процессе исключения. Масштабирование будет обсуждаться ниже в § 3.7. Если предположить, что все элементы матрицы A имеют сравнимую величину, то из уравнения (3.26) становится ясно, что рост элементов при увеличении k можно сдержать, если потребовать, чтобы последовательные главные элементы были достаточно велики. Для дальнейшего обсуждения этого вопроса необходимо более точно сформулировать понятия, относящиеся к выбору главного элемента. К началу k-го шага исключения имеется матрица $A^{(k)}$, и, для того чтобы вычислить следующую матрицу $A^{(k+1)}$, требуется зафиксировать главный элемент. В качестве главного элемента вовсе не обязательно выбирать элемент $a_{kk}^{(k)}$. Можно взять любой элемент $a_{ij}^{(k)}$, отличный от нуля, с индексами $i, j \geq k$, а затем поменять местами i-ю и k-ю строки и j-й и k-й столбцы с тем, чтобы переместить этот элемент в позицию (k, k) (см. рис. 3.1). Эти перестановки не вносят

никаких принципиальных сложностей в алгоритм исключения. Конечно, нет никакой необходимости фактически выполнять соответствующие пересылки в памяти ЭВМ: мы просто снабжаем строки и столбцы соответствующими векторами перестановок, в которых и записываются требуемые перемещения строк и столбцов. Формально, если (3.1) — заданная система линейных уравнений, то производится решение системы (3.3), где матрицы перестановок P и Q содержат всю необходимую информацию, относящуюся к перестановкам строк и столбцов.

Границы α_{ij} элементов редуцированных матриц $a_{ij}^{(k)}$ могут весьма существенным образом зависеть от выбора матриц P и Q. С другой стороны, известно, что структура разреженности матриц L и U, вычисляемых в процессе выполнения треугольной факторизации $PAQ = LU$, также очень сильно зависит от выбора этих матриц перестановок. В некоторых случаях матрицы P и Q можно определить до того, как начнется факторизация, в других случаях окончательный вид этих матриц может быть получен лишь в процессе исключения, но во всех случаях принимается во внимание как сохранение разреженности, так и устойчивость к воздействию ошибок округления.

Стратегии выбора главного элемента будут обсуждаться в следующем параграфе. Однако здесь заслуживает упоминания важный случай, когда матрица A — симметричная положительно определенная. Если a — максимальное значение диагональных элементов матрицы A и используется диагональный выбор главных элементов, то из оценки (2.59) следует, что любой элемент каждой из матриц $A^{(k)}$ не превосходит α по модулю. Таким образом, выбор главных элементов на диагонали — это все, что требуется для поддержания численной устойчивости в рассматриваемом случае; в то же время порядок, в котором производится выбор главных элементов, можно определять лишь на основе соображений, связанных с разреженностью. Этот простой подход положил начало важным исследованиям, результаты которых излагаются в гл. 4.

Продолжая обсуждение границ ошибок, определяемых оценкой (3.42), рассмотрим вопросы, касающиеся выбора машинной точности ε_M. Если в ЭВМ используется двоичная арифметика и вещественное число представляется при помощи t-битовой мантиссы и порядка, то $\varepsilon_M = 2^{-t}$. Общий объем памяти, требуемый для хранения множества вещественных чисел, примерно пропорционален значению t, и для ЭВМ, допускающих изменение значения t, время выполнения элементарной операции также можно принять пропорциональным t. Поэтому в худшем случае общая стоимость вычисления матриц L и U пропорциональна t^2. С другой стороны, вычислительная ошибка убывает экспоненциально

при увеличении t. Отсюда видно, что существенное повышение точности может быть достигнуто ценой сравнительно небольших дополнительных затрат, если использовать арифметику повышенной точности. Этот факт не совсем очевиден, но он редко упоминается в литературе по разреженным матрицам. Например, при увеличении t от 24 до 56 бит, что отвечает однократной и двойной точности на машинах фирмы IBM, затраты увеличатся приблизительно в четыре раза, в то время как граница ошибки уменьшится в $2^{32} \approx 4 \times 10^9$ раз. Конечно, если увеличение объема памяти повлечет за собой необходимость использования периферийных запоминающих устройств, увеличение затрат окажется более значительным.

В заключение обратим внимание на параметры n_{ij}, которые были определены в (3.30). Эти параметры также оказывают существенное влияние на границы ошибок, определяемые неравенствами (3.42). Значения n_{ij} ограничены величиной min (i, j), и указанная граница достигается для полных матриц; отсюда вытекает, что если матрица A имеет большой размер, а матрицы L и U не являются достаточно разреженными, то некоторые значения n_{ij} могут быть велики, и это приведет к ухудшению границ ошибок. Кроме того, заметим, что значение $n_{ij} - 1$ равно числу изменений элемента a_{ij} в процессе исключения; если допускается возможность роста элементов на каждом шаге исключения, то некоторые значения α_{ij} в оценках (3.42) могут стать чрезмерно большими, если только n_{ij} не являются достаточно малыми числами, для чего в свою очередь требуется, чтобы матрицы L и U были достаточно разреженными. Таким образом, всякий раз, когда мы пытаемся сохранять разреженность, мы, помимо других ее преимуществ, получаем понижение границ ошибок округления. Этот факт подчеркивался многими авторами [Tewarson, 1971; Duff, 1972, 1974a, 1977; Bunch, 1974b; Bunch, Rose, 1976; Gear, 1975]. Конечно, разреженность сама по себе не может гарантировать численную устойчивость, и выбор главных элементов все же должен подчиняться условиям, предотвращающим чрезмерный рост величин α_{ij}. В указанном отношении алгоритмы, сохраняющие разреженность, эффективны не только с точки зрения экономии памяти и вычислений, но и в смысле достижения существенного улучшения точности решения.

3.6. ЧИСЛЕННАЯ УСТОЙЧИВОСТЬ И ВЫБОР ГЛАВНЫХ ЭЛЕМЕНТОВ

Исследуем теперь наиболее распространенные стратегии выбора главного элемента с точки зрения численной устойчивости. Напомним, что активная подматрица на k-м шаге включает в себя все элементы $a_{ij}^{(k)}$, для которых $i \geq k$, $j \geq k$. Элементы активной

подматрицы будем называть *активными элементами.* Для изложения нам потребуются следующие четыре подмножества элементов активной подматрицы:

S_{pc} — подмножество первоначальных кандидатов, состоящее из тех ненулевых элементов активной подматрицы, среди которых осуществляется поиск. Это подмножество определяется с целью сокращения объема поиска при выборе главного элемента, что приводит к уменьшению трудоемкости выбора;

S_{st} — подмножество элементов из S_{pc}, удовлетворяющих некоторому условию устойчивости;

S_{sp} — подмножество элементов из S_{pc}, удовлетворяющих некоторому критерию сохранения разреженности;

$S_{piv} = S_{st} \cap S_{sp}$ — множество, из которого выбирается главный элемент. Элементы этого множества удовлетворяют как требованиям сохранения разреженности, так и условиям устойчивости.

Определения множеств S_{st} и S_{sp} не обязаны быть независимыми. Иногда S_st определяется как подмножество S_{sp}, или наоборот. Главный элемент выбирается из множества S_{piv}. Если это множество содержит более одного элемента, вновь принимаются во внимание соображения разреженности, и на их основе производится окончательный выбор. Таким образом, трудоемкость понижается настолько, насколько это возможно, за счет улучшения разреженности, в то время как условия устойчивости по-прежнему удовлетворяются. Если на некотором этапе оказывается, что множество S_{piv} пусто, то требования разреженности ослабляются, и множество S_{sp} переопределяется. Понятно, что выбранный критерий устойчивости не должен быть слишком жестким с тем, чтобы множество S_{st} было достаточно широким. На практике явное определение указанных четырех множеств обычно не производится, так как оказывается возможным использование упрощенных процедур.

В этом параграфе затрагиваются в основном вопросы, связанные с множеством S_{st}; множество S_{sp} обсуждается лишь кратко, однако детальное его рассмотрение можно найти в гл. 4 и 5.

Наиболее важный частный случай возникает в предположении, что матрица A — симметричная положительно определенная. В этом случае все диагональные элементы матрицы A положительны. Если α — значение наибольшего по модулю диагонального элемента матрицы A и все главные элементы выбираются на диагонали в произвольном порядке, то $\alpha_{ij} \leq \alpha$ для всех i, j в силу неравенств (2.59). Это означает, что элементы последователь-

ных матриц $A^{(k)}$ не превзойдут по модулю значения α, и, таким образом, условия устойчивости будут удовлетворены, если применять выбор главных элементов на диагонали. Таким образом, на *k-м.* шаге множество S_{st} состоит из элементов $a_{ii}^{(k)}$ при $i \geq k$, которые все положительны, так как активная часть $A^{(k)}$ также симметрична и положительно определена, и выбор главного элемента может быть осуществлен только из соображений сохранения разреженности. Эта стратегия, получившая название *диагонального выбора главного элемента,* является предметом гл. 4. Диагональный выбор главного элемента формально отвечает равенству $Q = P^T$ в (3.3), т. е. выполнению только симметричных перестановок строк и столбцов матрицы A. Тогда все активные подматрицы $A^{(k)}$ симметричны и $L — U^TD$, так что можно хранить только верхний (или нижний) треугольник матрицы A и диагональную часть матрицы D, и на их месте выполнять исключение, что позволит сократить примерно вдвое как объем памяти, так и вычислительные затраты.

Для незнакоопределенной матрицы выбор главного элемента нельзя ограничить только позициями на диагонали. При *полном выборе главного элемента* множество первоначальных кандидатов S_{pc} на *k-м* шаге состоит из всех ненулевых элементов активной подматрицы. Ненулевой элемент, имеющей наибольшую абсолютную величину, скажем $a_{ij}^{(k)}$, выбираемся в качестве k-го главного элемента и перемещается в позицию (k, k) путем перестановки i-й и k-й строк и j-го и k-го столбцов. Множество S_{st} состоит из единственного элемента $a_{ij}^{(k)}$, и множество S_{sp} обязательно должно содержать этот элемент. В случае полного выбора главного элемента рост элементов матриц ограничен следующим образом

$$|a_{ij}^{(k-1)}| < k^{1/2}\left(2^{1}3^{1/2}\ldots k^{1/(k-1)}\right)^{1/2} a_0 < 2k^{(\ln k+2)/4}\, a_0, \quad (3.72)$$

где $a_0 = \max |a_{ij}|$ и символ ln обозначает натуральный логарифм. Для справок полезны значения этих границ, приведенные в табл. 3.3.

Таблица 3.3

k	$k^{1/2}[2^1 \ldots k^{1/(k-1)}]^{1/2}$	$2k^{(\ln k+2)/4}$
100	3.57×10^3	4.01×10^3
1000	8.65×10^6	9.59×10^6
2000	1.51×10^8	1.68×10^8
5000	9.61×10^9	1.06×10^{10}
10 000	2.93×10^{11}	3.25×10^{11}

Истинная верхняя граница намного меньше. Указанная стратегия гарантирует тесные границы для величин α_{ij} в (3.27), но не является удовлетворительной с точки зрения сохранения разре-

женности, так как не дает никакой свободы для управления степенью заполнения. Полный выбор главного элемента рекомендуется применять лишь для плотных матриц небольшого размера, когда главной целью является достижение наилучшей численной устойчивости.

Частичный выбор главного элемента заключается в отыскании наибольшего по модулю элемента в каком-либо столбце (или строке) активной подматрицы с последующим выполнением соответствующих перестановок. Линия выбора главного элемента (под линией подразумевается строка или столбец) определяется из соображений сохранения разреженности. Таким образом., множество S_{pc} содержит все активные ненулевые элементы, так как для отыскания линии выбора главного элемента нужно произвести просмотр всей активной подматрицы. Множество S_{sp} содержит в точности все ненулевые элементы линии выбора главного элемента. Множество S_{st} в этом случае является подмножеством S_{sp}: оно содержит наибольшие по модулю элементы из S_{sp}. Множество $S_{piv.}$ совпадает с S_{st}. Из уравнения (3.26) (пренебрегая величиной $e_{ij}^{(k)}$) или из рис. 3.1 можно вывести следующую оценку, справедливую, если используется частичный выбор главного элемента:

$$\left|a_{ij}^{(k+1)}\right| \leqslant 2 \max_{i,j} \left|a_{ij}^{(k)}\right|. \tag{3.73}$$

Эта оценка гарантирует, что на каждом шаге элементы могут увеличиваться не более чем в два раза, и, таким образом, в итоге увеличение произойдет не более чем в 2^{ν} раз, где $\nu = \max (n_{ij}) - 1$ — максимальное число операций, выполненных над каждым отдельно взятым элементом [Gear, 1975]. Однако действительный рост элементов оказывается, как правило, гораздо более медленным, и если требуется получить более точные границы ошибок, то необходимо осуществлять контроль величины элементов матриц или использовать хороший способ оценки величины элементов. Эти вопросы обсуждаются в § 3.7.

Частичный выбор главного элемента позволяет использовать некоторые преимущества разреженного случая, однако возможности этой схемы все же слишком ограниченны. Без существенного ухудшения результатов удается получить более гибкую схему, если использовать стратегию *порогового выбора главного элемента* [Reid, 1971b, 1977] [1]). В этой стратегии все ненулевые элементы активной подматрицы рассматриваются как первоначальные кандидаты и включаются в множество S_{pc}. Выбирается *параметр допустимости и* [2]), лежащий в полуинтервале $0 < u \leq 1$, и S_{st}

[1]) В оригинале — threshold pivoting. — *Прим. перев.*
[2]) В оригинале — tolerance. — *Прим. перев.*

определяется как множество всех элементов $a_{ij}^{(k)}$ из S_{pc}, удовлетворяющих одному из следующих условий:

$$\left|a_{ij}^{(k)}\right| \geqslant u \max_{k<p<n} \left|a_{pj}^{(k)}\right|, \tag{3.74а}$$

$$\left|a_{ij}^{(k)}\right| \geqslant u \max_{k<q<n} \left|a_{iq}^{(k)}\right|. \tag{3.74б}$$

Множество S_{sp} определяется независимо, обычно на основе какого-либо варианта критерия Марковица, рассматриваемого в гл. 5. Окончательный выбор главного элемента производится из множества $S_{piv} = S_{st} \cap S_{sp}$, причем опять привлекаются соображения сохранения разреженности; затем выбранный элемент перемещается в позицию (k, k) при помощи перестановок. Если множество S_{piv} оказывается пустым, то S_{st} должно быть расширено посредством уменьшения значения u. На практике, однако, работу этой процедуры организуют другим способом. Сначала выбирается элемент из S_{pc}, наилучший с точки зрения сохранения разреженности. Затем проверяется, выполняются ли условия (3.74) для этого элемента. Если условия (3.74) удовлетворяются, то выбранный элемент принимается в качестве главного; в противном случае выбирается новый элемент исходя из соображений сохранения разреженности. Однако последняя ситуация встречается сравнительно редко и дополнительный поиск не вносит большого вклада в общие затраты при условии, что выбрано разумное значение u. Значение $u = 1$ отвечает стратегии частичного выбора главного элемента. Считается, что конкретный выбор u не является очень критичным, и обычно рекомендуется выбирать значение $u = 0.25$. Значение $u = 0.1$ позволяет получить хорошее сохранение как разреженности, так и устойчивости [Duff, 1977], однако малые значения, например $u = 0.01$, можно использовать в случае небольших значений величины $\nu = \max(n_{ij}) - 1$, встречающихся в задачах линейного программирования [Tomlin, 1972].

Пренебрегая величиной $e_{ij}^{(k)}$ в уравнении (3.26) и предполагая справедливость условий (3.74), получаем оценку

$$\left|a_{ij}^{(k+1)}\right| \leqslant \left(1 + u^{-1}\right) \max_{i,\,j} \left|a_{ij}^{(k)}\right|, \tag{3.75}$$

которая гарантирует, что рост элементов матриц на каждом шаге ограничен множителем $(1 + u^{-1})$, и общее увеличение происходит не более чем в $(1 + u^{-1})^{\nu}$ раз [Gear, 1975]. Действительный рост элементов обычно происходит намного медленнее, и должен применяться его контроль либо оценивание (см. § 3.7). Пороговый выбор главного элемента является весьма популярной стратегией и используется в большинстве стандартных программ.

Две другие стратегии выбора главного элемента были предложены Златевым [Zlatev, 1980]. Эти стратегии зависят от пара-

метра устойчивости и, выбираемого в пределах $0 < u \leq 1$, и от параметра p, определяющего количество строк активной подматрицы, просматриваемых на k-м шаге исключения; здесь $1 \leq p \leq n - k + 1$. Предполагается, что p и u — фиксированные числа [1]; при этом уже малые значения p, скажем $\rho = 3$, могут дать хорошие результаты. Множество S_{pc} первоначальных кандидатов состоит из ненулевых элементов p строк активной подматрицы. Выбор указанных p строк осуществляется из соображений улучшения разреженности, например, это могут быть p строк, имеющие наименьшее количество ненулевых элементов на k-м шаге исключения. Условия устойчивости, используемые для того, чтобы определить, какие элементы из S_{pc} принадлежат множеству S_{st}, те же, что и в стратегии порогового выбора главного элемента по строкам, и определяются неравенством (3.74б). Множество S_{sp} определяется как подмножество S_{st} на основе следующего критерия, обеспечивающего локальную оптимизацию разреженности: считается, что элемент $a_{ij}^{(k)}$ из S_{st} принадлежит S_{sp}, если $\mu_{ij} \equiv (r_i - 1)(c_j - 1) = \mu$, где $\mu = \min_{i,j}(\mu_{ij})$, r_i — число активных ненулевых элементов в i-й строке на k-м шаге исключения, а c_j — число активных ненулевых в j-м столбце на k-м шаге исключения. Другими словами, множество S_{sp} содержит все ненулевые элементы из S_{st}, для которых произведение числа всех остальных ненулевых элементов в той же строке на число всех остальных активных ненулевых элементов в том же столбце минимально (перед началом выполнения k-го шага). В этом случае множество S_{piv} совпадает с S_{sp}.

Стратегия Златева заключается в выборе в качестве главного любого элемента из S_{piv}, *Улучшенная стратегия Златева* использует в качестве главного элемент из S_{piv}, имеющий наибольшее абсолютное значение. Обычно обе стратегии дают примерно одинаковую точность, однако улучшенная стратегия позволяет получить лучшие результаты в некоторых трудных случаях, так как устраняются «плохие» последовательности главных элементов, допускаемые первой стратегией.

Если матрица A — симметричная, но незнакоопределенная и желательно сохранить симметрию, можно выбирать в качестве главного любой диагональный элемент, удовлетворяющий, например, условию устойчивости (3.74) для порогового выбора главного элемента, а также некоторому критерию сохранения разреженности. Однако такой элемент может не существовать, как показывает следующий пример [Wilkinson, 1965]:

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

[1]) Если $k \leq n$. то значение p уменьшается. — *Прим. перев.*

где исключение с выбором главного элемента на диагонали терпит неудачу на первом же шаге. Здесь заслуживают упоминания две устойчивые процедуры, сохраняющие симметрию. Эти методы основаны на конгруэнтных преобразованиях, а не на исключении. В случае плотной матрицы как число операций, так и объем памяти удается сократить примерно в два раза по сравнению с гауссовым исключением с перестановками, которые разрушают симметрию. Первый метод, который был предложен Парлеттом и Ридом [Parlett, Reid, 1970] и переформулирован Аасеном [Aasen, 1971], основан на приведении данной матрицы к симметричной ленточной форме с шириной ленты $2\beta + 1 = 3$ (т. е. к трехдиагональной форме). Полученная трехдиагональная система решается при помощи гауссова исключения с перестановками; при этом симметрия теряется. Другой метод — это метод *блочной диагонализации* [Bunch, 1971; Bunch, Parlett, 1971]; в нем главные элементы выбираются на главной диагонали, причем если не удается выбрать «хороший» скалярный главный элемент, то производится выбор главной подматрицы размера 2×2 в качестве главного элемента. В этом случае матрица приводится к блочно-диагональной форме. Эти два метода, а также еще один подход обсуждались в работе [Bunch, 1974a] как для плотного, так и для ленточного случаев, хотя ни один из методов не сохраняет ленточную структуру. Метод, предложенный Баичем, был опробован Даффом и Ридом [Duff, Reid, 1976] в разреженном случае, и были получены удовлетворительные результаты. Дальнейшее обсуждение можно найти у Даффа [Duff, 1978].

В приложениях встречаются ситуации, когда исключение производится одновременно со сборкой матрицы A. Это происходит в случае применения фронтальной или многофронтальной стратегий, используемых в основном в связи с методом конечных элементов или методом конечных разностей, а иногда и в более общем контексте. В этих случаях на каждом шаге множество S_{pc} ограничивается той частью матрицы A, которая уже полностью собрана. В симметричном положительно определенном случае главные элементы выбираются на диагонали. В симметричном незнакоопределенном случае можно использовать метод Банча, основанный на выборе в качестве главных элементов блоков размеров 1×1 либо 2×2, Дальнейшее обсуждение этой современной эффективной техники дается в гл. 4.

3.7. КОНТРОЛЬ И ОЦЕНКА РОСТА ЭЛЕМЕНТОВ

В случае когда применяется пороговый выбор главного элемента, отдельные элементы матрицы A могут, вообще говоря, увеличиться в $(1 + u^{-1})^{\nu}$ раз, где u — параметр допустимости, ν — число операций, выполненных над этим элементом. Однако на практике

действительный рост элементов обычно оказывается намного меньшим. Если мы интересуемся, насколько точным является полученное решение, то нам потребуются значения α_{ij} в (3.27), и поэтому мы должны либо контролировать рост элементов, либо использовать для него хорошую оценку. Контроль обычно осуществляется путем введения одного параметра α, такого что $|a_{ij}^{(k)}| \leq \alpha$ для всех i, j и k, значения которого могут быть легко определены в процессе исключения. В этом случае можно применять более слабую оценку ошибки (3.44). Однако такого рода контроль требует введения дополнительных команд во внутренний цикл программы, и, кроме того, иногда не позволяет получить истинного представления о некоторых ошибках, так как величины α_{ij} явно не вычисляются, в то время как верхняя граница α может оказаться чрезмерно завышенной. В таком случае выгодным может оказаться применение хорошей оценки. Опишем способ оценки, предложенный Эрисманом и Ридом [Erisman, Reid, 1974]. Используя неравенство (3.12) и тот факт, что $|\alpha + b| \leq |a| + |b|$ для любых a и b, из соотношения (2.23) получаем, что справедлива следующая оценка:

$$|a_{ij}^{(k)}| \leqslant |a_{ij}| + \|l_{i1}, l_{i2}, \ldots, l_{is}\|_p \|u_{1j}, u_{2j}, \ldots, u_{kj}\|_q, \quad (3.76)$$

где $k < i, j \leq n$. Эта оценка удобна для практического использования при $p = 1$ и $q = \infty$, или наоборот. В первом случае имеем

$$|a_{ij}^{(k)}| \leqslant |a_{ij}| + l_i^{(k)} u_j^{(k)}, \quad (3.77)$$

где при $k < i, j \leq n$

$$l_i^{(k)} = \sum_{m=1}^{k} |l_{im}|, \quad u_j^{(k)} = \max_{1 \leqslant m \leqslant k} |u_{mj}|. \quad (3.78)$$

Значения параметров $l_i^{(k)}$ и $u_j^{(k)}$ можно накапливать в двух вещественных массивах размера n каждый, и с их помощью проверять, не стал ли какой-либо элемент матрицы $A^{(k)}$ слишком велик. Если рост элементов становится неприемлемым, обычно оказывается необходимым увеличить значение параметра допустимости u и произвести исключение повторно, хотя можно также применить метод модификации главного элемента, принадлежащий Стюарту [Stewart, 1974]. Организованный указанным образом контроль роста элементов выполняется вне внутреннего цикла программы; более того, можно делать проверку лишь через каждые 10 или 50 шагов исключения, так как $l_i^{(k)}$ и $u_j^{(k)}$ монотонно возрастают с увеличением k. Конечно, если матрица A — симметричная и положительно определенная, то, как уже отмечалось выше, в случае диагонального выбора главного элемента никакого роста элементов не происходит и контроль становится ненужным.

3.8. МАСШТАБИРОВАНИЕ

На практике часто случается, что все элементы заданной матрицы A имеют сравнимую величину. В этой ситуации исключение может производиться непосредственно. В противном же случае для получения достаточно точной факторизации необходимо применить тот или иной способ *масштабирования.* Масштабирование обычно выполняется посредством умножения строк и столбцов матрицы A на подходящие масштабные множители. Формально, если система уравнений имеет вид $Ax = b$, то мы находим две диагональные матрицы R и C, а затем решаем систему

$$(RAC)\ (C^{-1}x) = (Rb). \tag{3.79}$$

Простейший метод получил название *уравновешивания* [Van der Sluis, 19701, Говорят, что матрица B *уравновешена по строкам* относительно p-нормы, если $\| e_i B \|_p = 1$ для всех i, где e_i — i-й столбец единичной матрицы I. Матрица B *уравновешена по столбцам,* если $\| Be_j \|_p = 1$ для всех j. Метод Ван дер Слюиса заключается в определении матрицы R таким образом, что RA уравновешена по строкам относительно некоторой нормы, с последующим выбором матрицы C из условия уравновешенности матрицы RAC по столбцам. Заметим, что полученная матрица RAC обычно не является уравновешенной по строкам. Однако эта простая процедура иногда может давать неудовлетворительные результаты [Curtis, Reid, 1972]. Вариант этого метода, называемый *балансировкой,* получается, если положить $C = R^{-1}$, а матрицу C определить таким образом, чтобы суммы модулей элементов одноименных строк и столбцов были примерно одинаковы [Parlett, Reinsch, 1969]. Матрица C определяется при помощи итерационной процедуры; с тем, чтобы предотвратить внесение ошибок округления в результате выполнения масштабирования, элементы этой матрицы выбираются в виде степеней основания системы счисления, принятой в данной ЭВМ. Балансировка используется главным образом при решении задач на собственные векторы и собственные значения в несимметричном случае, так как собственные значения не изменяются после выполнения этого преобразования [Smith et al., 1976, p. 200] [1)]. В результате балансировки улучшается число обусловленности матрицы [Wilkinson, 1965, p. 192].

Более эффективная процедура масштабирования была предложена Хэммингом [Hamming, 1971]. Диагональные элементы матриц R и C определяются следующим образом:

$$r_{ii} = \beta^{-r_i},\ c_{ii} = \beta^{-c_i}, \tag{3.80}$$

[1)] Балансировка, являясь преобразованием подобия, сохраняет собственные значения матрицы в любом случае; в несимметричном случае ее применение позволяет улучшить обусловленность матрицы, составленной из собственных векторов, — *Прим. перев.*

где β — основание позиционной системы счисления, принятой в данной ЭВМ ($\beta = 2$ для двоичных машин), так что элемент матрицы *RAC* имеет вид $a_{ij}\beta^{-r_i-c_j}$. Хэмминг предложил уменьшать разброс величин элементов матрицы *RAC* при помощи вычисления значений ρ_i и γ_j, минимизирующих величину

$$\sum_{a_{ij}\neq 0} (\log_\beta |a_{ij}| - \rho_i - \gamma_j)^2 \qquad (3.81)$$

с последующим выбором в качестве r_i и c_j ближайших целых к ρ_i и γ_j соответственно. Выбирая в качестве масштабных множителей целые степени основания системы счисления, можно быть уверенным, что в результате масштабирования не будут внесены ошибки округления. Однако в случае, когда значение β велико, такое масштабирование может оказаться слишком грубым, и для получения более качественного масштабирования, возможно, лучше будет взять значения $r_i = \rho_i$ и $c_j = \gamma_j$. Реализация этой процедуры не требует больших затрат. Кертис и Рид [Curtis, Reid, 1972] сообщают о хороших результатах; они используют для отыскания минимума функции (3.81) метод сопряженных градиентов, который обычно требует лишь от 7 до 10 обращений к каждому элементу матрицы. В работе [Tosovic, 1973] также сообщается о хороших результатах. Аналогичный метод был предложен Фалкерсоном и Вольфе [Fulkerson, Wolfe, 1962]. Они использовали методы линейного программирования для минимизации

$$\max_{a_{ij}\neq 0} |\log_\beta |a_{ij}| - \rho_i - \gamma_j|, \qquad (3.82)$$

однако Кертис и Рид [Curtis, Reid, 1972] нашли, что этот подход дает худшие результаты по сравнению с подходом, предложенным Хэммингом.

Скилом [Skeel, 1981] было отмечено, что в случае, когда частичный выбор главного элемента используется совместно с уравновешиванием и множества S_{sp} на каждом шаге исключения фиксированы, оказывается возможным навязать практически любой выбор главных элементов при помощи того или иного способа масштабирования матрицы. Скил показал, что выбор главного элемента по столбцу с уравновешиванием по строкам приводит к оценкам ошибок того же типа, что и выбор главного элемента по строке без уравновешивания, и наоборот. Им также было установлено, что в тех ситуациях, когда важна малость нормы невязки $\|r\|$, определенной в (3.66), погрешность в случае выбора главного элемента по столбцу без уравновешивания оказывается настолько малой, насколько это вообще можно предполагать, в то время как для выбора главного элемента по строке оказывается необходимым выполнять уравновешивание по столбцам.

Глава 4

Упорядочение для гауссова исключения: симметричные матрицы

4.1. ВВЕДЕНИЕ: ПОСТАНОВКА ЗАДАЧИ

В этой главе изучается вопрос об изменениях разреженности, происходящих в результате применения гауссова исключения к симметричной разреженной матрице A с целью получения треугольного разложения A в виде

$$A = U^T D U \tag{4.1}$$

или

$$A = U'^T U', \tag{4.2}$$

где U — верхняя треугольная матрица с единичной диагональю, D — диагональная, U' — верхняя треугольная, причем матрицы U и U' имеют одинаковую структуру разреженности (одинаковый портрет). Элементы матрицы U (или U') вычисляются в процессе исключения, и в тех позициях, где в матрице A располагались нулевые элементы, в этих матрицах могут возникнуть элементы, отличные от нуля. При вычислениях сумм могут происходить взаимные уничтожения, однако эта ситуация на практике встречается редко, за исключением некоторых случаев, заслуживаю-

щих отдельного рассмотрения. Следуя обычной практике, мы будем пренебрегать возможностью взаимных уничтожений. Таким образом, всякому элементу матрицы Л, отличному от нуля [1] соответствует расположенный в той же позиции ненулевой элемент матрицы U. Поэтому портрет матрицы U складывается из всех позиций, где находятся ненулевые элементы верхнего треугольника, и заполнения, т. е. позиций дополнительных ненулевых элементов, появляющихся во время исключения. Заметим, что мы будем иметь дело с позициями ненулевых элементов, но не с их значениями.

Существуют три серьезных причины, по которым заполнение следует считать нежелательным явлением:

(1) Необходимо отводить память для хранения возникших ненулевых элементов. При этом степень заполнения может оказаться весьма значительной. Наиболее крайний пример, часто цитируемый в литературе [см., например, Reid, 1977, p. 109, Duff, 1977, p. 504] — это матрица с заполненными первой строкой, первым столбцом и диагональю и остальными элементами, равными нулю: если выполнить исключение в первом столбце, то все остальные позиции матрицы "подвергнутся заполнению. Если матрица A симметрична и положительно определена, то размеры заполнения и позиции новых ненулевых элементов могут быть определены до того, как фактически начнется исключение. Таким образом, становится возможным заранее выделить необходимый дополнительный объем памяти. Однако если матрица A незнакоопределена, то это не удается сделать, и проблема распределения памяти существенно усложняется.

(2) Время, затрачиваемое при выполнении факторизации на ЭВМ, быстро увеличивается с ростом заполнения, так как приходится выполнять гораздо большее количество арифметических операций. Обычно объем памяти и число операций вместе определяют, поддается ли вообще данная задача решению прямыми методами. Круг задач, к которым применимы прямые методы, тесно связан, таким образом, с нашей способностью сводить заполнение к минимуму.

(3) Границы ошибок увеличиваются вместе с ростом заполнения, как это видно из оценки (3.42) предыдущей главы, где величины n_{ij}, определенные соотношением (3.30), могут рассматриваться в качестве меры разреженности матрицы U. Многие авторы отмечают ключевую роль того факта, что каждый элемент матрицы A лишь несколько раз претерпевает изменение в процессе исключения, и поэтому чрезмерного накопления вычислительных ошибок

[1]) И расположенному на диагонали или выше диагонали. — *Прим. перев.*

не происходит. С другой стороны, сдерживание роста ошибок должно рассматриваться как одна из целей, преследуемых при разработке алгоритмов, уменьшающих заполнение; этой точке зрения уделяется недостаточно внимания в литературе по разреженным матрицам. Чрезмерное заполнение может повлечь за собой слишком сильное накопление вычислительных ошибок, что вынудит применить итерационный метод вместо прямого при решении системы линейных алгебраических уравнений большого размера.

К настоящему времени известно достаточно много об источниках возникновения заполнения и для его уменьшения разработаны эффективные процедуры. Все они основаны на использовании свободы в выборе главных элементов, которые в случае положительно определенной матрицы A могут выбираться на диагонали в любом порядке (см. § 3.6). Можно использовать другую интерпретацию этого подхода, согласно которой строки и столбцы матрицы A переупорядочиваются любым способом, сохраняющим симметрию, а затем производится исключение с выбором главных элементов на диагонали в естественном порядке. Хорошо известен тот факт, что размеры заполнения чрезвычайно существенно зависят от того, какая именно выбрана перестановка. Так, если рассмотреть упомянутый выше «крайний» пример матрицы, все элементы которой равны нулю, кроме первой строки, первого столбца и диагонали, то при перестановке первого и последнего ее столбцов и первой и последней строк ненулевыми в полученной матрице будут последняя строка, последний столбец и диагональ, так что при исключении вообще не возникает ни одного нового ненулевого элемента.

Сформулируем теперь рассматриваемую задачу явно. Пусть требуется решить систему

$$Ax = b \tag{4.3}$$

с разреженной симметричной матрицей A. Если P — матрица перестановки, так что $P^TP = /$, то систему линейных уравнений (4.3) можно переписать в виде

$$PAP^{r}Px = Pb, \tag{4.4}$$

или, если обозначить $y = Px$ и $c = Pb,$ в виде

$$By = c, \tag{4.5}$$

где B — P^TAP — переупорядоченная форма матрицы $A,$ также являющаяся разреженной и симметричной. Кроме того, если матрица A положительно определена, то B также будет положительно определенной. С другой стороны, степень заполнения и число операций при факторизации матрицы B оказываются существенно зависящими от перестановки P. Наша задача заключается

в отыскании приемлемой матрицы P, или, что то же самое, приемлемого упорядочения строк и столбцов матрицы A, которое необходимо выполнить прежде, чем приступить к решению преобразованной системы $By = c$. При этом, вообще говоря, приемлемое упорядочение отличается от исходного упорядочения уравнений и неизвестных системы (4.3).

Говорят, что упорядочение является оптимальным в смысле заполнения, если оно приводит к наименьшему возможному заполнению. Упорядочение считается оптимальным в смысле затрат арифметических операций, если число операций, выполняемых при обработке переупорядоченной матрицы, минимально. Для матрицы A порядка n существует $n!$ различных упорядочений, из которых одно или более оказывается оптимальным в смысле заполнения, и несколько упорядочений (не менее одного) оказываются оптимальными в смысле минимизации числа арифметических операций. Можно было бы пожелать найти такие оптимальные упорядочения, однако, к сожалению, задача их отыскания представляется чрезвычайно трудной и о существовании эффективных алгоритмов, пригодных для этой цели, не известно ничего. Существующие процедуры являются эвристическими и обычно основываются на попытках отыскания упорядочений, обеспечивающих понижение заполнения и числа арифметических операций, но не гарантирующих достижения точного минимума этих величин.

Теория графов представляет собой чрезвычайно полезный инструмент для анализа алгоритмов обработки разреженных матриц. В § 4.2 даются некоторые основные понятия теории графов, однако если читатель проявит более глубокий интерес, то можно рекомендовать ему ознакомиться с монографиями [König, 1950, Harary, 1969]. В § 4.5 и 4.6 обсуждаются ленточные и профильные методы, основанные на глобальном рассмотрении процесса заполнения путем выделения некоторых областей матрицы, где допускается появление ненулевых элементов. В § 4.7 даются результаты из теории графов, необходимые для понимания материала остальных параграфов этой главы, которые посвящены обсуждению методов, более детально анализирующих заполнение. Сравнение методов является трудной проблемой, так как эффективность метода зависит от конкретной задачи или от класса задач, для решения которых он используется. Некоторые ссылки на литературу по этому вопросу можно найти в соответствующих параграфах.

Основная часть материала данной главы посвящена положительно определенным матрицам, однако в параграфе 4.17 изучаются также симметричные незнакоопределенные матрицы.

Методы, описанные в данной главе, реализованы программами (см., например, каталог программ [Heath, 1982] и работу [Duff, 1978]).

4.2. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ ГРАФОВ

Граф $G = (V, E)$ состоит из множества V *вершин* $u, v, w, \ldots$ и множества E *ребер,* где ребром называется пара (u, v) вершин из V. Для обозначения числа элементов множества, или его *мощности,* будем использовать две вертикальные черты; так, $|V|$ — это число вершин, а $|E|$ — число ребер графа G. Если между парами (u, v) и (v, u) не делается различия, то говорят, что ребра представляются неупорядоченными парами вершин и граф является *неориентированным.* Если же пары вершин, представляющие собой ребра, считать упорядоченными, то граф называют *ориентированным,* или, сокращенно, *орграфом.* Неориентированный граф можно рассматривать и как орграф, для которого если пара (u, v) образует ребро, то (v, u) также является ребром. Граф с вершинами называют *помеченным* (или *упорядоченным,* или *пронумерованным),* если установлено взаимно однозначное соответствие между его вершинами и целыми числами $1, 2, \ldots, n$ [1]. В помеченном графе мы часто будем ссылаться на вершины по их номерам, и если отображение α задает соответствие между вершинами графа и натуральными числами, то иногда для помеченного графа будет использоваться обозначение $G_a = (V, E, \alpha)$.

Любой матрице A можно поставить в соответствие граф. Так, если A — квадратная $n \times n$-матрица с элементами a_{ij}, и все ее диагональные элементы отличны от нуля [2], то соответствующий граф будет неориентированным, помеченными вершинами его будут $v_1, v_2, \ldots, v_n$, и пара (v_i, v_j) будет являться ребром в том и только в том случае, когда $a_{ij} \neq 0$. Диагональный элемент a_{ii}, отличный от нуля, соответствует в этом случае *петле* (v_i, v_i), и предположение о том, что $a_{ii} \neq 0$ для всех i, отвечает тому, что граф содержит все петли. Если это действительно так, то обычно нет необходимости явно учитывать наличие петель в графе. В гл. 5 будут обсуждаться орграфы и другие типы графов, которые обычно ставятся в соответствие несимметричным матрицам. В настоящей главе рассматриваются только симметричные матрицы с ненулевыми элементами на диагонали, и поэтому анализируются только неориентированные графы. На рис. 4.1 показан пример симметричной матрицы и соответствующего ей помеченного графа. Вопросы хранения информации, задающей то или иное представление графа, в памяти ЭВМ были изучены в § 1.4.

Симметричные перестановки строк и столбцов симметричной матрицы не изменяют ее графа, оказывая влияние лишь на нумерацию его вершин. Указанное свойство инвариантности обусловливает важность аппарата теории графов для технологии разреженных матриц. Более определенно, если A — симметричная

---

[1]) Подразумевается, что $n = |V|$. — *Прим. перев.*

[2]) Неявно предполагается, что A симметрична. — *Прим. перев.*

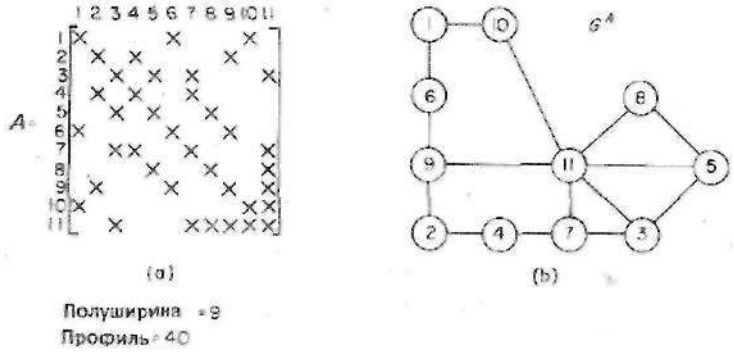


Рис. 4.1. Структура (портрет) симметричной разреженной матрицы и соответствующий помеченный неориентированный граф.

матрица и $B = PAP^T$, где P — матрица перестановки (см. § 2.6), а G_A и G_B — графы, связанные с матрицами A и B соответственно, то G_A и G_B идентичны с точностью до нумерации вершин.

Если вершинам поставлены в соответствие точки, а ребрам — простые дуги на плоскости, то получается *плоская укладка* графа. Изображение графа на бумаге, подобное тому, какое показано на рис. 4.1 (*b*), представляет собой пример плоской укладки графа. Граф называется *плоским,* если для него существует плоская укладка, никакие две дуги которой не пересекаются. Класс матриц с плоским графом замечателен тем, что для них существуют «хорошие» с точки зрения гауссова исключения упорядочения. Граф, изображенный на рис. 4.1 (*b*), является плоским.

Подграф $G' = (V', E')$ графа $G = (V, E)$ — это граф, образованный некоторыми (может быть, всеми) вершинами графа G и некоторыми ребрами графа G: $V' \subseteq V$, $E' \subset E$. Подграф называется *частичным графом,* если V' состоит из некоторых вершин G, а E' включает в себя ребра (u, v) графа G, такие что $u \in V'$ и $v \in V'$:

$$V' \subset V,$$
$$E' = \{(u, v) \in E \mid u \in V', v \in V'\}, \qquad (4.6)$$

где фигурные скобки { } используются для обозначения множества. На рис. 4.1 (*b*) вершины 3, 5, 7, 8 и 11 вместе с ребрами (3, 5), (5, 11), (5, 8), (8, 11), (11, 7), (3, 7) и (3, 11) образуют частичный граф.

Если (u, v) — ребро, то вершины u и v называют *смежными*. Говорят, что ребро (u, v) *инцидентно* вершинам u и v. *Степенью* вершины называют число инцидентных ей ребер. Если W — подмножество вершин графа G, то его *смежным множеством* Adj(W) называют множество всех вершин, не принадлежащих W и смеж-

нех с какой-либо вершиной из W. Именно если $G=(V, E)$ и $W \subset V$, то

$$\mathrm{Adj}(W) = \{u \in V \setminus W \mid \exists v \in W \ni (u, v) \in E\},$$

где через $V \setminus W$ обозначается множество всех вершин из V, не принадлежащих W. На рис. 4.1 вершины 1 и 6 являются смежными и обе эти вершины имеют степень 2. Если множество W образовано вершинами 1 и 6, то $\mathrm{Adj}(W) = \{9, 10\}$. Подграф, в котором любые две вершины смежные, называют *кликой*. На рис. 4.1 подграф с вершинами 3, 7, 11 есть клика.

Путем называется упорядоченное множество попарно различных вершин $(u_1, u_2, \ldots, u_{m+1})$, таких что для $i = 1, 2, \ldots, m$ вершины u_i и u_{i+1} — смежные [1]. Число m называют *длиной* пути. Путь длины m можно также рассматривать как упорядоченное множество m ребер $(u_1, u_2), (u_2, u_3), \ldots, (u_m, u_{m+1})$. Говорят, что вершины u и v *соединены путем,* если существует путь, для которого вершины u и v являются концевыми. Путь называется *циклом,* если $u_1 = u_{m+1}$. *Расстояние* $d(u, v)$ между двумя вершинами u и v определяется как длина наикратчайшего пути, соединяющего эти вершины. Для данной вершины u наибольшее расстояние между u и любой другой вершиной графа называется *эксцентриситетом* $e(u)$ вершины u. Наибольшее значение эксцентриситета по всем вершинам графа называется *диаметром* графа. *Периферийной* вершиной называется вершина, эксцентриситет которой равен диаметру графа. Многие алгоритмы для разреженных матриц, основанные на обработке графа матрицы, требуют задания «стартовой» вершины с большим значением эксцентриситета. Выбор периферийной вершины был бы идеальным с этой точки зрения; однако, к сожалению, достаточно эффективные алгоритмы для отыскания периферийных вершин неизвестны. С другой стороны, существуют хорошие алгоритмы, позволяющие находить *псевдопериферийные* вершины [Gibbs, Poole, Stockmeyer, 1976]. Псевдопериферийная вершина u определяется следующим условием: если v — любая вершина, для которой $d(u, v) = e(u)$, то $e(v) = e(u)$. Указанное определение гарантирует, что эксцентриситет псевдопериферийной вершины будет «большим», и действительно, обычно он оказывается близким к диаметру графа. На рис. 4.1 вершины 1 и 3 соединены путем (1, 10, 11, 3) длины 3, а также путем (1, 6, 9, 11, 3) длины 4. Таким образом, расстояние между вершинами 1 и 3 равно 3. Путь (5, 8, 11, 3, 5) является циклом. Диаметр графа равен 4, и периферийными будут вершины 1, 2, 4, 5 и 6, так как их эксцентриситет равен 4. Для этого примера все псевдопериферийные вершины совпадают с перифе-

[1]) Вершины u_1, u_{m+1} называют концевыми, и они могут совпадать (если путь является циклом). — *Прим. перев.*

рийными; такое совпадение, по-видимому, часто встречается на практике.

Граф называется *связным,* если для каждой пары его вершин найдется путь, который их связывает. В противном случае граф называют *несвязным.* Всякий несвязный граф состоит из двух или более *связных компонент. Разделителем* называется множество вершин, удаление которых вместе с инцидентными им ребрами приводит к появлению несвязного графа (или к увеличению числа связных компонент, если граф уже был несвязным). Разделитель называют *минимальным,* если никакое его собственное подмножество не является разделителем. Разделитель, состоящий из одной вершины, называется *разрезающей вершиной.* Граф, изображенный на рис. 4.1, — связный. Множество, образованное вершинами 7 и 11, представляет собой минимальный разделитель, и удаление этих вершин разбивает граф на две связные компоненты с вершинами 3, 5, 8 и 10, 1, 6, 9, 2, 4.

Для данного подмножества вершин $W \subset V$ *оболочкой* Span (W) множества вершин W называют объединение этого множества и всех вершин, связанных с вершинами, принадлежащими W. Если W состоит из единственной вершины v, то Span (v) совпадает со связной компонентой графа, содержащей эту вершину, а в общем случае Span (W) состоит из всех связных компонент графа, содержащих какую-либо вершину из W.

Понятие достижимого множества оказалось весьма полезным при изучении гауссова исключения [George, Liu, 1978a]. Если для данного графа G и подмножества его вершин S выбраны две не совпадающие вершины u и v, не принадлежащие S, то будем говорить, что вершина v *достижима из вершины* u *через* S в случае, когда либо u и v соединены путем длины единица (т. е. вершины u и v — смежные), либо u и v соединены путем, все вершины которого, кроме начальной и конечной, принадлежат S. Для заданного S и вершины $u \notin S$ *достижимое множество* Reach (u, S) из u через S определяется как множество всех вершин, достижимых из u через S. Заметим, что если S пусто или если вершина u не принадлежит Adj (S), то в этом случае Reach (u, S) = Adj (u). Если на рис. 4.1 выбрать S = {7, 3}, то Reach (5, S) = {8, 11, 4} и Reach (8, S) = Adj (8) = {5, 11}. Отметим также, что сама вершина u не принадлежит множеству Reach (u, S).

Существует ряд способов разбиения графов. *Разбиение* получается, если сгруппировать вершины графа в попарно непересекающиеся подмножества S_0, S_1, … , S_m. Если граф — несвязный и в качестве подмножеств выбраны его связные компоненты, то получается *разбиение на компоненты.* Если множество вершин разбито на уровни, то получается *структура уровней;* этот важный класс разбиений, обсуждаемый в § 4.3, используется при построении многих алгоритмов обработки разреженных матриц.

Неориентированный граф, обладающий свойством связности и не имеющий циклов, называется *деревом.* В рассматриваемом контексте важность деревьев заключается в том, что матрица, граф которой является деревом, может быть переупорядочена таким образом, что при выполнении гауссова исключения заполнение будет отсутствовать [Parter, 1961]. В дереве для любых двух вершин существует в точности один путь, соединяющий их [Berge, 1962]. Дерево называется *корневым,* если указано, что одна из его вершин является корнем. Единственность пути, соединяющего корень с любой другой вершиной *и* дерева, используется для того, чтобы установить отношение предшествования на множестве вершин дерева: если вершина о принадлежит такому пути, то *v* называется *предком и,* а вершина *и* называется *потомком v.* Если вершины *и* и *v* смежны, то *v* — *отец* вершины *и, а и* — *сын* вершины *v.* Если *и* и *v* — предки вершины *w*, то говорят, что *и* — *старший* (*младший*) предок *w* по сравнению с *v,* если расстояние *d* (*r*, *и)* между корнем *r* и вершиной *и* меньше (больше), чем расстояние *d* (*r, v*). Самым старшим предком любой вершины корневого дерева является корень *r*. Множество предков вершины называют ее *родословной,* а множество всех ее потомков называют *потомством* этой вершины.

Как и всякий граф, дерево может быть помечено путем нумерации вершин. Существуют различные схемы нумерации, обладающие теми или иными полезными свойствами [Aho, Hopcroft, Ullman, 1976]. Для наших целей важным является *монотонное упорядочение,* при котором каждая вершина имеет номер, меньший, чем ее отец.

Разбиение графа, не являющегося деревом, можно использовать для построения нового графа, называемого *факторграфом,* который уже представляет собой дерево. На основе этой идеи строятся методы, использующие «хорошие» свойства деревьев даже в случае матриц, графы которых отличны от дерева. Пусть S_1, S_2, … , S_m — разбиение множества V вершин G на подмножества. Факторграф имеет m множеств S_i в качестве *составных вершин,* а ребра этого графа определяются так: (S_i, S_j) есть ребро факторграфа в том и только том случае, когда существуют две вершины $u \in S_i$, u $v \in S_j$, такие что в графе G эти вершины смежные. Если факторграф представляет собой дерево, то его называют *фактордеревом,* а соответствующее разбиение — *древовидным разбиением.* На рис. 4.2 показаны граф, его структура уровней и его фактордерево. Цифрами около кружков указано монотонное упорядочение фактордерева. Алгоритм для отыскания древовидного разбиения графа общего вида принадлежит Джорджу [George, 1977] и описан ниже в § 4.9.

Для заданного связного графа $G = (V, E)$ *остовным деревом* $T = (V, E')$ называется подграф, представляющий собой дерево

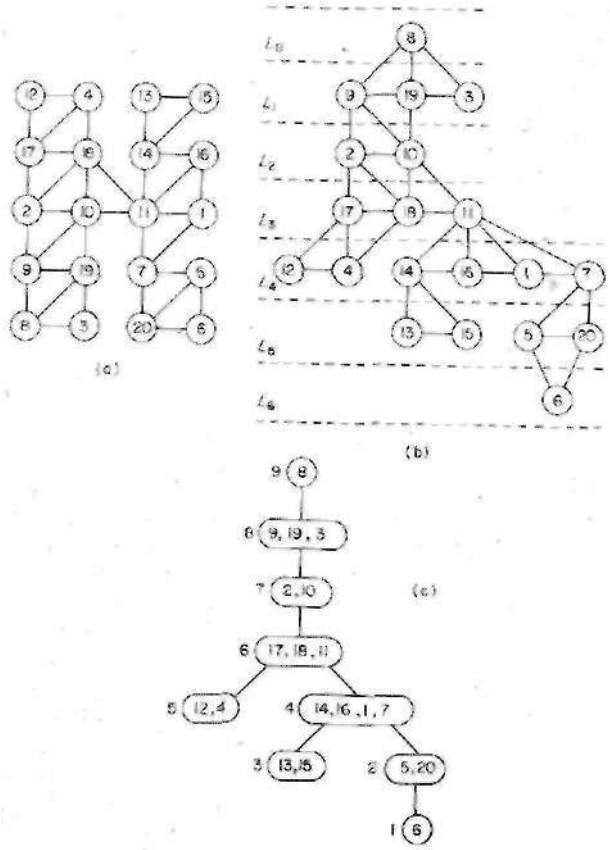


Рис. 4.2. Неориентированный граф (a), его структура уровней (b) и соответствующее фактор-дерево (c). На рис. (c) показано также монотонное упорядочение фактор-дерева.

и содержащий все вершины графа G. Остовное дерево можно получить при помощи просмотра графа и обрывания циклов путем удаления ребер при условии, что граф остается связным. Удобной процедурой, пригодной для этой цели, является поиск в глубину [Tarjan, 1972], который обсуждается в § 4.14. Остовное дерево графа, изображенного на рис. 4.1, показано на рис. 4.3. Совокупность деревьев с попарно различными множествами вершин, совместно дающими все V, называется *остовным лесом.* Остовный лес получается при разрывании циклов в несвязном графе.

Важным источником задач с разреженными матрицами являет-

ся метод конечных элементов [Zienkiewicz, 1977], и некоторые идеи, обсуждающиеся в данной главе, первоначально возникли именно в связи с этим методом. В простейшей версии метода решение дифференциального уравнения в области, принадлежащей плоскости или пространству, осуществляется путем разбиения области на конечные элементы с узлами в вершинах и, возможно, с дополнительными узлами на границах конечных элементов и внутри их. Затем формируется разреженная матрица A, в которой каждая строка и столбец отвечают некоторому узлу получившейся сетки, причем $a_{ij} \neq 0$ в том и только том случае, если узлы i и j принадлежат одному и тому же конечному элементу. Рассмотрим теперь граф, отвечающий матрице A, называемый *конечноэлементным графом.* Существует взаимно однозначное соответствие между вершинами этого графа и узлами сетки. Две вершины графа соединены ребром тогда и только тогда, когда соответствующие узлы сетки принадлежат одному и тому же конечному элементу; таким образом, каждому конечному элементу в графе соответствует клика. Сетка сама по себе может рассматриваться как представление другого графа, называемого *скелетом,* конечноэлементного графа. В этом случае конечноэлементный граф можно получить путем добавления к скелету всевозможных ребер, соединяющих узлы, принадлежащие одному и тому же конечному элементу.

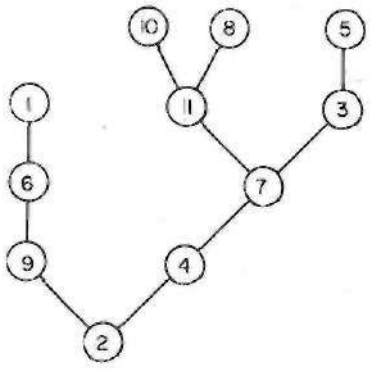


Рис. 4.3. Остовное дерево для графа, показанного на рис. 4.1 (b).

Если методом конечных элементов решается двумерная задача, то получается двумерная сетка и скелетный граф является плоским. Если применяются треугольные элементы с тремя узлами, то конечноэлементный граф совпадает со своим скелетом, и, следовательно, также представляет собой плоский граф. Однако любой другой тип элементов, даже простые четырехугольники с четырьмя узлами, порождает граф, не являющийся плоским. Графы не являющиеся плоскими, но имеющие плоский скелет, называются *почти плоскими графами.* Указанное определение легко обобщается на случай трех пространственных измерений. Почти плоские и почти трехмерные графы используются в § 4.12 в связи с алгоритмом обобщенных вложенных сечений.

В следующих параграфах мы обсудим некоторые теоретико-графовые алгоритмы, представляющие интерес для технологии разреженных матриц.

4.3. ПОИСК В ШИРИНУ И СТРУКТУРЫ УРОВНЕЙ СМЕЖНОСТИ

В предыдущем параграфе упоминалось, что разбиение графа G = (V, *E)* можно осуществить путем группировки вершин в несколько попарно непересекающихся подмножеств. *Структуры уровней смежности,* или просто *структуры уровней,* представляют собой весьма важный класс разбиений. Структура уровней L_0, $L_1,\ldots,L_m$, состоящая из $m + 1$ уровней, получается, если указанные подмножества определены следующим образом:

$$\begin{aligned} &\mathrm{Adj}\,(L_i) \subseteq L_{i-1} \cup L_{i+1},\ 0 < i < m,\\ &\mathrm{Adj}\,(L_0) \subseteq L_1,\\ &\mathrm{Adj}\,(L_m) \subseteq L_{m-1}. \end{aligned} \tag{4.7}$$

Число m называется *длиной* структуры уровней, а *ширина* структуры уровней определяется как максимальное количество вершин, образующих каждый уровень. В структуре уровней каждый уровень L_i, $0 < i < m$, является разделителем графа. Говорят, что структура уровней имеет *корень* в L_a; если $L_0 \subset V$ и каждое следующее множество L_i смежно с объединением предыдущих множеств:

$$L_i = \mathrm{Adj}\left(\bigcup_{j=0}^{i-1} L_j\right), \quad i > 0. \tag{4.8}$$

Если L_0 состоит из единственной вершины u, т. е. $L_0 = |u|$, то будем говорить, что структура уровней имеет *корень в вершине u.*

Поиском называется процедура, при помощи которой совершается обход вершин и ребер графа $G = (V, E)$ в некоторой последовательности. Последовательность, в которой обходятся вершины, может быть использована для упорядочения графа, а свойства ребер, выявляемые в процессе поиска, можно использовать для разбиения ребер на классы. Многие теоретико-графовые алгоритмы используют такой подход, и алгоритмы, применяемые в технологии разреженных матриц, не являются в этом смысле исключением. Примерами могут быть алгоритм Гиббса для отыскания псевдопериферийных вершин или алгоритмы сечений Джорджа для симметричных матриц. Поиск начинается с некоторой произвольно выбранной вершины *s.* Затем производится обход остальных вершин в последовательности, устанавливаемой по некоторому правилу. Говорят, что осуществляется *поиск в ширину,* если перед тем, как перейти к следующей вершине, просматриваются все ребра, инцидентные текущей вершине. О *поиске в глубину* говорят, если рассматривается только одно из ребер и в качестве новой вершины берется та вершина, которая инцидентна этому ребру. Поиск в глубину обсуждается в § 4.14. В данном параграфе изучается использование поиска в ширину для построе-

ния корневой структуры уровней. Во время поиска в ширину происходит сортировка вершин по уровням, а ребра помечаются как *древесные* либо *поперечные.*

Уровнем L_0 служит множество, состоящее из единственной стартовой вершины s. На каждой отдельной стадии поиска нам известны все вершины некоторого уровня L_i. Все вершины предыдущих уровней уже были посещены, и вершины уровня L_i тоже помечены как посещенные. Затем для вершины $v \in L_i$ мы перебираем ребра, ведущие из вершины v в другие вершины. Если оказалось, что ребро ведет в непомеченную вершину да, то считается, что эта вершина принадлежит уровню L_{i+1}, а ребро помечается как древесное. Если ребро ведет в уже помеченную вершину x, то ребро (v, x) помечается как поперечное. Эта уже помеченная вершина χ может принадлежать только L_i либо L_{i+1}, так что поперечные ребра могут соединять лишь вершину одного и того же уровня или вершины смежных уровней. Древесные ребра соединяют только вершины смежных уровней, и при $|j - k| > 1$ не может быть связей между вершинами уровней L_j и L_k. Таким образом, каждый уровень L_i, $1 < i < m$, представляет собой разделитель графа. Более того, в процессе поиска получается остовное дерево, образованное множеством V вершин исходного графа и древесными ребрами. Алгоритм, разработанный авторами статьи [Rose, Tarjan, Lueker, 1976], использует очереди (см. § 1.2) для хранения вершин в том порядке, в каком они посещаются. Пусть для графа $G = (V, E)$ через V_v обозначено множество уже посещенных вершин, а через E_t — множество ребер, уже помеченных как древесные. Алгоритм заключается в следующем:

Шаг 1 (Инициализация). Формируем очередь, содержащую стартовую вершину s. Полагаем level (s) = 0.

Шаг 2 (Формирование элемента разбиения). Если очередь пуста, то выполнение алгоритма прекращается. В противном случае исключить v из конца очереди и определить множество S еще не посещавшихся вершин, смежных с v:

$$S \leftarrow \text{Adj}\ (v) \cap (V \setminus V_v).$$

Шаг 3 (Сортировка вершин и ребер). Если $S = \emptyset$, то перейти к шагу 2. В противном случае для каждой вершины $w \in S$ выполнять следующие операции:

(3 а) поставить да в начало очереди;

(3 б) положить level (w) = level (v) + 1;

(3 в) пометить ребро (v, w) как древесное:

$$E_t \leftarrow E_t \cup (v, w);$$

(3 г) пометить вершину w как посещенную:

$$V_v \leftarrow V_v \cup \{w\}.$$

Шаг 4 (Завершение цикла). Перейти к шагу 2.

После того, как выполнение алгоритма завершилось, все вершины v, для которых level (v) = i, образуют уровень L_i, а все ребра из L_t — поперечные. В качестве примера применения алгоритма рассмотрим граф, изображенный на рис. 4.1 (b), и выберем вершину 10 в качестве стартовой. Вначале вершина 10 является единственной вершиной в очереди и level (10) = 0. Затем вершина 10 удаляется из очереди и определяется ее смежное множество S = {1, 11}, причем обе эти вершины не посещались. Поэтому полагаем level (1) = 1, level (11) = 1 и помечаем ребра (10, 1) и (10, 11) как древесные. Вершины 1 и 11 ставятся в начало очереди.

Теперь, так как вершина 10 удалена из очереди, мы обнаруживаем в ее конце вершину 1. Удаляем вершину 1 и определяем ее смежное множество, содержащее лишь одну непосещавшуюся вершину b. Поэтому получаем level (b) = 2 и помечаем ребро (1, 6) как древесное. Затем из конца очереди выбирается вершина 11. Множество непосещавшихся вершин, смежных с ней, есть S = {9,7,3,8}. Эти вершины заносятся во второй уровень, и ребра, соединяющие их с вершиной 11, помечаются как древесные. Читателю предоставляется закончить построение структуры уровней самостоятельно; окончательный ее вид показан на рис. 4.4. Остовное дерево определяется на этом рисунке своими ребрами, выделенными сплошными линиями. Пунктирные линии соответствуют поперечным ребрам. Общее число древесных ребер равно $|V| - 1 = 10$, а общее число поперечных ребер составляет $|E| - |V| + 1 = 4$. Длина структуры уровней равна $m = 3$,

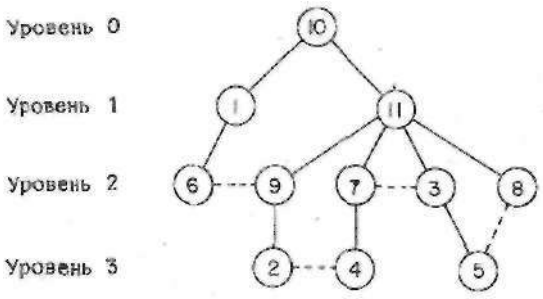


Рис. 4.4. Структура уровней для графа, изображенного на рис. 4.1, (b), полученная при помощи поиска в ширину. Древесные ребра остовного дерева показаны сплошными линиями. Поперечные ребра показаны пунктирными линиями.

а ширина равна 5. Заметим, что множество вершин {1, 11} первого уровня является разделителем графа. Множество {6, 9, 7, 3, 8}, образованное вершинами второго уровня, также представляет собой разделитель.

4.4. ОТЫСКАНИЕ ПСЕВДОПЕРИФЕРИЙНОЙ ВЕРШИНЫ И УЗКОЙ СТРУКТУРЫ УРОВНЕЙ ГРАФА

Алгоритм, описанный в этом параграфе, разработан авторами статьи [Gibbs, Poole, Stockmeyer, 1976]. Хотя первоначально алгоритм предназначался для уменьшения ширины ленты и профиля разреженной матрицы, в настоящее время этот алгоритм широко используется для определения вершин графа, имеющих большой эксцентриситет. Во многих важных алгоритмах для разреженных матриц предусматривается задание вершины с большим значением эксцентриситета (имеется в виду вершина, принадлежащая графу, связанному с разреженной матрицей). Алгоритм Гиббса использует в качестве стартовой вершину г, имеющую наименьшую степень (и поэтому, вероятно, обладающую большим эксцентриситетом) и строит структуру уровней с корнем в *r*. Если *v* — вершина последнего уровня, *то* длина структуры уровней равна расстоянию между *r* и *v,* которое в свою очередь совпадает с эксцентриситетом вершины *r*. Однако эксцентриситет вершины *v* по меньшей мере равен длине структуры уровней и может превосходить ее; в последнем случае эксцентриситет вершины *v* будет больше, чем эксцентриситет вершины *r.* Поэтому алгоритм строит структуры уровней с корнями в вершинах последнего уровня предыдущей структуры в соответствии с возрастанием степеней стартовых вершин, пока не получится структура уровней с большей длиной. Указанная процедура повторяется до тех пор, пока длины всех возможных структур не окажутся совпадающими; в этом случае вершина *r* является псевдопериферийной. При практической реализации на ЭВМ время вычислений можно сократить за счет отказа от завершения построения структур, обладающих слишком большой шириной, как только появление таких структур будет установлено [Crane, Gibbs, Poole, Stockmeyer, 1975]. Другая экономичная модификация была предложена Джорджем [George, 1977]. Структура уровней с корнем в *r* может содержать в последнем уровне более одной вершины; Джордж предложил выбирать в последнем уровне из каждой его связной компоненты по одной вершине, имеющей минимальную степень. Модифицированный алгоритм будет обязательно находить вершину с большим эксцентриситетом, хотя и не всегда строго удовлетворяющую определению псевдопериферийной вершины. Модифицированный алгоритм заключается в выполнении следующих шагов.

Шаг 1. Найти вершину r с минимальной степенью.
Шаг 2. Построить структуру уровней с корнем в вершине r.
Шаг 3. Найти все связные компоненты частичного графа, отвечающего последнему уровню построенной структуры уровней.
Шаг 4. Для каждой связной компоненты определяем вершину, имеющую наименьшую степень, и строим соответствующую этой вершине структуру уровней. Если возникают широкие структуры уровней, то отказываемся от их рассмотрения. Если для некоторой вершины v оказывается, что соответствующая ей структура обладает большей длиной, чем структура с корнем в r, то полагаем $r \leftarrow v$ и переходим к шагу 3.
Шаг 5. Вершина r является искомой вершиной с достаточно большим значением эксцентриситета.

Для графа, изображенного на рис. 4.1, одна из возможных стартовых вершин с наименьшей степенью, равной 2, — это вершина 10. Структура уровней с корнем в вершине 10 имеет длину, равную 3 (см. пример, приведенный в § 4.3). Вершины 2, 4 и 5 образуют последний уровень структуры. Из первой связной компоненты с вершинами 2, 4 выбираем вершину 2 и находим, что структура уровней с корнем в вершине 2 имеет длину 4. Таким образом, вершина 2 «лучше» вершины 10. На последнем уровне структуры с корнем в вершине 10 мы находим единственную вершину 5. Структура уровней с корнем в вершине 5 также имеет длину 4, и поэтому вершина 2 является искомой. В данном случае удалось найти истинную периферийную вершину (диаметр графа равен 4), и этого же часто удается достичь на практике.

4.5. УМЕНЬШЕНИЕ ШИРИНЫ ЛЕНТЫ СИММЕТРИЧНОЙ МАТРИЦЫ

Как уже упоминалось в § 1.5, ширина ленты симметричной матрицы зависит от упорядочения ее строк и столбцов. В данном параграфе обсуждается *алгоритм Катхилл—Макки* [Cuthill, McKee, 1969, Cuthill, 1972], который представляет собой систематическую процедуру, предназначенную для перенумерации строк и столбцов таким образом, чтобы в результате соответствующей перестановки ширина ленты матрицы уменьшилась.

Алгоритм производит обработку неориентированного графа, связанного с матрицей, и требует задания стартовой вершины. Определяется нумерация графа, а затем строки и столбцы переставляются в соответствии с полученной нумерацией, причем симметрия матрицы сохраняется. Нумерация вершин производится следующим образом. Стартовой вершине присваивается номер 1. Тогда для $i = 1, 2, \ldots, n$, где n — порядок матрицы, все вершины,

смежные с i-й вершиной и еще не пронумерованные, нумеруются последовательно в порядке возрастания степеней.

В качестве стартовой можно выбрать одну из вершин, имеющую минимальную степень. В работах [George, Liu, 1975, George, 1977] предложена лучшая стратегия, предусматривающая использование псевдопериферийной вершины, которая в свою очередь находится при помощи алгоритма Гиббса, описанного в § 4.4. Другие стратегии выбора стартовой вершины, а также стратегии, предусматривающие использование более чем одной стартовой вершины, были предложены в работах [Cheng, 1973a, Cheng, 1973b], однако в работе [George, 1977] установлено, что они обладают меньшей эффективностью. Для многих вычислительных целей комбинация алгоритма Катхилл—Макки с выбором псевдопериферийной вершины в качестве стартовой рассматривается как стандартная процедура благодаря своей простоте и эффективности [1]).

Предлагались также другие алгоритмы, предназначенные для уменьшения или минимизации ширины ленты. Метод, описанный в работе [Alway, Martin, 1965], заключается в систематическом поиске перестановок, уменьшающих ширину ленты, производимом до тех пор, пока не будет достигнут теоретически возможный минимум. Розеном [Rosen, 1968] был предложен метод, основанный на непосредственных перестановках пар строк и столбцов. Этот метод работает быстрее, чем предыдущий, однако в случае, когда задано недостаточно хорошее начальное приближение к искомой перестановке, он может дать в результате ширину ленты, заметно превосходящую минимальное значение [Cuthill, McKee, 1969]. Оба алгоритма работают лучше, если применяются для модификации перестановок, полученных при помощи алгоритма Катхилл—Макки. Алгоритм, предложенный в работе [Arany, Smyth, Szoda, 1971], является методом максимальной степени, аналогичным методу Катхилл—Макки с тем отличием, что в нем производится специальный выбор вершин высокой степени для середины структуры уровней. Для некоторых матриц этот метод может привести к хорошим результатам при сравнительно небольших вычислительных затратах, особенно в случае, когда граф матрицы содержит разрезающую вершину, которая выбирается в качестве стартовой. Алгоритмы, позволяющие находить разрезающие вершины, описаны в работах [Paton, 1971, Shirey, 1969, Martelli, 1973].

Тьюарсоном [Tewarson, 1971] был предложен вероятностный метод для переупорядочения произвольной разреженной матрицы с целью сужения ленты или с целью приведения к какой-либо другой желаемой форме.

---

[1]) Эта процедура в виде пакета подпрограмм содержится в книге Джорджа и Лю. — *Прим. перее.*

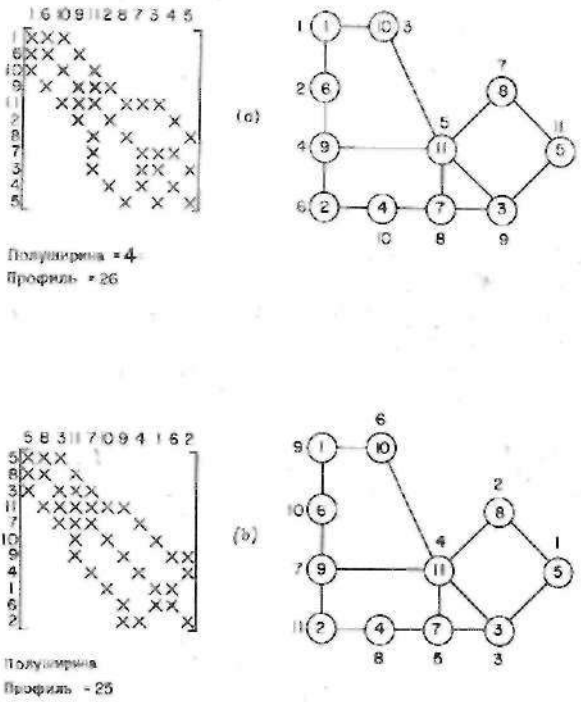


Рис. 4.5. Два примера применения алгоритма Катхилл — Макки к графу, изображенному на рис. 4.1 (b). Показаны переупорядоченная матрица и новое помечивание графа, если (a) в качестве стартовой выбрана вершина 1 и (b) в качестве стартовой выбрана вершина 5. Новое помечивание графа показано цифрами рядом с кружками, а исходное помечивание — цифрами внутри кружков.

На рис. 4.5 приведен пример применения алгоритма Катхилл—Макки. Рассмотрим симметричную разреженную матрицу и отвечающий ей неориентированный граф, показанные на рис. 4.1. Порядок матрицы равен 11, и она имеет сравнительно большую ширину ленты, равную 9. Граф этой матрицы рассматривался в 4.2, где указывалось, что периферийными являются вершины 1, 2, 4, 5 и 6. Выберем в качестве стартовой вершину 1. Новый номер этой вершины также будет равен 1, как показано на рис. 4.5, где исходная нумерация вершин показана внутри кружков, а новая нумерация — рядом с ними. На первом шаге множество уже перенумерованных вершин состоит из единственной вершины 1. Смежными будут вершины 6 и 10, и каждая из них имеет степень, равную 2. Выбор можно сделать любым образом, и мы пометим вершину 6 как вторую, а вершину 10 — как третью. Следующими кандидатами являются вершины 9 и 11, имеющие степень

3 и 5 соответственно; поэтому вершину 9 помечаем как четвертую, а вершину 11 — как пятую. Выполнение алгоритма продолжается аналогичным образом до тех пор, пока не будет полностью найдена новая нумерация. Получающаяся перенумерация и соответствующая переупорядоченная матрица показаны на рис. 4.5 (а). Полуширина ленты сокращается до значения, равного 4. Важно, однако, отметить, что фактическое переупорядочение элементов матрицы в памяти ЭВМ, как правило, не производится. Вместо этого оказывается достаточным запомнить полученную перестановку в массиве, скажем с именем IPER. Тогда, если требуется обратиться к *i*-й строке или *i*-му столбцу матрицы, то на самом деле производится обращение к строке или столбцу с номером IPER (*i*) [Gustavson, 1976а]. Для нашего примера:

Позиция = 1 2 3 4 5 6 7 8 9 10 11
IPER = 1 6 10 9 11 2 8 7 3 4 5.

Если требуется, например, вызвать четвертую строку матрицы, изображенной на рис. 4.5 (а), то мы находим IPER (4) = 9 и берем 9-ю строку исходной матрицы, показанной на рис. 4.1.

Другой пример применения алгоритма Катхилл—Макки показан на рис. 4.5 (*b*). На этот раз в качестве стартовой выбирается периферийная вершина с номером 5. Получаемая нумерация графа и соответствующая переупорядоченная матрица показаны на рисунке. Переупорядоченная матрица и в этом случае имеет полуширину ленты, равную 4.

4.6. УМЕНЬШЕНИЕ ПРОФИЛЯ СИММЕТРИЧНОЙ МАТРИЦЫ

В § 1.6 профиль симметричной матрицы был определен как число элементов в оболочке матрицы. Там же упоминалось, что уменьшение профиля представляется весьма желательным, если матрица хранится в соответствии со схемой Дженнингса, и что оно может быть достигнуто путем переупорядочения строк и столбцов. В данном параграфе будут обсуждаться некоторые алгоритмы, которые пригодны для этой цели. Рассматриваемые алгоритмы производят операции над графом, связанным с разреженной матрицей.

Было установлено [George, 19711, что путем упорядочения, полученного при помощи алгоритма Катхилл—Макки (обсуждавшегося в предыдущем параграфе), часто можно найти упорядочение, существенно уменьшающее профиль матрицы. Такой метод получил название *обратного алгоритма Катхилл—Макки.* При обращении порядка строк и столбцов шириналенты остается неизменной, однако профиль не увеличивается; этот факт был доказан в работе [Liu, Sherman, 1975]. Примеры применения этого метода

Рис. 4.6. Примеры применения обратного алгоритма Катхилл — Макки к матрицам, изображенным на рис. 4.5 (a) и 4.5 (b) соответственно.

приведены на рис. 4.6, где матрицы (a) и (b) представляют собой переупорядоченные в обратном порядке матрицы (a) и (b) на рис. 4.5 соответственно. Профиль уменьшается с 26 до 24 в первом случае и с 25 до 23 — во втором. Профиль исходной матрицы, показанной на рис. 4.1, равен 40.

Другим важным средством уменьшения профиля является алгоритм Кинга [King, 1970], который работает следующим образом. Выбираем вершину, имеющую минимальную степень, и присваиваем ей номер 1. Множество вершин разбивается теперь на три подмножества: *A, B* и *C*. Множество *A* состоит из всех уже пронумерованных вершин графа. Множество *B* определяется как смежное по отношению к *A,* т. е. *B* = Adj (*A*) и, таким образом, состоит из всех вершин, которые смежны какой-либо вершине из *A*. Множество *C* состоит из всех остальных вершин. Тогда на каждом шаге алгоритма следующей нумеруется та вершина подмножества *B,* которая ведет к присоединению наименьшего количества вершин из *C* к множеству *B*. После того как очередная вершина выбрана, производится соответствующее переопределение множеств *A, B* и *C*. Если граф не является связным, то алгоритм можно применять отдельно к каждой из его связных компонент. Примеры применения этого алгоритма к графу, изображенному на рис. 4.1, показаны на рис. 4.7. Минимальную степень имеют вершины 1, 10, 6, 2, 4, 8 и 5. Выбираем вершину 1 в качестве стартовой (эта вершина является также периферийной) и помечаем ее как первую. Исходная нумерация указывается цифрами внутри кружков, а новая — цифрами рядом с кружками. На первом шаге три введенные выше множества определяются следующим образом (в терминах исходной нумерации):

$$A = \{1\},$$
$$B = \{6,\ 10\},$$
$$C = \{\text{все остальные вершины}\}.$$

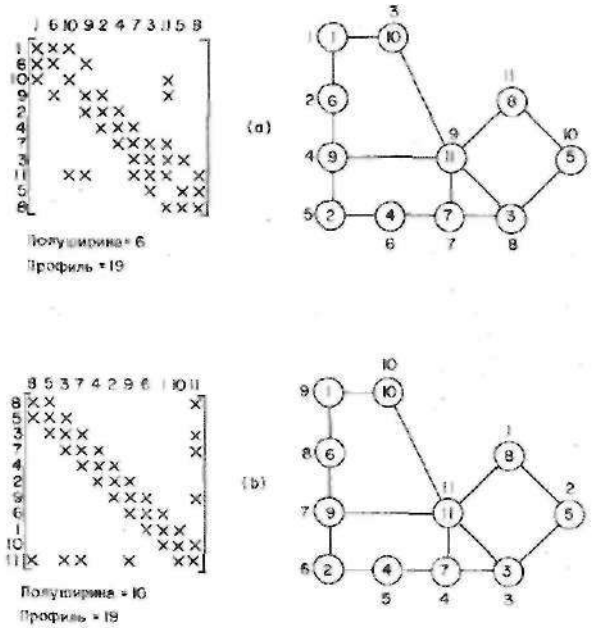


Рис. 4.7. Примеры применения алгоритма Кинга к графам, изображенным на рис. 4.1. Получающиеся переупорядоченные матрицы и новые помечивания графов показаны для случаев, когда в качестве стартовых выбраны (a) вершина 1 и (b) вершина 8.

Таким образом, кандидатами для нумерации служат вершины 6 и 10. Если следующей пронумеровать вершину 6, то в множество B будет включена вершина 9, а если взять вершину 10, то к множеству B тоже присоединится одна вершина 11. Возникшая неопределенность разрешается произвольным образом, и мы выбираем ту возможность, когда вершина 6 нумеруется в качестве второй. Переопределяя теперь множества A, B и C, получаем

$$A = \{1,\ 6\},$$
$$B = \{9,\ 10\},$$
$$C = \{\text{все остальные вершины}\}.$$

Теперь мы должны пронумеровать вершину 10, так как в этом случае к множеству B присоединяется лишь одна вершина 11. Читателю предоставляется завершить выполнение алгоритма самостоятельно в качестве упражнения. Получающаяся в результате переупорядочения матрица показана на рис. 4.7 (a). Ее профиль равен 19.

Если выбрать в качестве стартовой вершину 8, хотя и имеющую минимальную степень, но не являющуюся периферийной, то по-

лучается упорядочение, показанное на рис. 4.7 (b), с профилем, также равным 19. Другие выборы стартовой вершины приводят к следующим результатам: для вершины 4 профиль равен 22, для вершин 5 или 10 профиль опять равен 19; заметим, однако, что эти значения могут зависеть от принятой стратегии разрешения возникающих неопределенностей. Полуширина ленты, получаемая для указанных трех упорядочений, составляет 6, 5 и 10 соответственно, а для примеров, показанных на рис. 4.7, полуширина равна 6 и 10; видно, что все эти значения превосходят полуширину ленты, равную 4 (см. рис. 4.6), полученную в результате применения обратного алгоритма Катхилл—Макки. Таким образом, можно заключить, что ценой увеличения ширины ленты удается добиться уменьшения профиля; заметим, что обратный алгоритм Катхилл—Макки не использует эту возможность. Следует, однако, предостеречь читателя от того, чтобы на основе анализа приведенных простых примеров он пришел к выводу, что алгоритм Кинга всегда работает лучше, чем обратный алгоритм Катхилл—Макки.

Обратный алгоритм Кинга был предложен Катхилл [Cuthill, 1972], однако обнаружилось, что он ничем не лучше исходного алгоритма Кинга [Duff, 1977]. Вариант алгоритма Кинга, предложенный Леви [Levy, 1971], на каждом шаге использует в качестве кандидатов для нумерации вершины как из *B,* так и из *C*. Методом Леви можно получать дополнительное уменьшение профиля, однако вычислительные затраты при этом увеличиваются. Указывалось также, что для минимизации профиля можно использовать алгоритм Гиббса [Gibbs, Poole, Stockmeyer, 1976], обсуждавшийся в § 4.4.

Формулировка проблемы минимизации профиля в виде задачи целочисленного программирования [Gass, 1969] была предложена Тьюарсоном [Tewarson, 1967b]. Итерационный метод минимизации профиля был предложен в работе [Akyuz, Utku, 1968]. Этот метод неэкономичен, если использовать его независимо от других подходов, однако он дает неплохие результаты, если доступно хорошее начальное приближение к искомому упорядочению [Cuthill, 1972]. Таким образом, его можно применять для улучшения упорядочений, полученных каким-либо другим методом.

4.7. ТЕОРЕТИКО-ГРАФОВЫЕ ОСНОВЫ СИММЕТРИЧНОГО ГАУССОВА ИСКЛЮЧЕНИЯ

До сих пор мы рассматривали процесс накопления ненулевых элементов в треугольных множителях матрицы A с глобальной точки зрения. Теперь мы переходим к описанию более тонких алгоритмов, использующих детальный анализ процесса заполнения, и нашей первой задачей будет установление правил, определяющих действительные позиции ненулевых элементов. Пусть A —

симметричная разреженная матрица порядка n, а $G^A = (V, E)$ — отвечающий ей помеченный граф. Предположим, что к матрице A применяется гауссово исключение по столбцам, в результате чего получается разложение $A = U^T DU$. К началу k-го шага все поддиагональные элементы в столбцах с номерами 1, 2, ..., $k - 1$ исключены. Затем *k-я* строка, умноженная на соответствующие коэффициенты, вычитается из каждой строки, имеющей ненулевой элемент в *k-м* столбце под диагональю. При выполнении этих операций в строках $k + 1, \ldots, n$ справа от k-го столбца могут возникать новые ненулевые элементы. В принципе возможны взаимные уничтожения, в результате чего образуются новые нулевые элементы, однако на практике это происходит редко, за исключением некоторых специальных случаев, и этой возможностью мы будем пренебрегать. Рассмотрим активную подматрицу k-го шага (напомним, что активная подматрица образована элементами $a_{ij}^{(k)}$ с индексами $i, j \geq k$). Пусть G^k — граф, отвечающий k-й активной подматрице; его называют *графом исключения.* Множество вершин этого графа образовано последними $n - k + 1$ вершинами графа G^A, выбранными в соответствии с принятой нумерацией. Граф исключения содержит все ребра, которые присутствовали в G^A (в силу того что мы пренебрегли возможностью взаимных уничтожений), а также дополнительные ребра, отвечающие ненулевым элементам, возникшим в процессе выполнения предыдущих $k - 1$ шагов исключения. Последовательность графов $G^1 = G^A$, G^2, ... может быть построена рекурсивно с использованием следующего правила [Parter, 1961]:

Для того чтобы получить граф G^{k+1}, нужно удалить из графа G^k *k-ю* вершину [1]) и добавить все возможные ребра, связывающие между собой вершины, смежные в G^k с k-й вершиной. В качестве примера применения этого правила рассмотрим матрицу и граф, изображенные на рис. 4.1. На рис. 4.8 показаны графы исключения, соответствующие трем первым шагам. Соответствие между заполнением и новыми ребрами, добавляющимися к графу, представляется очевидным. Закончить построение последовательности графов исключения предоставляется читателю в качестве упражнения.

В терминах теории графов правило Партера означает, что смежное множество k-й вершины становится кликой после ее исключения. Таким образом, гауссово исключение приводит к систематическому порождению клик в графе. По мере продвижения процесса исключения клики увеличиваются, или множества клик объединяются и получаются большие клики; этот процесс получил название *слияния клик* [Duff, 1981a]. Клики обладают весьма полезными свойствами. Клика с m вершинами имеет $m(m - 1)/2$

[1]) Вместе со всеми инцидентными ей ребрами. — *Прим. перев.*

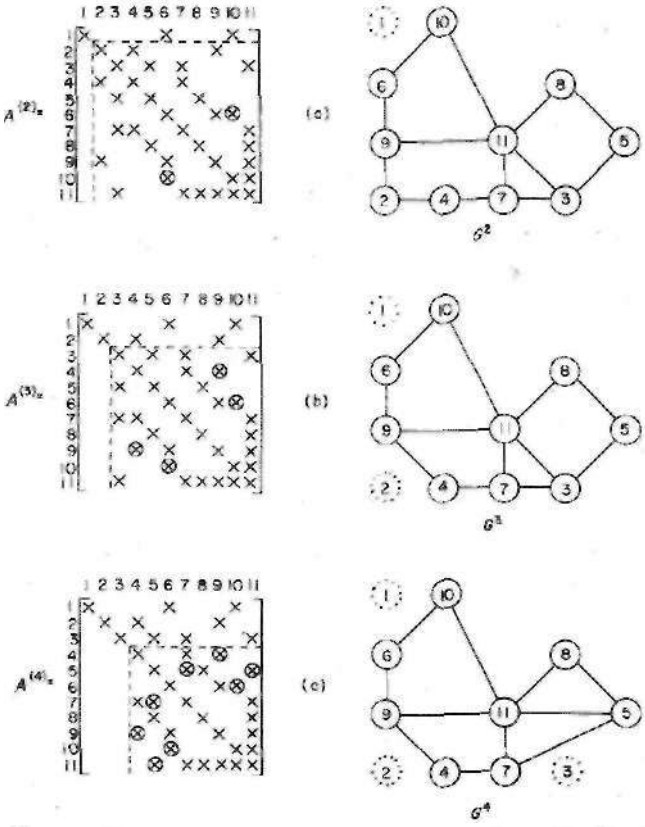


Рис. 4.8. Три первых шага исключения и соответствующие графы исключения для матрицы, изображенной на рис. 4.1 (а). Позиции заполнения обведены кружками.

ребер, однако в памяти ЭВМ она может быть представлена только списком вершин без каких-либо упоминаний о ребрах; для этого можно использовать, например, целые списки (см. обсуждение, приведенное в § 1.3). Клики, представленные подобным образом, могут быть слиты за время, пропорциональное количеству их вершин. С матричной точки зрения клика соответствует заполненной главной подматрице, и хранение списка вершин клики эквивалентно хранению списка номеров линий (линией считается строка или столбец), определяющих эту подматрицу. Поскольку формирование и слияние клик производится в процессе гауссова исключения систематически, можно достичь существенной экономии как памяти, так и вычислительных затрат, если использовать свойства клик при реализации символической части алгоритмов.

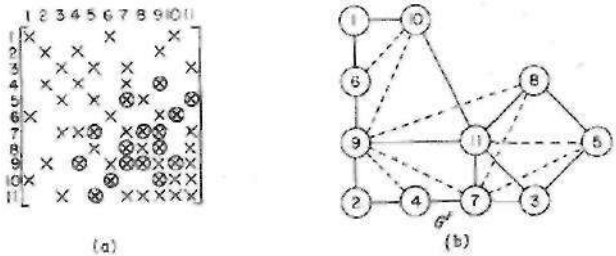


Рис. 4.9. Структура матрицы $U + U^T$ и соответствующий граф заполнения для матрицы, показанной на рис. 4.1 (а), полученные при выполнении исключения в естественном порядке.

Конечно, остается необходимость в хранении и переработке числовых значений всех элементов из заполненных главных подматриц.

Задача определения позиций заполнения может обсуждаться с различных точек зрения. Рассмотрим верхний треугольный множитель U. Эта матрица — несимметричная, и ей нельзя непосредственно поставить в соответствие неориентированный граф. Однако структура расположения ненулевых элементов над диагональю у матрицы U та же самая, что и у матрицы $U + U^T$; этой матрице можно сопоставить неориентированный граф G^F, называемый *графом заполнения.* Граф заполнения имеет n вершин; он содержит все ребра, которые были в графе G^A, и все дополнительные ребра, отвечающие ненулевым элементам, появившимся в процессе исключения. Если обозначить множество ребер, возникших при исключении, через F, то $G^F = (V, E \cup F)$, где $F \cap E = \emptyset$ и $G^A = (V, E)$. Следующее правило определяет множество ребер графа G^F, и тем самым расположение ненулевых элементов в матрице U [Parter, 1961]; (i, j) является ребром графа G^F в том и только в том случае, если либо (i, j) — ребро графа G^A, либо (i, k) и (j, k) суть ребра G^F для некоторого $k < i, j$. На рис. 4.9 показаны портрет матрицы $U + U^T$ и граф заполнения G^F, соответствующие матрице, изображенной на рис. 4.1 (а). Ребро (9, 11) принадлежит графу G^F, так как (9, 11) является ребром графа G^A. Ребро (8, 9) принадлежит графу G^F, так как ребрами графа G^F являются (8, 7) и (9, 7), причем $7 < 8$ и $7 < 9$. Заметим, что ребра (8, 7) и (9, 7) не входят в граф G^A. Читатель может убедиться в том, что любые две вершины, соединенные в графе G^F на рис. 4.9 пунктирной линией, смежны в свою очередь с другой вершиной, имеющей меньший номер. Множество F ребер возникших в процессе факторизации, образовано ребрами, показанными пунктиром.

Оба указанных выше правила существенно рекурсивны и поэтому не могут дать непосредственного ответа на вопрос о том, принадлежит ли некоторое ребро графу G^F или не принадлежит (т. е. на вопрос о том, равен ли нулю некоторый элемент матрицы U или нет), пока исключение не будет выполнено фактически или по крайней мере не будет смоделировано [1]). Следующая теорема, сформулированная в терминах достижимых множеств, введенных в § 4.2, дает прямой ответ на поставленный вопрос [George, Liu, 1976, Rose, Tarjan, Lueker, 1976]:

> Пусть $i < j$, и пусть Q_t — множество всех вершин с номерами, меньшими i. Тогда (v_i, v_j) является ребром графа G^F (т. е. элемент u_{ij} матрицы U отличен от нуля) в том и только том случае, когда $u_j \in$ Reach (v_i, Q_t) для *исходного* графа G^A.

Например, пусть требуется определить, принадлежит ли ребро (8, 9) графу G^F. Так как $i = 8$, то $Q_8 = \{1, 2, 3, 4, 5, 6, 7\}$. Наличие путей (8, 11) и (8, 5, 3, 7, 4, 2, 9), концевые вершины которых не принадлежат Q_8, показывает, что Reach $(8, Q_8) = \{9, 11\}$. Таким образом, $9 \in$ Reach $(8, Q_8)$ и поэтому (8, 9) является ребром G^F. Заметим, что при $i > j$ правило сохраняет силу, если поменять местами индексы i и j;

Отметим еще одно важное понятие, тесно связанное с предметом нашего обсуждения. Напомним, что разделителем S графа называется множество вершин, удаление которых делает граф несвязным, разбивая его на две или более связные компоненты; таким образом, в получающемся графе не существует путей, соединяющих две вершины, расположенные в различных связных компонентах. Отсюда немедленно следует, что если вершины разделителя S пронумерованы после вершин одной из связных компонент, и вершины v_i и v_j принадлежат различным компонентам, то $Q_i \cap S = \emptyset$ и v_j не принадлежит Reach (v_i, Q_i); поэтому не только $a_{ij} = 0$, но и $u_{ij} = 0$. Таким образом, непосредственно в треугольный множитель U вносится целый блок, состоящий только из нулевых элементов. Указанное свойство используется в методах сечений, которые обсуждаются в § 4.10 и 4.13. Так, для примера, приведенного на рис. 4.1, наличие разделителя {9, 11), который нумеруется после компоненты {2, 4, 7, 3, 5, 8}, приводит к тому, что ни одна из вершин этой компоненты не связана ни с одной вершиной оставшейся компоненты {6, 1, 10}, как может проверить читатель по графу G^F, показанному на рис. 4.9. Соответствующие элементы матрицы U равны нулю. Другим примером выбора разделителя может служить множество |7, 11, 8}, которое разъединяет множества {3, 5} и {10, 1, 6, 9, 2, 4}, как в этом легко может убедиться читатель.

---

[1]) Например, на уровне графов исключения. — *Прим. перев.*

Следующая теорема [George, Liu, 1981] может быть использована для характеризации графа исключения:

> Пусть u — вершина графа исключения G^k, и пусть Q_k, — множество вершин $\{v_1, v_2, \ldots, v_{k-1}\}$ (т. е. множество уже исключенных вершин исходного графа G^A). Тогда множество вершин, смежных с вершиной и в графе G^k, совпадает с множеством Reach (u, Q_h) в исходном графе G^A.

В качестве примера рассмотрим граф G^A, показанный на рис. 4.8 (с). Имеем $Q_4 = \{1, 2, 3\}$. Изучение графа G^A, приведенного на рис. 4.1, показывает, что Reach $(6, Q_4) = \{9, 10\}$, и это множество совпадает со смежным множеством вершины 6 в графе G^4. Другие случаи оставляются читателю в качестве упражнения.

4.8. АЛГОРИТМ МИНИМАЛЬНОЙ СТЕПЕНИ

В этом параграфе мы начинаем обсуждение алгоритмов, основанных на попытках уменьшить возникающее во время исключения заполнение при условии, что матрица системы симметричная, положительно определенная и разреженная, причем никаких предположений относительно структуры ее разреженности не делается (например, не предполагается, что матрица ленточная). В § 3.6 было показано, что любой диагональный элемент $a_{ii}^{(k)}$ при $i \geq k$ удовлетворяет условию устойчивости (т. е. принадлежит S_{st}) и поэтому может быть использован в качестве k-го главного элемента. Все алгоритмы, обсуждаемые в данной главе, используют это свойство, и главные элементы выбираются на диагонали в порядке, определяемом только из соображений сохранения разреженности.

Алгоритм минимальной степени [Tinney, 1969] считается лучшим методом общего назначения для выбора главного элемента. Этот метод достаточно прост, экономичен и эффективен; он широко используется по крайней мере для решения задач не слишком большого размера. Фактически алгоритм минимальной степени является симметричной версией алгоритма Марковица для несимметричных матриц, который будет обсуждаться в гл. 5. Главная идея алгоритма заключается в локальной минимизации заполнения и числа операций на каждом шаге исключения за счет выбора главного элемента в той строке и столбце, которые обеспечивают внесение наименьшего числа ненулевых элементов в треугольные множители. Рассмотрим гауссово исключение по столбцам, которое можно легко сочетать с алгоритмом минимальной степени. На k-м шаге исключения в поддиагональных позициях столбцов с 1-го по $(k - 1)$-й расположены нулевые элементы, как показано в примерах на рис. 4.8. Затем k-*я* строка нормируется

и кратные этой строки вычитаются из тех строк, которые имеют ненулевые элементы в поддиагональных позициях *k*-го столбца. Указанные ненулевые элементы *k*-го столбца образуют *k*-й столбец нижнего треугольного множителя U^T, в то время как ненулевые элементы в *k*-й строке, лежащие справа от диагонали, становятся ненулевыми элементами верхнего треугольного множителя *U*. Таким образом, если мы желаем минимизировать число ненулевых элементов матрицы *U* (или U^T), вычисляемых на *k-м* шаге, то для этого достаточно просмотреть активную подматрицу (образуемую строками матрицы $A^{(k)}$ с *k*-й по *n*-ю), выбрать строку (или столбец) с наименьшим числом ненулевых элементов, скажем *l-ю* строку (или столбец), и переставить *l-ю* строку с *k*-й строкой и *l*-й столбец с *k*-м столбцом в матрице $A^{(k)}$ перед тем, как приступить к выполнению *k*-го шага исключения.

При реализации на ЭВМ алгоритма минимальной степени требуется ввести массив целых чисел, скажем с именем NZ, и перед началом выполнения алгоритма задать в нем значения, равные числу внедиагональных ненулевых элементов в каждой строке матрицы *A*. Массив NZ модифицируется на каждом шаге исключения. К началу *k*-го шага в NZ (i), $i = k,\ldots n$, содержится число внедиагональных ненулевых элементов в *i*-й строке в столбцах с *k*-го по *n*-й. Строка с минимальным числом ненулевых элементов принимаемая в качестве главной, выбирается на основе анализа значений NZ (*i*), $i = k, \ldots, n$, после чего выполняется *k*-й шаг исключения. Значения элементов массива NZ корректируются во время выполнения исключения. Ясно, что в корректировке нуждаются только те элементы массива NZ, которые отвечают строкам, имеющим ненулевой элемент в *k*-м столбце. Когда исключается один из таких ненулевых элементов, расположенный, скажем, в позиции (*j*, *k*) матрицы $A^{(k)}$, мы вычитаем единицу из NZ (*j*), а затем прибавляем к NZ (*j*) единицу всякий раз, когда в *j*-й строке появляется новый ненулевой элемент. В алгоритме минимальной степени могут одновременно определяться и портрет матрицы *U*, и численные.значения ее ненулевых элементов. С другой стороны, если требуется решить несколько линейных систем с одним и тем же портретом, но с различными значениями ненулевых элементов, алгоритм может выполняться символически с целью определения переупорядочения матрицы *A* и портрета матрицы *U*. Затем при решении серии задач упорядочение и портрет матриц остаются неизменными, а гауссово исключение выполняется численно столько раз, сколько требуется.

Рассмотрим в качестве примера матрицу, показанную на рис. 4.1. Инициализация массива NZ осуществляется следующим образом:

позиция: 1 2 3 4 5 6 7 8 9 10 11

NZ: 2 2 3 2 2 2 3 2 3 2 5.

Видно, что есть несколько строк с двумя внеди а тональными элементами. Так как никаких соглашений относительно разрешения подобных неопределенностей не принималось, выберем любую из них, скажем первую строку, и перейдем к исключению. Так как ненулевые элементы первого столбца встречаются в строках. 6 и.10, то в коррекции нуждаются только значения NZ (6) и NZ (10); мы вычитаем из них единицу, а затем прибавляем число новых ненулевых элементов, появившихся после исключения в строках 6 и 10 соответственно. Полученная матрица $A^{(2)}$ показана на рис. 4.8 (а), а значения, содержащиеся в массиве NZ, теперь таковы:

позиция: 1 2 3 4 5 6 7 8 9 10 11
NZ: *x* 2 3 2 2 2 3 2 3 2 5,

где символом *«x»* помечено значение, которое нигде далее не используется.

Опять есть несколько строк с двумя внедиагональными ненулевыми элементами. Выбирая, например, в качестве главной вторую строку, мы продолжаем исключение и получаем матрицу $A^{(3)}$, изображенную на рис. 4.8 (b). Значения, содержащиеся в массиве NZ, станут такими:

позиция: 1 2 3 4 5 6 7 8 9 10 11
NZ: *x x* 3 2 2 2 3 2 3 2 5

На следующем шаге в качестве главной может быть выбрана любая строка с двумя внедиагональными ненулевыми элементами, скажем десятая. Читатель может проверить, что одно из возможных упорядочений, полученных в результате применения алгоритма минимальной степени, имеет вид, показанный на рис. 4.10. Такое упорядочение приводит к заполнению всего лишь 5 позиций в верхнем треугольнике переупорядоченной матрицы.

Рассмотренный выше подход к реализации метода минимальной степени имеет тот недостаток, что не удается заранее оценить требуемый объем памяти. Другой подход, основанный на некоторых результатах из теории графов, был описан в монографии Джорджа и Лю [George, Liu, 1981], где также приводится текст соответствующих программ на Фортране. При этом используется граф исходной матрицы *A,* так что требуемый объем памяти удается оценить, не прибегая к фактическому построению портрета матрицы *U.* На *k*-м шаге число внедиагональных ненулевых элементов в строке (или столбце) активной подматрицы равно степени соответствующей вершины в графе исключения G^k. Поэтому все, что нам нужно сделать

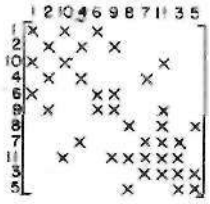


Рис. 4.10. Упорядочение минимальной степени для матрицы, показанной на рис. 4.1 (а).

на k-*м* шаге, — это выбрать вершину минимальной степени в графе G^k (что отвечает названию алгоритма) и пронумеровать ее как k-ю вершину, т. е. как следующую вершину, подлежащую исключению. Напомним теперь, что в соответствии с изложенным в § 4.7, множество вершин, смежных с вершиной v в G^k, есть не что иное, как достижимое множество Reach (v, Q_k), где $Q_k = \{v_1, \ldots, v_{k-1}\}$ — множество вершин, уже исключенных из графа, причем оператор Reach применяется к *исходному* графу G^A матрицы A. Кроме того, если вершина v исключается на k-м шаге, то следующий граф исключения G^{k+1} получается удалением вершины v из графа G^k и включением ребер, соединяющих между собой все вершины из Reach (v, Q_k), где оператор Reach опять применяется к исходному графу G^A. На k-м шаге среди вершин графа G^k степень могут изменить только вершины из Reach (v, Q_k). Таким образом, никаких дополнительных обращений к графу исключения не требуется. Массив NZ инициализируется значениями степеней вершин графа G^A. Затем, на k-м шаге, осуществляется поиск вершины минимальной степени путем просмотра позиций массива NZ с k-й по n-ю; найденную вершину помечают как v_k. После этого осуществляется коррекция позиций массива NZ, соответствующих вершинам из Reach (v_k, Q_k). Если $u \in$ Reach (v_k, Q_k), то степень u становится равной | Reach (u, Q_{k+1})|, где $Q_{k+1} = Q_k \cup \{v_k\} = \{v_1, v_2, \ldots, v_k\}$, а вертикальные черточки употребляются для обозначения количества элементов множества.

Джорджем и Лю было предложено еще одно усовершенствование алгоритма минимальной степени [George, Liu, 1981]. Оно основывается на понятии *неразличимых вершин*. На k-*м* шаге исключения две непомеченные вершины считаются неразличимыми, если

$$\text{Reach}\,(u, Q_k) \cup \{u\} = \text{Reach}\,(v, Q_k) \cup \{v\}. \qquad (4.9)$$

Приведем теперь краткую сводку свойств неразличимых вершин. Неразличимые вершины имеют одинаковые степени. Если вершины u и v оказались неразличимыми на k-*м* шаге, то они будут оставаться неразличимыми и на всех последующих шагах исключения до тех пор, пока одна из них не будет исключена. Далее, если u и v станут вершинами минимальной степени и вершина u будет исключена на некотором шаге, то вершина v будет продолжать оставаться вершиной минимальной степени, и ее можно исключать на следующем шаге, не прибегая к поиску вершины минимальной степени. Эти свойства используются для повышения эффективности алгоритма.

Наличие неразличимых вершин можно устанавливать при помощи следующих условий. Пусть C_1 и C_2 — две связные компо-

ненты подграфа $G(Q_k)$ [1]) на шаге k, и пусть

$$R_1 = \mathrm{Adj}(C_1), \qquad R_2 = \mathrm{Adj}(C_2). \tag{4.10}$$

Тогда вершины $R_1 \cap R_2$ следует расценивать в качестве кандидатов в неразличимые. Рассмотрим множество

$$T = R_1 \cup R_2 \cup C_1 \cup C_2. \tag{4.11}$$

Если

$$u \in R_1 \cap R_2 \tag{4.12}$$

и

$$\mathrm{Adj}(u) \subseteq T, \tag{4.13}$$

то

$$\mathrm{Reach}(u, Q_k) \cup \{u\} = R_1 \cup R_2. \tag{4.14}$$

Таким образом, все вершины из $R_1 \cap R_2$. смежные множества которых целиком лежат в T, являются неразличимыми. Указанное условие не предназначено для отыскания всех возможных множеств неразличимых вершин, однако обычно оно позволяет это сделать. Его главным достоинством является простота и легкость реализации. Кроме того, если все неразличимые вершины исключаются одна за другой путем нумерации их, скажем, как $k, k+1, \ldots, k+l$, то все соответствующие строки треугольного множителя U будут иметь одну и ту же структуру. Если применяется компактная схема хранения Шермана, то этот факт можно использовать для уменьшения накладной памяти (см. § 1.10).

Усовершенствованный алгоритм использует обсуждавшиеся выше идеи. Для графа $G^A = (V, E)$ алгоритм заключается в следующем:

Шаг 1 (инициализация). Полагаем $S \leftarrow \emptyset$ и определяем степени всех вершин графа G^A.

Шаг 2 (выбор). Выбираем вершину $u \in V \setminus S$, имеющую минимальную степень.

Шаг 3 (исключение). Пусть W — множество, образованное вершиной u и всеми вершинами, неразличимыми с u. Нумеруем вершины из W как следующие вершины упорядочения.

Шаг 4 (коррекция степеней). Для $v \in \mathrm{Reach}(u, S) \setminus W$ корректируем степень вершины v, полагая ее равной $|\mathrm{Reach}(v, S \cup W)|$. Определяем неразличимые вершины в множестве $\mathrm{Reach}(u, S) \setminus W$.

---

[1]) Через $G(Q)$ обозначается частичный подграф, определяемый подмножеством вершин $Q \subseteq V$. — *Прим. перев.*

Шаг 5 (продолжение цикла или прекращение внполнения). Полагаем $S \leftarrow S \cup W$. Если $S = V$, то прекращаем выполнение алгоритма; в противном случае переходим к шагу 2.

В результате работы алгоритма строится, как уже указывалось, упорядочение минимальной степени. Кроме того, с целью подготовки структуры данных для хранения верхнего треугольного множителя U можно получить таблицу, задающую число ненулевых элементов в каждой строке U путем запоминания вычисляемой на шаге 3 степени каждой вершины непосредственно перед ее исключением. С вычислительной точки зрения наличие такой таблицы представляется весьма желательным, так как появляется возможность заранее определить необходимый объем памяти. Заметим, однако, что после выполнения алгоритма пользователь может обнаружить, что памяти не хватает, или использование такого объема памяти обойдется слишком дорого, в результате чего будет принято решение о применении других методов. Если же пользователь располагает достаточным объемом памяти ЭВМ и принимает решение о продолжении процедуры, то единственным пока полезным результатом окажется полученное упорядочение, в то время как для определения портрета матрицы U пользователь должен будет применить другой алгоритм.

С другой стороны, стандартный алгоритм минимальной степени существенно проще; он строит упорядочение и портрет U одновременно, и может даже вычислять соответствующие числовые значения в том же проходе. Его недостаток в том, что невозможно предсказать требуемый объем памяти. Пользователь должен выбирать это значение наугад. Если отведенная для работы алгоритма память исчерпывается, то его выполнение прекращается, причем не дается никакого ответа на вопрос о том, какой объем памяти будет достаточным. В настоящее время наблюдается тенденция к удешевлению памяти, так что указанный недостаток является, может быть, не столь серьезным.

В случае когда алгоритм упорядочения не строит портрет матрицы U, его приходится определять отдельно. Существует алгоритм, позволяющий сделать это за время, пропорциональное числу внедиагональных ненулевых элементов матрицы U. Идея алгоритма [Rose, Tarjan, Lueker, 1976] лучше всего может быть описана в терминах достижимых множеств. Как уже указывалось в § 4.7, структура i-й строки матрицы U вполне определяется множеством Reach (v_i, Q_i) в исходном графе G^A, где $Q_i = \{v_1, v_2, \ldots, v_{t-1}\}$. Таким образом, требуется определить множества Reach (v_i, Q_i) для $i = 1, 2, \ldots, n - 1$, и это можно сделать, используя следующее рекурсивное соотношение. Рассмотрим k-ю строку матрицы U и обозначим через m_k столбцовый индекс первого нену-

левого элемента в этой строке. В терминах теории графов величина m_k может быть определена следующим образом:

$$m_k = \min \{j \mid v_j \in \text{Reach}\,(v_k,\ Q_k)\}. \tag{4.15}$$

Теперь рассмотрим *i-ю* строку матрицы *U*, и пусть *W* — множество всех вершин v_k, таких что $k < i$ и $m_k = i$. В матричных терминах, *W* — это множество номеров строк *U,* имеющих первый ненулевой элемент в *i*-м столбце. Тогда

$$\text{Reach}\,(v_i,\ Q_i) = \text{Adj}\,(x_i) \cup \left(\bigcup_k \{\text{Reach}\,(v_k,\ Q_k) \mid v_k \in W\} \right) \setminus Q_{i+1}. \tag{4.16}$$

Другими словами, мы подвергаем слиянию портреты строк *U* с номерами в *W* и портрет *i*-й строки матрицы *A* для того, чтобы получить портрет *i*-й строки матрицы *U*. Преимущество описанной процедуры заключается в том, что не привлекаются другие строки, содержащие ненулевые элементы в *i*-м столбце *U*.

Рассмотрим, например, граф, изображенный на рис. 4.1, и вычислим Reach (7, Q_7)· Допустим, что в нашем распоряжении находятся все предыдущие достижимые множества:

$$\begin{aligned}
&\text{Reach}\,(1,\ Q_1) = \{6,\ 10\},\ m_1 = 6,\\
&\text{Reach}\,(2,\ Q_2) = \{4,\ 9\},\ m_2 = 4,\\
&\text{Reach}\,(3,\ Q_3) = \{5,\ 7,\ 11\},\ m_3 = 5,\\
&\text{Reach}\,(4,\ Q_4) = \{7,\ 9\},\ m_4 = 7,\\
&\text{Reach}\,(5,\ Q_5) = \{7,\ 8,\ 11\},\ m_5 = 7,\\
&\text{Reach}\,(6,\ Q_6) = \{9,\ 10\},\ m_6 = 9.
\end{aligned}$$

Таким образом, $W = \{4,\ 5\}$ в силу $m_4 = m_5 = 7$. Мы должны выполнить слияние множеств Adj (7) $= \{3,\ 4,\ 11\}$, Reach (4, Q_4) и Reach (5, Q_6) и удалить вершины с номерами, меньшими или равными 7. Получаем

$$\text{Reach}\,(7,\ Q_7) = \{8,\ 9,\ 11\},$$

что является правильным результатом. Читатель может проверить правильность построения приведенных достижимых множеств путем просмотра матрицы заполнения и графа заполнения, указанных на рис. 4.9.

4.9. ДРЕВОВИДНОЕ РАЗБИЕНИЕ СИММЕТРИЧНОЙ РАЗРЕЖЕННОЙ МАТРИЦЫ

Может оказаться, что граф матрицы представляет собой дерево. Если матрица переупорядочена в соответствии с монотонной нумерацией этого дерева, то гауссово исключение с последовательным выбором главных элементов вдоль диагонали не приводит

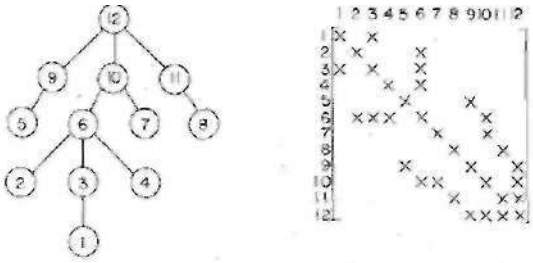


Рис. 4.11. Монотонное упорядочение дерева и соответствующий портрет матрицы.

к возникновению заполнения [Parter, 1961]. Это важное свойство иллюстрирует рис. 4.11, на котором показано монотонно упорядоченное дерево и соответствующая матрица. Читатель может легко убедиться в наличии упомянутого свойства. Деревья являются частным случаем класса триангулированных графов [Rose, 1970], характеризующихся отсутствием заполнения при выполнении гауссова исключения.

Если граф матрицы не является деревом, то часто оказывается возможным найти древовидное разбиение графа, т. е. разбиение, при котором фактор-граф представляет собой дерево. Составные вершины фактордерева отвечают подмножествам строк и столбцов матрицы. Таким образом, древовидное разбиение порождает разбиение матрицы, которую можно теперь рассматривать как *блочную*. Элементы блочной матрицы суть матрицы, причем диагональные блоки квадратные, и граф этой матрицы совпадает с фактор-деревом. Таким образом, открывается возможность использования преимуществ, даваемых «хорошими» свойствами деревьев, в случае разреженных матриц общего вида.

В этом параграфе будет обсуждаться *алгоритм измельченного фактордерева* [George, 1977, George, Liu, 1978b], который используется для построения как можно более подробного древовидного разбиения неориентированного графа.

Древовидным разбиением может служить структура уровней графа; при этом каждый уровень является членом разбиения, и этому разбиению соответствует блочно-трехдиагональное разбиение матрицы. Корневая структура уровней, имеющая большую длину, получается, если в качестве корня выбрать псевдопериферийную вершину. Алгоритм Джорджа использует в качестве исходной структуру уровней с корнем в псевдопериферийной вершине и находит более подробное, «измельченное» разбиение (имеющее большее число членов). Алгоритм измельчения основан на том наблюдении, что частичный граф, отвечающий некоторому

уровню, объединенному со всеми последующими уровнями, может быть несвязным. Например, на рис. 4.2(b) частичный граф, отвечающий уровням L_4, L_5 и L_6, имеет две связные компоненты, и частичный граф, отвечающий уровням L_5 и L_6, также имеет две связные компоненты. Фактордерево, изображенное на рис. 4.2 (c), отражает этот факт. Алгоритм отыскания псевдопериферийной вершины обсуждался в § 4.4.

Рассмотрим граф $G = (V, E)$, и пусть L_0, L_1, ..., L_m—структура уровней длины $m + 1$ с корнем в псевдопериферийной вершине. Чтобы объяснить, как работает алгоритм Джорджа, рассмотрим сначала частичный граф $G(L_l)$, образованный всеми вершинами уровня L_l, и всеми теми ребрами, оба конца которых лежат в L_l. Граф G (L_l) может быть несвязным. Пусть S — любое подмножество вершин L_l. Множество

$$Y = \text{Span}(S, G(L_l)) \tag{4.17}$$

определяется как объединение S и тех вершин из L_l которые соединены с вершинами из S путями, целиком лежащими в G (L_l). Если S состоит из единственной вершины v, то Y представляет собой связную компоненту L_l, содержащую эту вершину, однако в общем случае Y может состоять из нескольких связных компонент L_l. Например, для графа, изображенного на рис. 4.2(b), при $l = 5$ имеем $L_5 = \{13, 15, 5, 20\}$. Если $S = \{20\}$, то Span ($S, G(L_5)$) = $\{5, 20\}$. Если же $S = \{13, 5\}$, то Span (S, $G(L_5)$) = $= \{13, 15, 5, 20\}$, т. е. это множество включает в себя две связные компоненты.

Алгоритм использует стек (см. § 1.2) для временного хранения подмножеств S в том случае, когда они не могут использоваться непосредственно после их построения. При помощи подмножеств S осуществляется частичное формирование членов разбиения, ожидающих полного завершения построения. Алгоритм можно описать следующим образом:

Шаг 0 (инициализация). Найти псевдопериферийную вершину r, построить структуру уровней L_0, L_1, ..., L_m с корнем в вершине r, положить $l = m$ выбрать любую вершину v, принадлежащую уровню L_m. Положить $S = \{v\}$ и очистить стек.

Шаг 1 (пополнение S и удаление из стека). Если стек пуст, то перейти к шагу 2. В противном случае, приняв, что T есть множество узлов на вершине стека, перейти к шагу 2, если $T \cap L_l$ пусто, и удалить T из стека, положив S $\leftarrow$ $\leftarrow S \cup T$, если оно непусто.

Шаг 2 (формирование возможного члена разбиения). Определить множество $Y \leftarrow$ Span (S, G (L_l)). Если какая-либо вершина из Adj (Y) принадлежит L_{l+1}, и эта вершина еще не

присоединена к члену разбиения, то перейти к шагу 5. В противном случае принять Y в качестве нового члена разбиения.

Шаг 3 (внутреннее помечивание нового члена разбиения). Произвести нумерацию множества $Y \setminus S$ при помощи обратного алгоритма Катхилл—Макки. Затем пронумеровать S в произвольном порядке.

Шаг 4 (следующий уровень). Положить $l \leftarrow l - 1$. Если $l < 0$, то прекратить выполнение алгоритма. В противном случае сформировать множество $S \leftarrow \text{Adj}(Y) \cap L_l$ и перейти к шагу 1.

Шаг 5 (частично сформированный член разбиения). Поместить S на вершину стека. Выбрать вершину v_{l+1}, смежную с Y и принадлежащую L_{l+1} после чего проложить путь v_{l+1}, v_{l+2}, ... , v_{l+t}, каждая вершина которого v_{l+i}, принадлежит уровню L_{l+i} и концевая вершина v_{l+i} такова, что путь нельзя продолжить на следующий уровень, т. е. v_{l+t} не имеет смежных вершин в L_{l+t+1}. Положить $S \leftarrow \{v_{l+t}\}$ и перейти к шагу 1.

К началу шага 3 множество S состоит из всех вершин множества Y, смежных с некоторой вершиной из L_{l+1}. Таким образом, если множество $Y \setminus S$ непусто, то оно состоит из всех вершин множества Y, не смежных с вершинами множества L_{l+1}. Упорядочивая сначала $Y \setminus S$ при помощи обратного алгоритма Катхилл—Макки, мы добиваемся того, что профиль диагональных блоков матрицы уменьшается, и это позволяет применить для их хранения экономичную схему Дженнингса (см. § 1.6). Порядок, в котором определяются члены разбиения, непосредственно обеспечивает монотонное упорядочение фактордерева, как и требуется в условии теоремы Партера.

В качестве примера применения алгоритма рассмотрим граф, изображенный на рис. 4.2 (а). Если в качестве корня структуры уровней выбрана периферийная вершина $r = 8$, то получается структура, показанная на рис. 4.2 (b). Первое множество S, определяемое на шаге 1, есть $S = \{6\}$; на шаге 2 определяется $Y = \{6\}$ и вершина 6 образует первый член разбиения и помечивается единицей. Алгоритм продолжается на шаге 4, где строится множество $S = \{5, 20\}$; на шаге 1 множество S не изменяется, а на шаге 2 алгоритм вычисляет $Y = \{5, 20\}$, и это множество принимается в качестве второго члена разбиения.

На шаге 4 алгоритм находит $S = \{7\}$, единственная вершина на уровне 4, смежная с вершинами 5 и 20. S не изменится на шаге 1, так как стек все еще пуст; на шаге 2 алгоритм определяет $Y = \{14, 16, 1, 7\}$ и находит, что Y имеет вершины, смежные с вершинами из L_5, которые еще не включены в разбиение. Таким

образом, множество S = {7} помещается на вершину стека и совершается переход к шагу 5, где после выбора v_5 = 13 определяется, что требуемый путь не может быть проложен из этой вгршины, так что $i = 1$. Построив S = {13}, алгоритм переходит к шагу 1, где S не изменяется, и затем на шаге 2 определяется множество $Y = \{13, 15\}$, которое образует третий член разбиения.

Затем на шаге 4 алгоритм определяет S = {14}. Так как на этот раз на вершине стека находится множество {7}, и множества {7} и {14} принадлежат одному и тому же уровню, то на шаге 1 происходит коррекция множества S и оно становится равным {14, 7}. Переходя к шагу 2, получаем Y = {14, 16, 1, 7}, и это множество принимается в качестве четвертого члена разбиения. Читатель может продолжить выполнение алгоритма. В результате получается фактордерево, показанное на рис. 4.2(c), помеченное в монотонном порядке. Множества вершин, принадлежащих каждой составной вершине, указаны внутри кружков, а вовне их указано помечивание.

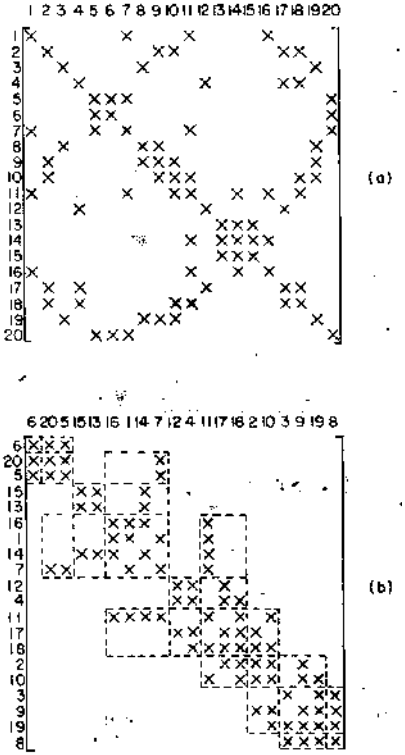


Рис. 4.12. Алгоритм измельченного фактордерева: (a) структура матрицы, отвечающей графу, изображенному на рис. 4.2 (a); (b) блочный вид переупорядоченной матрицы, отвечающий фактордереву, изображенному на рис. 4.2 (c).

На рис. 4.12 (a) изображен портрет разреженной матрицы, отвечающий графу, показанному на рис. 4.2 (a). При выполнении гауссова исключения над такой матрицей сильное заполнение будет происходить в самых различных ее частях. На рис. 4.12 (b) показана переупорядоченная матрица с блочной структурой, отвечающей фактордереву на рис. 4.2 (c), которое построено и пронумеровано в соответствии с описанным алгоритмом. Если выполнить гауссово исключение над этой матрицей, то заполнение будет небольшим и, кроме того, оно будет ограничено диагональными блоками.[1])

[1]) Если (в общем случае) хранить внедиагональные блоки матрицы U в факторизованном виде, см. монографию Джорджа и Лю. — *Прим. перев.*

Экспериментальные расчеты тестовых задач проводились Джорджем [George, 1977]. Символическая факторизация блочных матриц требует специального рассмотрения, так как в отличие от скалярного случая, где произведение двух ненулевых элементов всегда будет ненулевым элементом, произведение двух ненулевых разреженных блоков может представлять собой нулевой блок. Такие нулевые блоки не повлекут заполнения и поэтому не должны рассматриваться при построении блочной структуры треугольных множителей. Алгоритм, учитывающий эту особенность, был предложен в работе [George, Rashwan, 1981].

4.10. ВЛОЖЕННЫЕ СЕЧЕНИЯ

Алгоритмом *вложенных сечений* называется метод, предложенный Джорджем [George, 1973], предназначенный для систематического разбиения графа матрицы при помощи разделителей (соответствующие определения были даны в § 4.2). После того как разделитель найден, его вершины помечиваются и удаляются из графа, в результате чего граф распадается на две (или более) связные компоненты. Затем находятся разделители для каждой связной компоненты, и процедура продолжается, формируя все меньшие и меньшие сечения, «вставляемые» в граф; когда все вершины графа окажутся пронумерованными, выполнение процедуры прекращается. После этого осуществляется соответствующая перестановка строк и столбцов матрицы. *Упорядочение вложенных сечений,* получаемое указанным образом, имеет ряд полезных свойств.

Изложенная идея проиллюстрирована на рис. 4.13. Предположим, что множество вершин представлено прямоугольником R_0, изображенным на рис. 4.13(а). Выберем разделитель S_0, образованный множеством вершин, удаление которых разъединяет граф на две (в нашем случае) компоненты R_1^1 и R_1^2. При упорядочении сначала нумеруются вершины из R_1^1, затем вершины из R_1^2 и, наконец, вершины из S_0. Портрет матрицы, соответствующий такому упорядочению, показан на рис. 4.13 (b), где заштрихованы области, которые могут содержать ненулевые элементы. Это упорядочение обладает тем важным свойством, что заполнение, возникающее при гауссовом исключении, также оказывается локализованным в заштрихованных областях матрицы.

Процедура повторяется рекуррентно. Множество R_1^1 разбивается разделителем S_1^1 на две изолированные компоненты R_2^1 и R_2^2, а множество R_1^2 — на R_2^3 и R_2^4 при помощи разделителя S_1^2. Затем эти множества перенумеровываются в порядке R_2^1, R_2^2, S_1^1, R_2^3, R_2^4, S_1^2. Как и раньше, множество S нумеруется в последнюю очередь. Новое упорядочение матрицы показано на рис. 4.13 (с). И в этом случае заполнение будет сосредоточено в заштрихован-

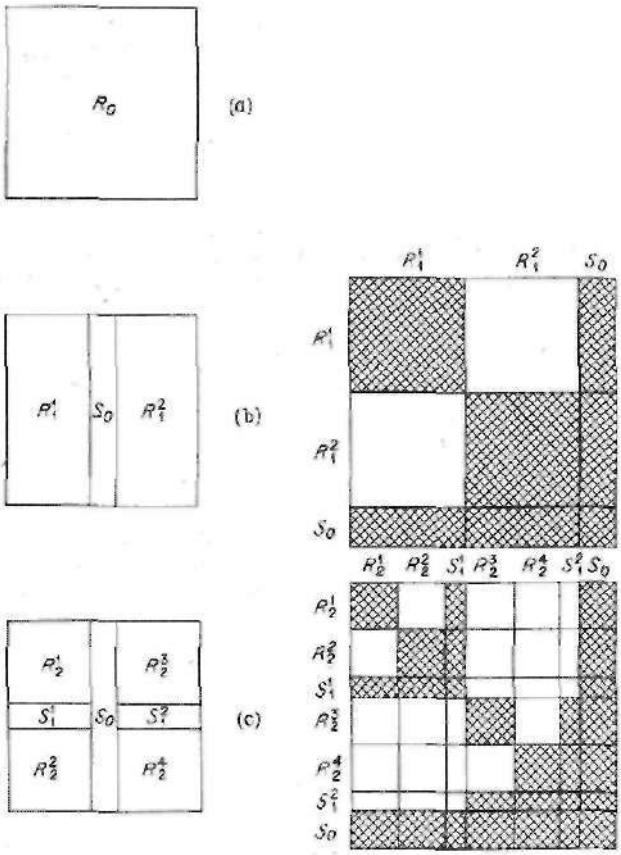


Рис. 4.13. Иллюстрация метода вложенных сечений.

ных областях матрицы, которые теперь стали меньше. В процессе выполнения процедуры и измельчения разбиения возникают множества *R,* в которых нельзя найти разделитель. В случае появления такого *R* в качестве соответствующего множества 5 берется само множество *R* и нумеруются его вершины. Выполнение процедуры прекращается, если все множества типа *R* исчерпаны. Существует взаимно однозначное соответствие между каждым множеством R_i^l и его разделителем S_i^l, и с получающимся разбиением можно связать *дерево вложенных сечений.* Каждое множество типа *S* образует вершину дерева, корнем дерева является S_0, сыновьями вершины S_0 считаются вершины S_1^1 и S_1^2 и т. д. *Терминальными членами* дерева сечений служат множества типа S, которые выбраны равными соответствующим множествам типа *R,*

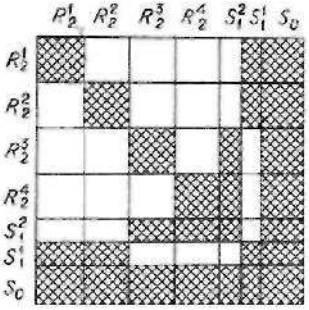


Рис. 4.14. Улучшенная версия метода вложенных сечений.

так как они не имеют потомков. *Высотой h* разбиения вложенных сечений называют максимальное значение индекса *l*, для которого по крайней мере одно из множеств R_l^i непусто. Заметим, что дерево сечений не совпадает с факторграфом, отвечающим построенному разбиению.

Прежде чем перейти к полному описанию алгоритма, сделаем несколько замечаний. Перенумерация вершин множеств типа *R* на каждом шаге оказывается неудобной, если ее выполнять так, как было показано на предыдущем примере, и на практике матрицу упорядочивают так, как показано на рис. 4.14; при этом свойство локализации заполнения сохраняется. Вершины каждого множества типа *S* нумеруются в обратном порядке [1]), как только такое множество будет построено, в то время как множества типа *R* остаются непронумерованными. Таким образом, окончательная нумерация получается непосредственно после окончания выполнения алгоритма.

При использовании упорядочения вложенных сечений выгодно, чтобы разделители имели небольшое количество вершин, так как в этом случае области матрицы, где будет происходить заполнение, станут невелики. В алгоритме используются минимальные разделители, так, что никакое собственное подмножество разделителя само разделителем не является. Разбиение вложенных сечений, полученное при помощи минимальных разделителей, будем называть *минимальным разбиением вложенных сечений.* Удобно также, чтобы множества типа *R,* получаемые при введении каждого разделителя, имели сравнимый размер, так как в этом случае обычно получаются разбиения более хорошего качества.

До сих пор мы не обсуждали, в каком именно порядке следует нумеровать вершины внутри каждого из множеств типа *S*. Для минимального разбиения вложенных сечений было доказано [George, Liu, 1976], что при любом внутреннем упорядочении множеств типа S степень заполнения и число операций при выполнении гауссова исключения остаются неизменными. В случае когда хранятся и обрабатываются только ненулевые элементы, множества S могут быть пронумерованы произвольным образом. Если, однако, используется схема, предусматривающая хранение

[1]) То есть в направлении от *N* к 1, так что, например, вершины S_0 получают номера *N*, *N* — 1, ... — *Прим. перев.*

части нулевых элементов, то может оказаться удобным применять для каждого множества типа S некоторое специальное упорядочение. Например, если хранение диагональных блоков осуществляется в соответствии со схемой Дженнингса (см. § 1.6), то удобно занумеровать терминальные множества типа S при помощи обратного алгоритма Катхилл—Макки (см. § 4.6), так как в этом случае происходит уменьшение профиля [George, 1977]. Эту идею не удается непосредственно применить к нетерминальным членам разбиения, так как соответствующие диагональные блоки претерпевают полное заполнение. Однако в некоторых случаях все же оказывается удобным использовать обратный алгоритм Катхилл—Макки для нумерации нетерминальных разделителей, поскольку в этом случае сокращается требуемый объем накладной памяти. Если рассматриваемая матрица возникла в результате конечно-элементной аппроксимации (см. § 4.2), то можно рекомендовать для упорядочения нетерминальных членов разбиения улучшенную версию обратного алгоритма Катхилл—Макки [Brown, Wait, 1981], так как в этом случае достигается уменьшение осцилляции в упорядочении узлов сетки.

Перейдем теперь к обсуждению *алгоритма автоматических вложенных сечений.* Исходя из неориентированного графа матрицы, алгоритм осуществляет построение структуры уровней с корнем в некоторой псевдопериферийной вершине. Структура уровней с корнем в псевдопериферийной вершине будет, вероятно, иметь много уровней, и полученное разбиение окажется поэтому достаточно мелким. Затем из «среднего» уровня выбирается минимальный разделитель, и его вершины нумеруются в обратном порядке [1]) каким-либо удобным способом. Для того чтобы явно описать алгоритм, мы опустим индексы l и j, введенные ранее для теоретических целей. Таким образом, через S и R в алгоритме обозначаются текущие множества S_l^j и R_l^j. Вводится также множество C, определяемое на каждом шаге как объединение всех разделителей, найденных на данном шаге и на всех предыдущих. Вертикальные черточки используются для обозначения количества элементов, содержащихся в множестве, т. е. $|S|$ = числу вершин в S. Алгоритм формулируется следующим образом:

Шаг 1. Пусть задан граф $G = (V, E)$, и пусть $N = |V|$ — число вершин графа. Положим $C = \emptyset$.

Шаг 2. Если $N = 0$, то прекратить выполнение алгоритма. В противном случае рассмотреть частичный граф $G(V \setminus C)$, полученный удалением из исходного графа всех вершин, принадлежащих C, вместе с инцидентными им ребрами. Найти связную компоненту графа $G(V \setminus C)$, скажем $G(R)$,

[1]) См. примечание на стр. 154. — *Прим. перев.*

и определить псевдопериферийную вершину u в графе $G(R)$. Псевдопериферийные вершины можно определять при помощи алгоритма Гиббса, описанного в § 4.4.

Шаг 3. В подграфе $G(R)$ построить структуру уровней $L_0, L_1, \ldots, L_m$ с корнем в u. Если $m < 2$, то положить $S \leftarrow R$ и перейти к шагу 5. В противном случае выбрать средний уровень с номером j, равным наибольшему целому числу, не превосходящему значения $(m + 1)/2$.

Шаг 4. Выбрать подмножество $S \subseteq L_j$, такое что S — минимальный разделитель графа $G(R)$.

Шаг 5. Множество S принимается в качестве нового члена разбиения. Положить $C \leftarrow C \cup S$ и пронумеровать вершины множества S числами от $N - |S| + 1$ до N.

Шаг 6. Положить $N \leftarrow N - |S|$ и перейти к шагу 2.

Во многих практически важных случаях хорошее разбиение может быть получено и тогда, когда не все разделители являются минимальными; можно требовать лишь, чтобы они были достаточно малы. Такие разделители можно определить на шаге 4, выбирая только те вершины из L_j, которые смежны с какой-либо вершиной из L_{j+1}. Другими словами, множество S получается исключением из L_j всех вершин, которые не смежны с вершиной из L_{j+1}. Такой разделитель, однако, может и не быть минимальным, как показывает пример, приведенный ниже. Реализация этого алгоритма на ЭВМ описана в монографии Джорджа и Лю [George, Liu, 1981]. Вложенные сечения могут сочетаться с частичным выбором главных элементов [Gilbert, Schreiber, 19821. В качестве примера рассмотрим граф, изображенный на рис. 4.2 (а), и его структуру уровней с корнем в периферийной вершине 8, показанную на рис. 4.2 (b). Так как $m = 6$, то $j = 3$, и первый разделитель должен быть выбран из множества $L_3 = \{17, 18, 11\}$. Однако L_3 не является минимальным разделителем, так как разделителями будут и множества {11} и {17, 18}. Для того чтобы граф был разбит на компоненты сравнимого размера, мы выберем $S_0 = \{11\}$ и пометим вершину 11 как двадцатую, см. рис. 4.15 (а). Удаление вершины 11 разбивает граф на две связные компоненты $R_1^1 = \{12, 4, 17, 18, 2, 10, 9, 19, 8, 3\}$ и $R_1^2 = \{13, 15, 14, 16, 1, 7, 5, 20, 6\}$.

Рассмотрим теперь R_1^1. Структура уровней с корнем в вершине 12 имеет длину $m = 5$, так что $j = 3$ и $S_1^1 = L_3 = \{9, 10\}$. Вершины 9 и 10 помечаем как девятнадцатую и восемнадцатую соответственно. Для R_1^2 периферийной является вершина 13, соответствующая структура уровней имеет длину $m = 6$ и для нее $j = 3$. Поэтому $S_1^2 = L_3 = \{1\}$. Вершину 1 помечаем как семнадцатую.

На этом этапе множество C представляет собой объединение S_0, S_1^1 и S_1^2, и его удаление приводит к образованию следующих

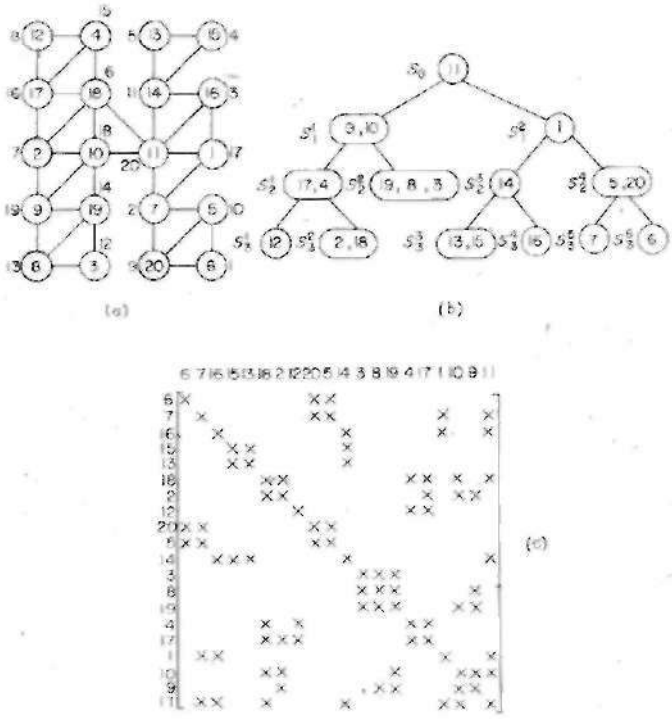


Рис. 4.15. Применение метода вложенных сечений к графу, изображенному на рис. 4.2 (a). Новое помечивание графа показано на рис. (a), дерево вложенных сечений — на рис. (b), а переупорядоченная матрица — на рис (c).

связных компонент: $R_2^1 = \{12, 4, 17, 18, 2\}$, $R_2^2 = (19, 8, 3\}$, $R_2^3 = \{13, 15, 14, 16\}$, $R_2^4 = \{7, 5, 20, 6\}$. Разделителем для R_2^1 является множество $S_2^1 = \{17, 4\}$. Множеству R_2^2 отвечает клика и поэтому нетривиального разделителя не существует, так что $S_2^2 = R_2^2$. Два других разделителя суть $S_2^3 = \{14\}$ и $S_2^4 = \{5, 20\}$. Вершины этих новых разделителей помечаются и удаляются из графа. Все оставшиеся множества типа R не имеют нетривиальных разделителей. Помечивание, полученное в результате выполнения алгоритма, показано на рис. 4.15 (a) цифрами вне кружков, а дерево сечений изображено на рис. 4.15 (b). Структура переупорядоченной матрицы показана на рис. 4.15 (c). Если над этой матрицей выполнить гауссово исключение, то заполнение составит только 14 позиций, в то время как для первоначального упорядочения на рис. 4.12 (a) оно составляет 27 позиций, а для упорядочения измельченного фактордерева — 2 позиции (см.

рис. 4.12 (b)). Конечно, не следует экстраполировать результаты, полученные для этого простого примера. Экспериментальные расчеты тестовых задач описаны Джорджем [George, 1977].

4.11. СВОЙСТВА УПОРЯДОЧЕНИЙ ПО МЕТОДУ ВЛОЖЕННЫХ СЕЧЕНИЙ

В данном параграфе будут описаны некоторые свойства упорядочений вложенных сечений, которые были доказаны Джорджем и Лю [George, Liu, 1976] в виде соответствующих теорем. Знание этих свойств представляется возможным для хорошего понимания излагаемого материала.

Рассмотрим граф G = (*V, E*) с разбиением, соответствующим методу вложенных сечений, и сравним различные нумерации его вершин. Говорят, что нумерация *совместима* с принятым разбиением, если вершины каждого члена разбиения пронумерованы последовательными целыми числами. На рис. 4.16 (a) показан граф с разбиением вложенными сечениями на три компоненты. Указанная на этом рисунке нумерация является совместимой с данным разбиением. Другая возможная совместимая нумерация показана на рис. 4.16 (c), в то время как рис. 4.16 (b) дает пример нумерации, несовместимой с разбиением. Алгоритмы, описанные в предыдущем параграфе, генерируют нумерации, совместимые с разбиением. Заметим, что порядок, в котором занумерованы вершины каждого из разделителей, несуществен; важно лишь то, что для нумерации разделителя используются последовательные целые числа.

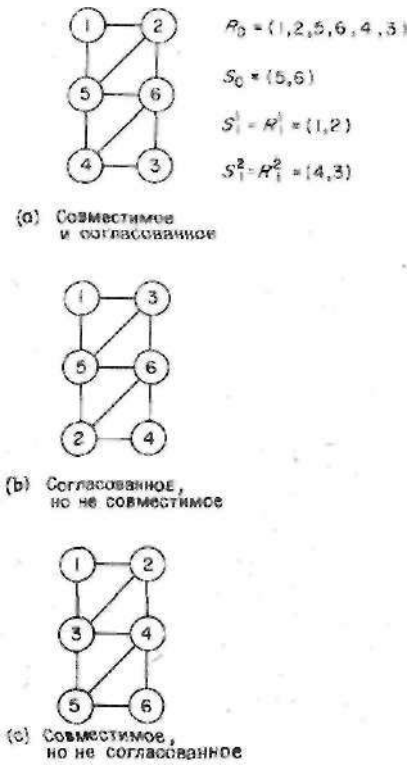


Рис. 4.16. Три возможные нумерации вершин графа с разбиением вложенными сечениями.

Рассмотрим теперь любое из множеств типа *R,* определенных разбиением вложенными сечениями, и соответствующий ему разделитель S. Нумерацию назовем согласованной с разбиением,

если все вершины из S пронумерованы после всех вершин, принадлежащих $R \setminus S$ (при условии, что $R \setminus S$ непусто). Если S = R, как это происходит в случае терминальных членов разбиения, то вершины S могут упорядочиваться произвольно. Нумерация, показанная на рис. 4.16 (a), согласована с разбиением, так как вершины S_0 помечены после вершин из $R_g \setminus S_0$. Другое согласованное упорядочение дано на рис. 4.16 (b); в обоих случаях (a) и (b) вершины терминальных членов S_1^1 и S_1^2 можно занумеровывать произвольным образом. Нумерация на рис. 4.16 (c) не является согласованной. Нумерации, получаемые в соответствии с алгоритмом, описанным в предыдущем параграфе, согласованы с разбиением.

Монотонное упорядочение дерева было определено в § 4.2 как такое упорядочение, при котором любая вершина дерева нумеруется прежде своего отца. Монотонное упорядочение дерева сечений порождает нумерацию вершин графа, которая является как совместимой, так и согласованной с разбиением вложенными сечениями, и именно на этом основана работа алгоритма, описанного в § 4.10. Если, кроме того, разбиение вложенными сечениями является минимальным (т. е. каждый разделитель — минимальный), то имеет место следующее важное утверждение: все нумерации, согласованные с рассматриваемым разбиением, эквивалентны в том смысле, что соответствующая переупорядоченная матрица будет иметь тот же объем заполнения в результате гауссова исключения (и потребуется такое же количество арифметических операций). Именно по этой причине мы ограничим рассмотрение только специальным классом нумераций, которые не только согласованы с разбиением, но и совместимы с ним. Так как использование нумераций более общего вида не дает никаких преимуществ, то алгоритм был ориентирован на построение только согласованных и совместимых упорядочений.

Предположим теперь, что для заданной матрицы A получено разбиение ее графа минимальными вложенными сечениями, и соответствующая переупорядоченная матрица A' факторизована в виде $A' = U^TDU$. Важным критерием качества разбиения служит количество Z ненулевых элементов в наддиагональной части матрицы U. Для величины Z справедлива следующая оценка:

$$Z \leqslant \sum |S_i^l| \left(|\mathrm{Adj}\,(R_i^l)| + (|S_i^l| - 1)/2\right), \qquad (4.18)$$

где суммирование распространяется на все разделители и соответствующие множества типа R, а вертикальные черточки используются для обозначения числа элементов множества. Граница достигается, если условие

$$\mathrm{Adj}\,(R_i^l) \subset \mathrm{Adj}\,(R_i^l \setminus S_i^l) \qquad (4.19)$$

выполнено во всех случаях. Указанное условие может нарушаться при использовании алгоритма, описанного в § 4.10, однако на

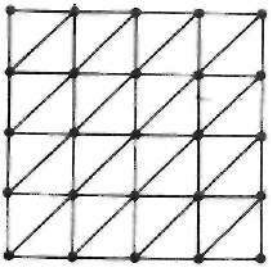


Рис. 4.17. Регулярная сетка, образованная треугольниками с тремя узлами. Этот же рисунок представляет соответствующий плоский граф.

практике это происходит редко, и приведенная выше граница Z для большинства случаев оказывается достаточно точной.

В § 4.2 было введено понятие плоского графа, т. е. графа, который можно изобразить на плоскости так, чтобы никакие два его ребра не пересекались. Для заданного графа можно определить, является ли он плоским (пленарным); алгоритм, выполняющий тест на планарность за время, пропорциональное числу вершин, был предложен Тарьяном [Tarjan, 1971]. С другой стороны, может быть известно, что граф матрицы является плоским по построению. Важным источником разреженных матриц с плоскими графами является метод конечных элементов. В его простейшей версии задача, описываемая дифференциальным уравнением, решается путем разбиения области, в которой задано уравнение, на конечные элементы с узлами в углах и, возможно, с дополнительными узлами на границе элемента или внутри его. Затем конструируется разреженная матрица A, в которой каждая строка и столбец соответствуют узлу сетки, причем $A_{ij} \neq 0$ в том и только том случае, когда узлы i и j принадлежат одному и тому же конечному элементу. Рассмотрим теперь граф, отвечающий матрице A. Существует взаимно однозначное соответствие между вершинами графа и узлами конечно-элементной сетки. Две вершины графа соединяются ребром в том и только том случае, если соответствующие узлы сетки принадлежат одному и тому же конечному элементу. Если задача решается в двух измерениях, то сетка — двумерная, и соответствующий граф оказывается плоским [1]). Рассмотрим этот случай более детально.

Регулярная сетка размера 5×5, составленная из треугольников с тремя узлами, показана на рис. 4.17. Тот же рисунок можно интерпретировать как изображение графа, связанное с конечно-элементной сеткой. Граф является плоским, однако оказывается, что приведенные ниже результаты сохраняют силу и для регулярной сетки, образованной четырехугольниками с четырьмя узлами, которой отвечает неплоский граф.

Рассмотрим разбиение регулярной конечноэлементной сетки размера $k \times k$ при помощи вложенных сечений (см. рис. 4.18). Вершины графа, отвечающие узлам, лежащим на линии сетки, используются в качестве разделителей. Первым разделителем 5 является вертикальная линия сетки. Следующие два разделителя

[1]) В случае если используются треугольные элементы. — *Прим. перев.*

выбираются на горизонтальной линии. Линии выбираются таким образом, чтобы каждый прямоугольник разделялся на части, как можно более близкие друг к другу по количеству узлов. Каждый раз, когда разделитель выбран, его узлы нумеруются в обратном порядке, т. е. узлы разделителя S_0 нумеруются числами от $n = k^2$ (общее число узлов сетки) до $n - k + 1$ и так далее. Процесс продолжается до тех пор, пока не будут пронумерованы все узлы. Если теперь построить матрицу A и факторизовать ее в виде U^TDU, то число Z ненулевых элементов в наддиагональной части матрицы U и количество P операций типа умножения, необходимых для вычисления матрицы U, равны [George, Liu, 1981]

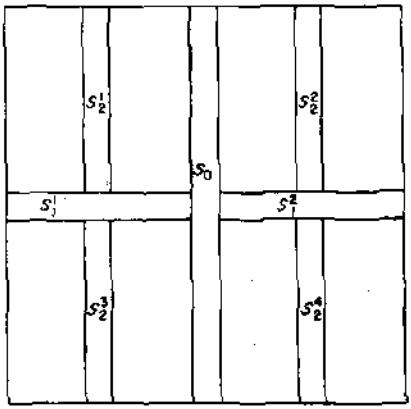


Рис. 4.18. Вложенные сечения регулярной сетки.

$$Z = \frac{31}{8} n \log_2 n + O(n), \qquad (4.20)$$

$$P = \frac{829}{84} n^{3/2} + O(n \log_2 n), \qquad (4.21)$$

где символ «O» означает «величина порядка». Хотя на практике сетка обычно бывает нерегулярной, для оценок объема памяти и числа операций остаются справедливыми формулы того же вида.

4.12. ОБОБЩЕННЫЕ ВЛОЖЕННЫЕ СЕЧЕНИЯ

На этом этапе изложения читателю должно быть ясно, что польза стратегии вложенных сечений зависит от того, насколько малы разделители графа. Существует прямая связь между размером разделителей и эффективностью гауссова исключения [Lipton, Rose, Tarjan, 1979]; разреженное гауссово исключение эффективно для любой матрицы, граф которой имеет «хорошие» разделители. Обратно, эффективность будет низкой при отсутствии «хороших» разделителей. Количественная характеристика качества разделителей может быть дана следующим образом. Рассмотрим граф $G = (V, E)$, имеющий $n = |V|$ вершин, и рассмотрим также всевозможные подграфы графа G. Мы определим границы размера разделителей графа G или любого его подграфа в терминах трех констант: α, β и σ, Именно, предложим,

что G или любой подграф G можно разъединить разделителем S на графы R_x и R_2, такие что

$$|R_1|, |R_2| \leqslant \alpha n, \quad |S| \leqslant \beta n^{\sigma}. \tag{4.22}$$

Это требование и определяет α, β и σ. Константы должны удовлетворять ограничениям $1/2 \leq \alpha < 1$ и $\beta > 0$ (за исключением того случая, когда граф G не имеет разделителей, и тогда $S = G$ и $\beta = \sigma = 1$). Указанные три константы используются для описания классов графов и оценки размеров памяти и времени, необходимых для выполнения гауссова исключения. Приведем сводку основных результатов.

Любая система уравнений, граф которой удовлетворяет условиям (4.22) при $\sigma = 1/2$, может быть упорядочена так, что при гауссовом исключении заполнение составит не более $c_3 n \log_2 n + O(n)$, а число умножений не более $c_7 n^{3/2} + O(n \log_2^2 n)$ [Lipton, Tarjan, Rose, 1977], где

$$c_3 = \beta^2 \left(\frac{1}{2} + \frac{2\sqrt{\alpha}}{1-\sqrt{\alpha}} \right) \Big/ \left(\log_2 \left(\frac{1}{\alpha} \right) \right),$$

$$c_7 = \beta^2 \left(\frac{1}{6} + \frac{\beta\sqrt{\alpha}}{1-\sqrt{\alpha}} \left(2 + \frac{\sqrt{\alpha}}{1+\sqrt{\alpha}} + \frac{4\alpha}{1-\alpha} \right) \right) \Big/ (1 - \alpha^{3/2} - (1-\alpha)^{3/2}). \tag{4.23}$$

Алгоритм для отыскания соответствующего упорядочения был описан в работе [Lipton, Tarjan, Rose, 1979]. Алгоритм основан на рекурсивном рассечении графа, как и тот, который был описан в параграфе 4.10, однако на этот раз время выполнения оказывается зависящим от трудоемкости определения разделителей, удовлетворяющих условию (4.22) при $\sigma = 1/2$. Если требуемые разделители могут быть найдены за время $O(m + n)$, где m — число ребер, а n — число вершин графа, то время работы алгоритма упорядочения составляет $O((m + n) \log n)$.

Одним из классов графов, удовлетворяющих условиям (4.22) при $\sigma = 1/2$, является класс плоских графов, для которого имеем также $\alpha = 2/3$ и $\beta = 2\sqrt{2}$ [Lipton, Tarjan, 1979]. Любой плоский граф можно переупорядочить таким образом, чтобы возникающее при разреженном гауссовом исключении заполнение не превосходило $c_3 n \log n + O(n)$ при числе умножений, не большем $c_7 n^{3/2} + O(n \log^2 n)$, где $c_3 \leq 129$ и $c_7 \leq 4002$. Кроме того, разделители могут быть найдены за время $O(n)$, так что время выполнения алгоритма упорядочения составляет $O(n \log n)$. Алгоритм отыскания разделителей дан в работе [Lipton, Tarjan, 1979].

Двумерные конечно-элементные графы являются почти плоскими и удовлетворяют условию (4.22) при $\sigma = 1/2$ и $\alpha = 2/3$. Если каждый конечный элемент имеет не более k узлов на

Таблица 4.1

Класс графов	Оценка заполнения	Оценка числа умножений	Замечания и ссылки
Любой, такой что $\sigma = \frac{1}{2}$	$c_3 n \log_2 n + O(n)$	$c_7 n^{3/2} + O(n \log^2 n)$	Время упорядочения составляет $O((m+n) \times \log n)$, если разделители можно найти за время $O(m+n)$. Константы c_3 и c_7 определены в (4.23) [Lipton, Rose, Tarjan, 1977].
Плоские графы (в этом случае $\sigma = \frac{1}{2}$, $\alpha = \frac{2}{3}$, $\beta = 2\sqrt{2}$)	$c_3 n \log_2 n + O(n)$	$c_7 n^{3/2} + O(n \log^2 n)$	$c_3 \leqslant 129$, $c_7 \leqslant 4002$. Время упорядочения составляет $O(n \log n)$ [Lipton, Tarjan, 1979; Lipton, Rose, Tarjan, 1979].
Двумерные конечноэлементные графы (в этом случае $\sigma = \frac{1}{2}$, $\alpha = \frac{2}{3}$, $\beta = 4\,[k/2]$)	$O(k^2 n \log n)$	$O(k^3 n^{3/2})$	k — максимальное число узлов на границе элементов. Время упорядочения составляет $O(n \log n)$ [Lipton, Rose, Tarjan, 1979].
Регулярная плоская сетка	$\frac{31}{8} n \log_2 n + O(n)$	$\frac{829}{84} n^{3/2} + O(n \log n)$	[George, Liu, 1981]
Любой такой, что $\sigma > \frac{1}{2}$	$O(n^{2\sigma})$	$O(n^{3\sigma})$	[Lipton, Rose, Tarjan, 1979]
Трехмерные сеточные графы (в этом случае $\sigma = 2/3$)	$O(n^{4/3})$	$O(n^2)$	[Lipton, Rose, Tarjan, 1979]
Любой такой, что $\frac{1}{3} < \sigma < \frac{1}{2}$	$O(n)$	$O(n^{3\sigma})$	[Lipton, Rose, Tarjan, 1979]
Любой такой, что $\sigma = 1/3$	$O(n)$	$O(n \log n)$	[Lipton, Rose, Tarjan, 1979]
Любой такой, что $\sigma < 1/3$	$O(n)$	$O(n)$	[Lipton, Rose, Tarjan, 1979]

границе, то $\beta = 4 \mid k/2 \mid$. Для всякого двумерного конечноэлементного графа существует упорядочение, приводящее к заполнению порядка $O(k^2 n \log n)$ и числу операций умножения порядка $O(k^3 n^{3/2})$. Так как разделители могут быть определены за время $O(n)$, то время выполнения алгоритма упорядочения составит $O(n \log n)$. Указанные оценки могут быть понижены по порядку, если использовать быстрые алгоритмы матричного умножения и факторизации Штрассена [Strassen, 1969] и Банча—Хопкрофта [Bunch, Hopcroft, 1974]. Если требуется решить систему уравнений только для одной правой части, то заполнение может быть уменьшено до $O(n)$ [1]) путем сохранения только части элементов в случае необходимости. Эта процедура была описана в работе [Eisenstat, Schults, Sherman, 1976] для обычного алгоритма вложенных сечений, однако возможно ее обобщение [Lipton, Tarjan, Rose, 1979].

Результаты, приведенные в этом и предыдущем параграфах, а также некоторые другие результаты сведены для наглядности в табл. 4.1.

4.13. ПАРАЛЛЕЛЬНЫЕ СЕЧЕНИЯ ДЛЯ КОНЕЧНОЭЛЕМЕНТНЫХ ЗАДАЧ

Эвристический метод, обсуждаемый в этом параграфе, был разработан А. Джорджем [George, 1980] первоначально для двумерных конечноэлементных задач. Он сочетает лучшие черты нескольких подходов, таких, как профильный метод Дженнингса, метод древовидного разбиения и метод сечений, и все же он достаточно прост и допускает несложную реализацию. Обобщение этого метода на трехмерные конечноэлементные задачи также может быть легко получено. Описанный ниже алгоритм может применяться в зависимости от желания пользователя для минимизации либо памяти, либо числа арифметических операций при факторизации, либо числа операций при обратной подстановке. Возможна даже минимизация *общей* стоимости выполнения алгоритма путем учета сравнительной стоимости использования каждого из ресурсов ЭВМ и количества правых частей, для которых требуется решить систему, причем это можно делать *автоматически.* Если размер задачи не слишком велик и преследуется цель минимизации объема памяти, то алгоритм требует меньше памяти, чем любой другой метод [2]); этот факт тесно связан с тем, что метод параллельных сечений требует небольшого объема накладной памяти. Конечно, предстоит еще сравнить его с двумя широко используемыми методами: ленточным методом и схемой Джен-

[1]) Точнее, до этой величины уменьшаются затраты памяти при реализации алгоритма. — *Прим. перев.*

[2]) Из перечисленных в предыдущих параграфах. — ***Прим. перев.***

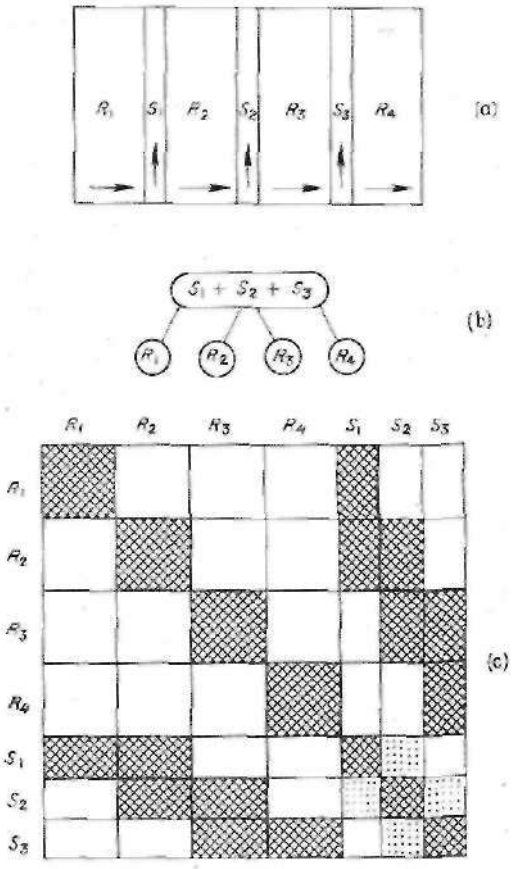


Рис. 4.19. Параллельные сечения регулярной сетки.

нингса [1]). По нашему мнению, параллельные сечения являются следующим кандидатом для принятия в качестве промышленного стандарта математического обеспечения конечноэлементных вычислений.

Идея метода проиллюстрирована на рис. 4.19 (а), где прямоугольниками представлены множества узлов двумерной конечноэлементной сетки. Выбирая σ разделителей небольшого размера (на рисунке $\sigma = 3$), образованных линиями узлов сетки, мы получаем разбиение сетки на $\sigma + 1$ блоков R_1, R_2, ... сравнимого размера. Если собрать все разделители з еще один блок, то возникает древовидное разбиение, показанное на рис. 4.19 (Ь).

[1]) Такое сравнение приведено в конце параграфа. Схемой Дженнингса автор называет профильный метод. — *Прим, перед.*

Достоинства древовидного разбиения с точки зрения уменьшения заполнения и числа операций обсуждались в § 4.9. Занумеруем теперь узлы каждого множества типа R последовательно, слева направо вдоль горизонтальных линий, начиная с нижнего левого угла, как это показано на рис. 4.19 (а) стрелками. После того как занумерованы все множества типа R, последовательно нумеруются разделители в порядке, указанном стрелками. Полученная нумерация отвечает монотонному упорядочению дерева. Матрица, связанная с конечноэлементной сеткой, разбивается на блоки, как это показано на рис. 4.19 (с), где заштрихованы области, которые могут содержать ненулевые элементы. При выполнении гауссова исключения над этой матрицей заполнение будет возникать только внутри заштрихованных областей, а также внутри областей матрицы, помеченных точками. Кроме того, следует учесть, что заштрихованные области заполнены ненулевыми элементами лишь частично. В нашем примере четыре первых диагональных блока имеют ленточную структуру. Это обстоятельство удается выгодно использовать, если применить неявную схему хранения для матриц, разбитых на блоки, описанную в § 1.11.

Остается определить количество разделителей σ. Величина σ является основным параметром метода. Чтобы найти ее значение, необходимо иметь явные выражения требуемого объема памяти и числа операций в виде функций, зависящих от σ. Любая из этих функций или их комбинация, описывающая общую стоимость выполнения алгоритма на данной ЭВМ, может затем быть минимизирована за счет соответствующего выбора σ. Так как эти функции дают лишь оценки сверху и, кроме того, мало меняются в окрестности точки минимума, то обычно оказывается достаточно взять в качестве значения a ближайшее целое к точке минимума. Анализ [George, Liu, 1981] был выполнен для регулярной сетки размеров $m \times l$ при $m \leq l$, подобной той, которая изображена на рис. 4.17. Если в качестве разделителей выбраны линии сетки, члены разбиения содержат примерно равное количество узлов и используется неявная схема хранения, то общий объем памяти составит асимптотически

$$S \approx \frac{ml^3}{\sigma} + \frac{3\sigma m^2}{2}. \qquad (4.24)$$

Указанное выражение достигает своего минимального значения

$$S_{\min} = \sqrt{6}\, m^{3/2} l + O(ml) \text{ при } \sigma = \sqrt{2}\, l/\sqrt{3m};$$

более тщательный анализ и опыт расчетов показывают, что лучшим приближением для значения σ, минимизирующего S, является $\sigma = \sqrt{2}l/\sqrt{3m + 13}$.

Общее число операций, требуемых при выполнении факторизации, асимптотически равно

$$\Theta_F \approx \frac{ml^3}{2\sigma^2} + \frac{7\sigma m^3}{6} + \frac{2m^2l^2}{\sigma}, \qquad (4.25)$$

и минимум этого выражения достигается при больших m и l для значения σ, примерно равного $\sigma = \sqrt{12}l/\sqrt{7m}$. Само значение минимума равно

$$\Theta_{F,\min} = \sqrt{28/3}\, m^{5/2}l + O(m^2l).$$

Общее число операций, требуемое для решения системы при условии, что факторизация уже получена, асимптотически равно

$$\Theta_S \approx \frac{4ml^2}{\sigma} + 3\sigma m^2. \qquad (4.26)$$

Указанная величина достигает своего минимального значения

$$\Theta_{S,\min} = 4\sqrt{3}\, m^{3/2}l + O(ml) \text{ при } \sigma = 2l/\sqrt{3m}.$$

Сравним теперь оценки эффективности метода параллельных сечений с оценками для методов, основанных на использовании ленточной или профильной схем совместно с постолбцовой нумерацией узлов сетки. Для ленточной или профильной схемы память, число операций для факторизации и число операций для решения составят соответственно примерно m^2l, $m^3l/2$ и $2m^2l$. Таким образом, если использовать параллельные сечения с оптимизацией по памяти, то требуется в $\sqrt{m/6}$ раз меньше памяти. Если использовать параллельные сечения с оптимизацией по числу операций для факторизации, то требуется в $\sqrt{m/37}$ раз меньше операций для факторизации. Если использовать параллельные сечения с оптимизацией по числу операций для решения, то требуется в $\sqrt{m/12}$ раз меньше операций для решения системы. Конечно, одновременная оптимизация по всем трем параметрам невозможна. Поскольку на практике значение m обычно заключено в пределах от 20 до 100 (напомним, что $l \geq m$), то можно заключить, что использование параллельных сечений может дать значительный выигрыш в объеме памяти или во времени решения серии задач с одной и той же матрицей в большинстве практических случаев, и что для задач большого размера можно также получить выигрыш по числу операций при выполнении факторизации. Читатель может оценить возможное понижение общей стоимости решения своей частной задачи на интересующей его ЭВМ.

Изложенные идеи могут быть перенесены также на случай нерегулярных конечноэлементных сеток с использованием аппарата структур уровней; такой подход позволяет заложить основу для построения автоматического алгоритма. Предполагается, что

структура уровней с корнем в некотором псевдопериферийном узле окажется длинной и узкой. Таким образом, число уровней играет роль величины /, а вместо m теперь выступает средняя ширина структуры. Затем можно вычислить значение σ, исходя из оптимизации оценки памяти, либо оценки числа операций при решении системы или любого другого желаемого критерия; при этом структура уровней считается приблизительно подобной регулярной сетке. Разделители затем выбираются среди уровней структуры, и узлы нумеруются с использованием обратного алгоритма Катхилл—Макки (см. § 4.6) внутри каждой связной компоненты; целью при этом ставится уменьшение профиля каждого диагонального блока матрицы. Детальное описание алгоритма выглядит следующим образом [George,- Liu, 1981]:

Шаг I (построение структуры уровней). Для заданной двумерной конечноэлементной сетки с N узлами найти псевдопериферийную вершину соответствующего графа $G(V, E)$ и построить структуру уровней с корнем в этой вершине. Пусть $L_0, L_1, \ldots L_l$ — уровни полученной структуры.

Шаг 2 (оценка расстояния δ между разделителями). Вычислить $m = N/(l + 1)$, оценить число разделителей σ, оптимизирующее требуемый критерий, и оценить расстояние между разделителями $\delta = (l + 1)/(\sigma + 1)$. Заметим, что теперь $l + 1$ играет роль параметра l, введенного в регулярном случае.

Шаг 3 (предельный случай). Если $\delta < l/2$ и $N > 50$, то перейти к шагу 4. В противном случае считать применение метода параллельных сечений нецелесообразным, положить $p = 0$ и перейти к шагу 6.

Шаг 4 (отыскание разделителей). Положить $i = 1$ и $S = \emptyset$.

(4.1) Положить $j = [i\delta - \frac{1}{2}]$ (т. е. j — ближайшее целое к $i\delta$). Если $j \geq l$, то перейти к шагу 5. В противном случае продолжать вычисления.

(4.2) Пусть S_i — множество всех вершин уровня L_j, смежных с вершинами из L_{j+1}.

(4.3) Положить $S \leftarrow S \cup S_i$.

(4.4) Положить $i \leftarrow i + 1$ и перейти к шагу (4.1).

Шаг 5 (определить блоки). Найти p связных компонент $R_1, R_2, \ldots, R_p$, образовавшихся после удаления из графа вершин множества S. Заметим, что значение p может отличаться от $\sigma + 1$, так как σ — не целое число, и, кроме того, получившееся разбиение графа может иметь большее число блоков.

Шаг 6 (внутренняя нумерация). Занумеровать каждое множество R_k, $k = 1, 2, \ldots, \rho$, используя обратный алгоритм Кат-

хилл—Макки для каждого из частичных графов. Наконец, занумеровать вершины множества S произвольным образом.

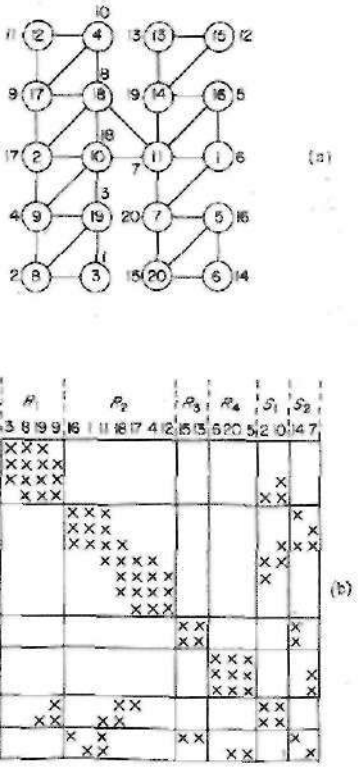


Рис. 4.20. Применение метода параллельных сечений к графу, изображенному на рис. 4.2. Получающаяся нумерация графа показана на рис. (a), а блочное разбиение матрицы — на рис. (b).

На шаге 6 можно рекомендовать использование улучшенной версии обратного алгоритма Катхилл—Макки [Brown, Wait, 1981], так как это обычно приводит к уменьшению осцилляции в упорядочении узлов сетки.

В качестве примера применения метода параллельных сечений рассмотрим граф, показанный на рис. 4.2 (a), и структуру уровней с корнем в периферийной вершине 8, показанную на рис. 4.2 (b). Так как $N = 20$ и $l = 6$, то получаем $m = 20/7$. Если нам требуется оптимизировать объем памяти, то нужно взять $\sigma = \sqrt{\bar{a}} + 1)/\sqrt{3m} \approx 13$ 2.13 и $\delta = (l + 1)/(\sigma + 1) \approx$

≈ 2.23. Уровни, служащие в качестве разделителей, выбираются на шаге 4; это — уровень 2 и уровень 4. Таким образом,

$$S_1 = \{2, 10\},$$
$$S_2 = \{14, 7\}.$$

Заметим, что вершины 14 и 7 являются единственными вершинами уровня 4, которые смежны с вершинами из L_5. Возникающие блоки образованы следующими множествами вершин:

$$R_1 = \{8, 9, 19, 3\},$$
$$R_2 = \{17, 18, 11, 12, 4, 16, 1\},$$
$$R_3 = \{13, 15\},$$
$$R_4 = \{5, 20, 6\}.$$

Таким образом, для этого примера p — 4. Каждое из четырех множеств типа R локально упорядочивается с использованием обратного алгоритма Катхилл—Макки. Два множества типа S затем нумеруются произвольным образом. Окончательная нумерация показана на рис. 4.20 (a), а структура соответствующей переупорядоченной матрицы — на рис. 4.20 (b). Легко понять, каковы будут в этом случае блочное разбиение и профильная структура диагональных блоков.

Программы на Фортране, реализующие этот алгоритм, были опубликованы в монографии Джорджа и Лю [George, Liu, 1981].

4.14. УПОРЯДОЧЕНИЯ ДЛЯ МЕТОДА КОНЕЧНЫХ ЭЛЕМЕНТОВ

Ниже будет дано краткое изложение тех численных аспектов метода конечных элементов, которые представляют интерес для нас. Метод используется для решения задач, описываемых дифференциальным уравнением в заданной области. Область разбивается на подобласти простой формы, называемые *конечными элементами,* у которых могут быть общие участки границы, но нет общих внутренних точек. Если область не ограничена, то можно применить *бесконечные элементы* [Bettess, 1977; Pissanetzky, 1982]. На границах элементов берутся *узлы*; дополнительные узлы могут быть взяты и внутри элементов. Если узел находится на границе двух элементов, будем говорить, что он принадлежит обоим элементам, а если узел находится в углу, — что он принадлежит всем элементам, заходящим в этот угол. Элементы и узлы нумеруются произвольным образом, и в совокупности множество элементов и узлов называется *сеткой.* На рис. 4.17 изображен двумерный пример, в котором элементами являются треугольники с узлами в углах. При применении метода формируется разреженная линейная система

$$Ax = b \qquad (4.27)$$

порядка n, где n — число узлов сетки. Наше обсуждение будет ограничено случаем, когда с каждым узлом ассоциированы только одна неизвестная и только одно уравнение, хотя в более общем случае на узел может приходиться более, чем одна переменная, и η может быть больше, чем число узлов.

Если сетка составлена из m элементов, то A вычисляют как сумму m *элементных матриц* $A^{(e)}$. Каждая матрица $A^{(e)}$ квадратная порядка n, зависит от конкретной решаемой задачи и имеет то важное свойство, что $a_{ij}^{(e)}$ может отличаться от нуля только, если узлы i и j принадлежат элементу e. Хотя в некоторых случаях $a_{ij}^{(e)} = 0$ и тогда, когда i и j принадлежат e, это бывает нечасто; для простоты мы предположим, что $a_{ij}^{(e)}$ не равен нулю в том и только том случае, когда узлы i и j принадлежат элементу e. Следовательно:

$$A = \sum_{e=1}^{m} A^{(e)}, \tag{4.28}$$

и a_{ij} не равен нулю в том и только том случае, когда существует конечный элемент, которому принадлежат узлы i и j (как обычно, мы пренебрегаем возможностью взаимного уничтожения). Более формально: пусть E — множество конечных элементов, E_i — множество элементов, содержащих узел i. $E_i \neq \emptyset$, поскольку всякий узел принадлежит хотя бы одному элементу, и $E_i \subseteq E$; во всех практических случаях E_i — собственное подмножество E. Тогда a_{ij} не равен нулю тогда и только тогда, когда $E_i \cap E_j \neq \emptyset$.

Часто говорят, что матрицы $A^{(e)}$ в уравнении (4.28) *собирают,* чтобы сформировать A. Для краткости мы будем также говорить, что «конечный элемент e собран», если элементная матрица $A^{(e)}$ вычислена и прибавлена к сумме в уравнении (4.28).

Пусть β — максимум модуля разности номеров (в принятой нумерации) всех пар узлов, принадлежащих одному и тому же элементу сетки, т. е.

$$\beta = \max_{e,\, k,\, l} \{ |k - l| \mid k \text{ и } l \text{ — узлы } e \}. \tag{4.29}$$

Тогда $a_{ij} = 0$, если $|i - j| > \beta$. Это значит, что A — ленточная матрица с полушириной ленты β. Вследствие этого очень популярны ленточные схемы для решения уравнения (4.27). Также очень популярен простой, метод уменьшения ширины ленты; он состоит в нумерации узлов сетки таким образом, чтобы β было мало. На рис. 4.21 приведен пример, где показаны три различных нумерации узлов простой сетки. Кроме того, показан портрет матрицы, соответствующей одной из нумераций.

Метод конечных элементов так важен для практики и настолько широко используется, что для эффективного решения (4.27)

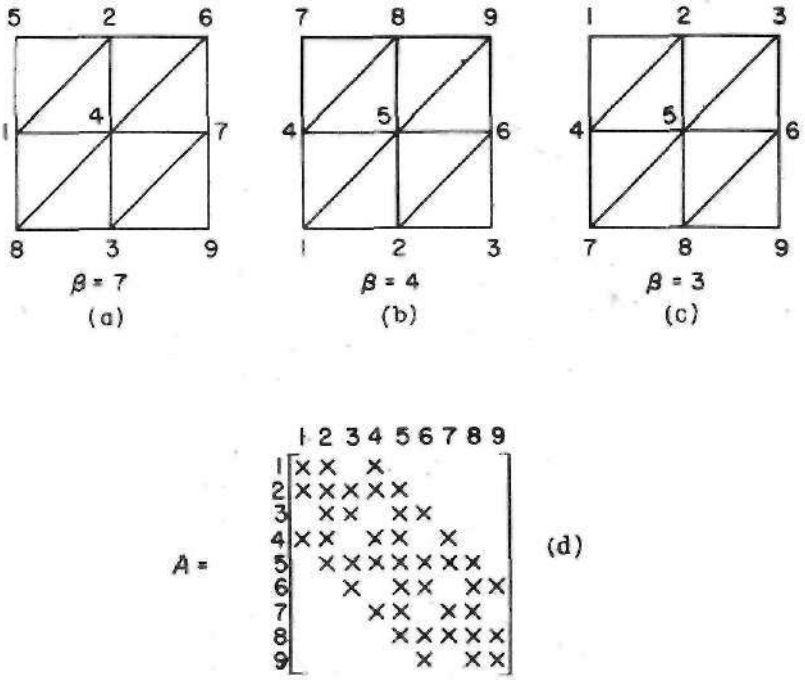


Рис. 4.21. Три различные нумерации узлов простой сетки. Портрет матрицы, показанный на рис. (d), имеет полуширину ленты β = 3 и соответствует нумерации (c).

помимо классических приемов типа ленточных матриц были разработаны специальные процедуры упорядочения. Среди этих процедур — методы параллельных и вложенных сечений, хотя позднее метод вложенных сечений был обобщен, как это объяснено в § 4.12; аналогичным образом можно обобщить и метод параллельных сечений. Как в ленточных методах, так и в методах сечений исходят из предположения, что до начала исключения в A суммирование, указываемое уравнением (4.28), завершено, и A полностью собрана. Теперь мы обсудим *фронтальный метод* [Irons, 1970], в котором делается обратное предположение: исключение начинается прежде, чем будет закончено суммирование в (4.28). Наше обсуждение (но не сам метод!) будет ограничено распространенным случаем, когда A — симметричная и положительно определенная. Поэтому при любом порядке исключения требование численной устойчивости, рассмотренное в гл. 3, будет удовлетворено, если главные элементы выбирать на главной диагонали.

Предположим, что на некотором этапе A частично собрана, и пусть C ($C \subset E$) — множество собранных конечных элементов (т. е. конечных элементов, элементные матрицы которых уже собраны). Заметим, что сумма в уравнении (4.28) может вычисляться в любом порядке, не обязательно в обычном порядке, определяемом индексом e; таким образом, Сможет быть, вообще говоря, произвольным подмножеством E. На данном этапе n узлов сетки могут быть разбиты на три множества:

G — множество полностью собранных узлов: $G = \{i \mid E_i \subseteq C\}$,

F — множество частично собранных узлов или *фронт:*

$$F = \{i \mid E_i \cap C \neq \emptyset, \quad E_i \cap (E - C) \neq \emptyset\},$$

H — множество не собиравшихся узлов: $H = \{i \mid E_i \subseteq E - C\}$.

Ясно, что если элемент e не входит в C, то $a_{ij}^{(e)} = 0$, если либо i, либо j, либо оба принадлежат G. Это значит, что все строки и столбцы A, ассоциированные с узлами из G, сейчас полностью собраны. Заметим еще, что $a_{ij} = 0$, если $i \in G$, а $j \in H$, потому что $E_i \cap E_j = \emptyset$. Вид частично собранной матрицы A на данном этапе показан на рис. 4.22. Понятно, что все неизвестные, соответствующие узлам из G, можно исключить и заполнение будет происходить только в области, отмеченной надписью «частично собрано». Как только это сделано, собираются новые элементы из E — C, переопределяются множества G, F и H и исключаются новые переменные. Фронт движется по сетке. Процесс продолжается, пока A не будет полностью собрана и разложена.

В практических реализациях фронтального метода исключения проводятся, как *только* они становятся возможными; порядок, в котором собираются элементы, берется такой, чтобы фронт оставался малым. Все узлы фронта лежат на границе между областью, покрытой элементами из множества C, и всеми прочими элементами. Если на каждом этапе эта область связная, то граница является непрерывной линией, а метод называется *однофронтальным.* Обобщение этой идеи не встречает затруднений: метод называют *многофронтальным,* если элементы из C покрывают две или более отдельных областей с независимыми границами.

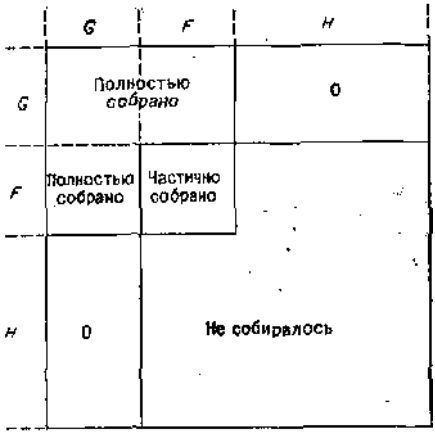


Рис. 4.22. Иллюстрация к понятию фронтального метода. Частично собранная матрица A.

Каждую такую область иногда называют *обобщенным, элементом* или *суперэлементом* [Speelpenning, 1978]. Выбор областей может быть продиктован техническими соображениями, например, они могут соответствовать частям физического тела. Этот подход известен под названием *«разбиение на подструктурьру* и особенно удобен, когда несколько таких частей имеют одинаковую структуру и для всех годится один и тот же анализ, или когда части очень сложны, и каждая конструируется своей группой инженеров. Процесс разбиения на подструктуры можно проводить на любую глубину, и для представления таких вложенных структур подходящими структурами данных являются деревья. Очень хорошее обсуждение этого и смежных вопросов можно найти в [Reid, 1981].

Другой возможный способ действий, имеющий целью уменьшение ширины фронта и хорошо приспособленный к автоматической обработке произвольных разреженных матриц,состоит в том, чтобы построить для ассоциированного с матрицей графа структуру уровней с корнем в псевдопериферийном узле, а затем использовать уровни как последовательные фронты. Поскольку каждый уровень является разделителем графа, необходимое условие для малости фронта — это наличие у графа хороших разделителей. Согласно § 4.12, это условие выполнено для планарных и почти планарных графов.

В сущности, фронтальный метод не пытается сократить заполнение; он уменьшает количество информации, которую необходимо единовременно держать в оперативной памяти машины. Если переменные исключаются, как только это станет возможно, соответствующие строки A могут быть немедленно записаны на внешнюю память, а в оперативной остается только подматрица, обозначенная на рис. 4.22 надписью «частично собрано». Если фронт мал, то мала и эта подматрица, и обычно ее обрабатывают как плотную матрицу. Получающаяся в результате программа, использующая технику плотных матриц, легко поддается параллельной обработке [Duff, 1980b], хотя на конвейерных машинах хорошо выполняются и разреженные программы [Coleman, Kushner, 1982]. Окончательная факторизованная форма A, накапливаемая в периферийном устройстве, может оказаться больше, чем если бы мы действительно пытались уменьшить заполнение.

В гл. 5 будет рассмотрено обобщение фронтального метода на случай, когда A не является симметричной и положительно определенной.

В некоторых случаях возможна иная формулировка метода конечных элементов. До сих пор мы неявно предполагали, что сетка собрана и что узел может принадлежать нескольким элементам. Теперь мы будем исходить из обратного допущения: у каждого элемента свой набор узлов, но множество элементов

разобрано, так что всякий узел принадлежит единственному элементу. Как и прежде, с каждым узлом ассоциировано одно неизвестное и одно уравнение; однако, вводится еще система добавочных связей, чтобы вынудить неизвестные, сопоставленные совпадающим узлам, иметь одинаковые значения. Для любого узла i разобранной сетки $E_i = \{e\}$, где e — единственный элемент, содержащий узел i, так что a_{ij} не равен нулю только тогда, когда i и j — узлы одного и того же элемента. Это значит, что если при нумерации узлам каждого элемента присваивать последовательные номера (совместимая нумерация), то ненулевые элементы матрицы группируются в квадратные диагональные блоки порядка, равного числу узлов соответствующего элемента. В результате получится блочно-диагональная матрица A, которую удобно хранить, вводя вектор указателей действительных позиций диагональных блоков; сами блоки могут находиться где угодно в памяти. В линейных задачах метода конечных элементов часто бывает, что многие блоки одинаковы, потому что отвечают одинаковым элементам. Достаточно хранить только одну копию одинаковых блоков, и так как даже для очень больших задач обычно требуется не более 10—20 элементов с различными размерами или типами, то A можно представить посредством вектора указателей длины, равной числу элементов сетки, и нескольких блоков с размерами, скажем 3×3 или 8×8. Далее, поскольку треугольное разложение порождает заполнение только внутри блоков и одинаковые блоки имеют одинаковое разложение, то каждый блок может быть факторизован независимо и заменен в памяти соответствующим треугольным множителем. Этим путем можно достигнуть экономии памяти и вычислительной работы.

Уравнения связей принимают вид

$$C^T x = 0, \tag{4.30}$$

и полная линейная система получается с помощью метода множителей Лагранжа (см., например, [Gallagher, 1975] [1])):

$$\left[\begin{array}{c|c} A & C \\ \hline C^T & 0 \end{array}\right] \left[\begin{array}{c} x \\ \hline \lambda \end{array}\right] = \left[\begin{array}{c} b \\ \hline 0 \end{array}\right]. \tag{4.31}$$

Здесь λ — вектор множителей Лагранжа. Эта система называется *диакоптической*. Ее матрица симметрична, но незнакоопределена. Для решения системы (4.31) можно применить методы, обсуждаемые в § 4.17. В качестве иллюстрации рассмотрим пример разобранной сетки, показанной на рис. 4.23. Связями будут;

[1]) Или Алексеев В. М., Тихомиров В. М., Фомин С. В. Оптимальное управление. — М.: Наука, 1979.

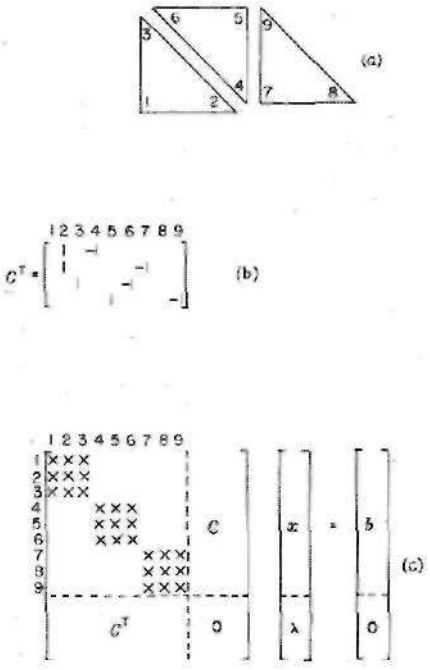


Рис. 4.23. Разобранная конечно-элементная сетка, матрица связей и диакоптическая система.

$x_2 = x_4$, $x_2 = x_7$, $x_3 = x_6$, $x_5 = x_9$,. Их запись в форме (4.30) имеет матрицу C^T, указанную в части (b) рисунка. Соответствующая линейная система находится в части (c). Хороший обзор этой тематики и ряд полезных ссылок содержится в [McDonald, Wexler, 1980]. *Окаймленные системы* формы, аналогичной (4.31), встречаются во многих приложениях (см., например, [Duff, 1977, p. 520]). Еще один возможный подход — метод разреженной таблицы [Hachtel, 1976].

4.15. ПОИСК В ГЛУБИНУ НА НЕОРИЕНТИРОВАННОМ ГРАФЕ

В §4.3 мы описали поиск как процедуру, посредством которой мы в некотором порядке обходим вершины и ребра графа, помечая вершины при их посещении и разбивая ребра на классы. Мы отметили также, что поиск используется во многих алгоритмах для разреженных матриц и что руководствуются различными правилами для указания на каждом шаге, какую вершину посетить следующей или какое ребро исследовать. В этом параграфе мы рассмотрим *поиск в глубину,* при котором граф обходят, пытаясь по возможности придерживаться последовательности вершина—ребро—вершина—ребро... Более подробно: пусть $G = (V, E)$ — неориентированный связный граф. В начале поиска ни одно из ребер не исследовано и ни одна вершина не посещена. Мы начинаем с произвольно выбранной вершины s и рассматриваем ее как первую текущую вершину и как первую посещенную вершину. Пусть V_0 и E_e — соответственно множество посещенных вершин и множество исследованных ребер [1]). Поначалу полагаем $E_e \leftarrow \emptyset$, $V_v \leftarrow \emptyset$. Затем на каждом шаге, исходя из текущей вершины, скажем $v \in V_v$, ищем неисследованное ребро,

[1]) Относительно обозначений: V_v — visited vertices, E_e — explored edges. — *Прим, перев.*

исходящее из v, например $(v, w) \notin E_e$. Возможны следующие случаи:

(1) w не посещалась раньше, т. е. $w \notin V_v$. Добавляем (v, w) к множеству исследованных ребер, т. е. $E_e \leftarrow E_e \cup (v, w)$, а w — к множеству посещенных вершин, т. е. $V_v \leftarrow V_v \cup w$. Вершине w присваивается очередной номер, и она становится текущей вершиной. Поиск продолжается исходя из w. Ребро (v, w) называется *древесным*.

(2) w посещалась раньше. Добавляем (v, w) к E_e и продолжаем поиск исходя из v, которая остается текущей вершиной. Ребро (v, w) называется *обратным*.

(3) нет неисследованных ребер, исходящих из v. В этом случае мы *возвращаемся,* т. е. переходим к предыдущей вершине нашего упорядочения, которая становится новой текущей вершиной, и возобновляем поиск.

Эта процедура повторяется, пока не будут посещены все вершины и исследованы все ребра. В конце поиска вершины упорядочены, а ребра разделены на древесные и обратные. Важно здесь то, что при удалении обратных ребер мы получим дерево T; поскольку T содержит все вершины G, это дерево является остовным. Таким образом, если G имеет $|V|$ вершин и $|E|$ ребер, то древесных ребер будет $|V| - 1$, а обратных $|E| - |V| + 1$. Кроме того, упорядочение, осуществляемое посредством поиска в глубину, — это обращенное монотонное упорядочение остовного дерева T. Алгоритм, реализующий поиск в глубину за время $O(|V|, |E|)$ с использованием такой же по порядку памяти, описан в [Tarjan, 1972].

Рассмотрим в качестве примера применение поиска в глубину к связному неориентированному графу на рис. 4.24 (а). Выберем 9 как начальную вершину; видим, что одним из ребер, исходящих

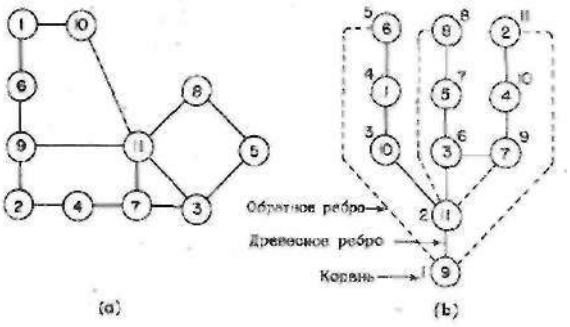


Рис. 4.24. Поиск в глубину на неориентированном связном графе рисунка 4.1. В части (b) ребра остовного дерева указаны сплошными линиями, обратные ребра — пунктирными линиями.

из 9, будет (9, 11). Исследуя вершину 11, возьмем, например, вершину 10, затем 1 и, наконец, 6. Все до сих пор исследованные ребра являются древесными и показаны на рис. 4.24 (b) сплошными линиями. Теперь текущей будет вершина 6, связанная только с вершиной 9, которая уже посещалась. Следовательно, ребро (6, 9) — обратное, что указано пунктирной линией, а 6 остается текущей вершиной. Поскольку исходящих из нее [1]) ребер больше нет, мы возвращаемся к вершине 11, где находим теперь (например) ребро (11, 3), затем (3, 5) и, наконец, (5, 8); все эти ребра — древесные. В вершине 8 мы обнаруживаем обратное ребро (8, 11) и возвращаемся к вершине 3. Далее находим древесные ребра (3, 7), (7, 4) и (4, 2) и обратное ребро (2, 9). Возвращаясь к 7, видим обратное ребро (7, 11). Результат показан на рис. 4.24 (b). Подграф, составленный из древесных ребер, указываемых сплошными линиями, есть остовное дерево. Упорядочение вершин задают числа, поставленные рядом с кружками. Разумеется, это упорядочение не единственно; читатель может в качестве упражнения найти другие.

Изучим более детально свойства поиска в глубину. Легко видеть, что обратное ребро всегда связывает потомка и предка. Действительно, пусть (v, w) — обратное ребро графа G; для определенности предположим, что при поиске вершина v была помечена раньше, чем w. Рассмотрим в T поддерево T_v с корнем в v. Мы входим в T_v при первом посещении v. Прежде чем покинуть T_v при возвращении, мы исследуем все ребра, инцидентные v. Ребро (v, w) инцидентно v, но не является древесным; оно не было выбрано, когда мы находились в v. Оно было выбрано тогда, когда мы перешли в w, но прежде, чем мы покинули T_v. Следовательно, w принадлежит T_v и представляет собой потомка v.

Итак, обратное ребро всегда соединяет вершину с одним из ее предков в остовном дереве. Древесные ребра также связывают вершину с ее предками в T. Это означает, что всякий путь между произвольными двумя вершинами v и w графа G необходимо должен содержать хотя бы одного общего предка v и w; удаление всех общих предков сделает v и w взаимно недостижимыми. Другими словами, ствол остовного дерева является разделителем G. Его устранение разбивает T на два или более несвязанных поддеревьев; их стволы в свою очередь суть разделители соответствующих подграфов. Это наблюдение приводит к *обратному глубинному упорядочению* [2]) для гауссова исключения. Из обсуждения в § 4.7 мы знаем, что если вершины разделителя занумерованы последними, то в верхнем множителе U появится целый блок нулей. Остовное дерево, получаемое путем поиска в глубину,

[1]) И неисследованных. — *Прим. перев.*

[2]) В оригинале — reverse depth-first ordering. — *Прим. перев.*

а тем самым число и размеры разделителей зависят от выбора начальной вершины и от стратегии, определяющей выбор очередного исследуемого ребра среди всех ребер, исходящих из данной вершины. Мы хотели бы, чтобы дерево и поддеревья имели короткие стволы и много ветвей сравнимой величины. Можно надеяться достигнуть этого следующим образом. Начинаем с вершины максимальной степени и присваиваем ей номер $\eta = |V|$. Затем проводим поиск в глубину и нумеруем вершины в ходе посещений, но в обратном порядке. Пусть на некотором шаге u — текущая вершина, V_v — множество посещенных вершин. Стратегия *короткого обратного ребра* такова: исследуется ребро, ведущее к вершине w, степень которой максимальна в множестве Adj $(u) - V_v$. Если таких вершин несколько, выбирается та, для которой максимально число $|\text{Adj}\ (w) \cap V_v|$. Определение «короткое» означает, что для обратного ребра (x, y) расстояние между x и y в остовном дереве мало. Данный метод направлен на порождение многих коротких обратных ребер и последовательную нумерацию смежных вершин; при этом получается кусочно-ленточная структура переупорядоченной матрицы с узкими ленточными участками, заключающими в себе заполнение. Стратегия *длинного обратного ребра* состоит в следующем: исследуется ребро, ведущее к вершине w из Adj $(u) \leftarrow V_v$, для которой максимально число $|\text{Adj}\ (w) - V_v|$. Если таких вершин несколько, выбирается та, для которой минимально число $|\text{Adj}\ (w) \cap \cap V_v|$. Этот метод имеет тенденцию к порождению длинных обратных ребер и лучших разделителей.

Применим в качестве примера эти методы к графу на рис. 4.2(а). Рассмотрим стратегию короткого обратного ребра. Начинаем с вершины 11, имеющей степень 6. В множестве Adj (11) вершины 10 и 18 обе имеют максимальную степень 5, и для обеих $|\text{Adj}\ (w) \cap \cap V_v| = 1$, поскольку $V_v = \{11\}$. Выберем для определенности вершину 10 и исследуем ребро (11, 10). Теперь следует посетить вершину 18, так как ее степень равна 5. После посещения вершины 18 кандидатами будут вершины 2 и 17; степень обеих — 4. Мы выбираем вершину 2, потому что она смежна с двумя посещенными вершинами, тогда как 17 — только с вершиной 18. Следующей мы посещаем вершину 9 [1]); выбор между вершинами 8 и 19 разрешается в пользу 19, которая смежна с двумя посещенными вершинами: 9 и 10 [2]). Читатель может самостоятельно продолжить поиск и убедиться, что будут получены остовное дерево и обратное глубинное упорядочение, показанные на рис. 4.25(а). Сразу видны разделители (11), (10, 18, 2) и (14).

[1]) Здесь также был произвол в выборе, поскольку в смысле стратегии короткого обратного ребра вершины 9 и 17 равноправны. — *Прим. перев.*

[2]) В действительности объяснение этого выбора проще: степень вершины 19 равна 4, в то время как степень вершины 8 — только 3. — *Прим. перев.*

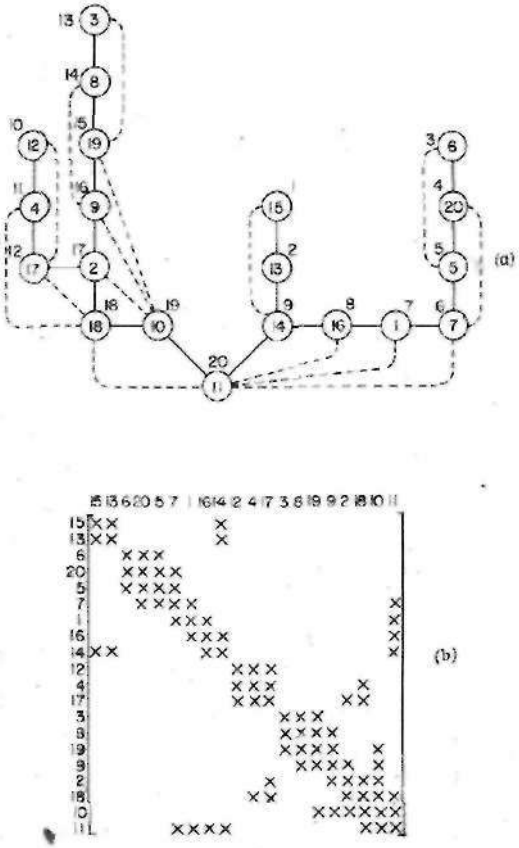


Рис. 4.25. Обратное глубинное упорядочение при стратегии короткого обратного ребра для графа рис. 4.2 (a).

Соответствующая переупорядоченная матрица приведена на рис. 4.25 (b). Выполняя для такой матрицы гауссово исключение, мы не порождаем никакого заполнения, причем этот результат достигается путем очень небольшой вычислительной работы. Причина, почему для графа на рис. 4.2 (a) существует упорядочение без заполнения, — в том, что этот граф триангулирован [Rose, 1970]; см. § 4.16.

Теперь рассмотрим применение к тому же графу стратегии длинного обратного ребра. Снова начинаем с вершины 11. Следующими кандидатами будут вершины 10 и 18; степени обеих = 5. Выберем для определенности вершину 10. В этот момент V_v = = {11, 10}, и каждая из вершин 18, 2, 9, 19 имеет три ребра,

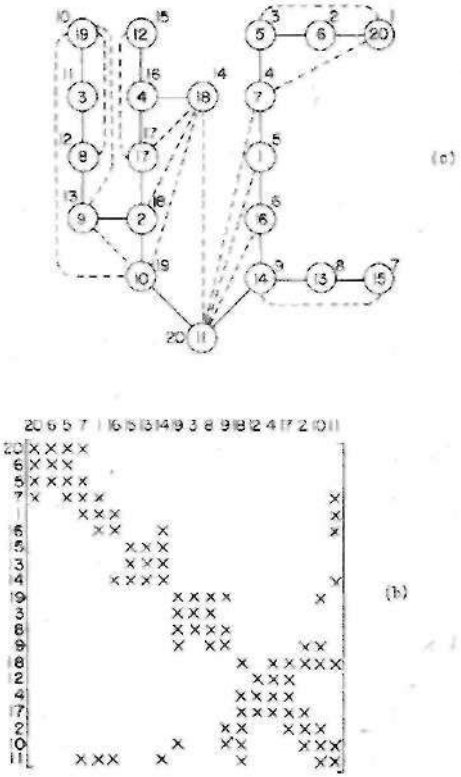


Рис. 4.26. Обратное глубинное упорядочение при стратегии длинного обратного ребра для графа рис. 4.2 (а).

ведущих к вершинам не из V_v. Вершину 18 удаляем из списка кандидатов, так как она смежна с обеими посещенными вершинами. В то же время вершины 2, 9 и 19 смежны только с одной из посещенных вершин. Выберем для определенности вершину 2 для следующего посещения.

В этом случае $V_v = \{11, 10, 2\}$; числа $|\text{Adj}(w) - V_v|$ для вершин 17, 18 и 9 равны соответственно 3, 2 и 2. Поэтому выбираем вершину 17. Следующими будут вершина 4, вводящая два новых ребра (в то время как вершины 12 или 18 вводили бы лишь по одному), и, наконец, вершина 12, смежная только с двумя посещенными вершинами (тогда как вершина 18 смежна с пятью). При возвращении в вершину 4 находим древесное ребро (4, 18). На рис. 4.26 (а) показано одно из возможных упорядочений, получаемых этим путем. Можно увидеть четыре разделителя: (11), (10, 2), (17, 4) и (14). Как и ожидалось, эта стратегия порож-

дает больше разделителей, чем стратегия короткого обратного ребра. Соответствующая переупорядоченная матрица приведена на рис. 4.26 (b). При гауссовом исключении в этой матрице появится 10 новых ненулевых элементов.

Если пользователь имеет дело с большой задачей, то сложный алгоритм упорядочения может оказаться выгодным. Наличие подходящего алгоритма способно даже определить возможность или невозможность решения задачи. Для задачи со средними размерами простой метод упорядочения часто дает большой выигрыш (при малых затратах на программирование) по сравнению со случаем, когда переупорядочение не проводится вообще.

4.16. ЛЕКСИКОГРАФИЧЕСКИЙ ПОИСК

В этом параграфе мы продолжим анализ упорядочений с малым заполнением для симметричных матриц, но теперь с иной точки зрения. Мы рассмотрим специальный класс матриц, которые могут быть упорядочены таким образом, что гауссово исключение не породит заполнения. Затем мы используем свойства этих матриц, чтобы сформулировать процедуру, которая находит упорядочение с малым заполнением для произвольной матрицы. Как обычно, обсуждение будет проводиться в терминах теории графов. Пусть $G^A = (V, E)$ — неориентированный граф, ассоциированный с симметричной матрицей A, и пусть $G^F = (V, E \cup F)$ — соответствующий граф заполнения, ассоциированный с $U + U^T$. Здесь U — треугольный множитель в разложении $A = U^T D U$ матрицы A, F — множество новых ребер (или ненулевых элементов U), возникших в ходе разложения. Если граф G^A допускает порядок исключения, для которого $F = 0$ (т. е. при выполнении исключения в этом порядке никакого заполнения не будет), то будем называть G^A *совершенным графом исключения.* Само упорядочение назовем *совершенным порядком исключения.* Заметим, что даже если G^A — совершенный граф исключения, то при исключении в другом порядке заполнение возможно. Заметим еще, что каждый граф G^F является совершенным графом исключения, потому что при проведении нового исключения в том же порядке никакого заполнения не было бы.

Граф называют *триангулированным,* если для всякого цикла длины 4 или большей найдется хотя бы одно ребро, соединяющее некоторую пару несоседних вершин этого цикла. Отметим различие между кликой и триангулированным графом. Каждая клика триангулирована, но не каждый триангулированный граф обязан быть кликой. Граф на рис. 4.1 (b) не триангулирован; например, путь (1, 6, 9, 11, 10, 1) представляет собой цикл длины 5, никакие две несоседние вершины которого не соединены ребром. Граф на рис. 4.2 (a) триангулирован, потому что в нем нельзя найти

цикл длины ≥ 4, не разделяемый хотя бы одним ребром. Упражнением для читателя может послужить проверка, что граф на рис. 4.17 не триангулирован. Всякое дерево есть триангулированный граф, поскольку не имеет циклов. Существуют и другие важные классы триангулированных графов; например, k — деревья [Rose, 19741 и интервальные графы [Fulkerson, Gross, 1965; Gilmore, Hoffmann, 1964].

Основное свойство, связывающее триангулированные графы с совершенными графами исключения, формулируется следующей теоремой [Rose, 1970; Rose et al., 1976]: граф G тогда и только тогда является совершенным графом исключения, когда он триангулирован.

Имея это основное свойство в виду, рассмотрим следующие варианты задачи об уменьшении заполнения. Если известно, что граф заданной матрицы триангулирован, мы можем пожелать найти совершенный порядок исключения, не создающий заполнения. Если неизвестно, триангулирован ли граф, то мы прежде всего сталкиваемся с задачей распознавания. Третья ситуация, часто возникающая в практике: граф $G^A = (V, E)$ не триангулирован, и мы ищем хорошее упорядочение. Поскольку соответствующий граф $G^F = (V, E \cup F)$, как мы знаем, будет совершенным графом исключения и, следовательно, триангулирован, то добавление к G^A множества ребер F можно рассматривать как *триангуляцию* графа G^A. Поэтому можно задаться вопросом, как триангулировать граф добавлением малого числа ребер. Эти задачи мы обсудим после того, как опишем лексикографический поиск.

Как и всякий поиск, *лексикографический поиск* — это систематическая процедура посещения вершин и исследования ребер графа $G = (V, E)$. Вершины обходятся в некоторой последовательности и нумеруются в обратном порядке, от n — $|V|$ к 1. В ходе поиска каждой вершине сопоставляется метка. Она представляет собой упорядоченное по убыванию подмножество множества $\{n, n-1, \ldots, 1\}$. Между метками устанавливается отношение порядка таким образом. Пусть даны две метки

$$L_1 = (p_1, p_2, \ldots, p_k), \qquad (4.32)$$
$$L_2 = (q_1, q_2, \ldots, q_l).$$

Полагаем $L_1 > L_2$, если для некоторого j

$$p_i = q_i, \quad i = 1, 2, \ldots, j-1,$$
$$p_j > q_j$$

или если

$$p_i = q_i, \quad i = 1, 2, \ldots, l,$$
$$k > l.$$

Две метки считаются равными, если $k = l$ и $p_i = q_i$ для $i = 1, 2, \ldots, k$. Так, например:

$$(7, 3) > (6, 4, 3, 2),$$
$$(7, 5, 2) > (7, 4, 3),$$
$$(7, 3, 1) > (7, 3),$$
$$(7, 3) = (7, 3).$$

Алгоритм лексикографического поиска формулируется следующим образом:

Шаг 1 (Инициализация). Приписать пустую метку всем вершинам. Положить $i \leftarrow n$.

Шаг 2 (Выбор новой вершины). Выбрать любую ненумерованную вершину v с наибольшей меткой и присвоить v номер i. В этот момент метка (v) имеет окончательную форму и при необходимости может быть сохранена. В противном случае ее можно уничтожить, так как алгоритмом она больше не используется.

Шаг 3 (Построение множества *S).* Найти множество S вершин w со свойствами: w ненумерована и либо w смежна с v, либо имеется путь $(v = v_1, v_2, \ldots, v_k, v_{k+1} = w)$, такой что метка (w) больше метки (v_j), $j = 2, 3, \ldots, k$.

Шаг 4 (Модификация меток). Для каждой вершины $w \in S$ добавить i к метке (w).

Шаг 5 (Цикл или остановка). Положить $i \leftarrow i - 1$. Если $i > 0$, перейти к шагу 2. В противном случае — остановка.

Алгоритм выбирает для следующего посещения вершину с наибольшей меткой. Это значит, что вершина w в описании шага 3 либо смежна с v, либо будет иметь больший номер, чем любая промежуточная вершина v_j пути. Так как номер w меньше, чем у v, то в обоих случаях в упорядоченном графе v должна быть достижима из w через множество вершин с меньшими, чем у w, номерами (определение достижимой вершины см. в § 4.2). Следовательно, в G^F имеется ребро (v, w). Другими словами, метка каждой вершины, получаемая в окончательной форме на шаге 2, — это в точности множество ненулевых элементов соответствующей строки верхнего треугольного множителя U.

Пусть F_α — множество новых ненулевых элементов [1]), возникающих при исключении в некотором порядке α. Будем говорить, что α — *минимальный порядок исключения,* если ни для какого другого порядка β не выполняется строгое включение $F_\beta \subset F_\alpha$. Если ни для какого другого порядка β не может быть

[1]) Имеются в виду позиции ненулевых элементов, а не их численные значения. — *Прим. перев.*

$| F_\beta | < | F_\alpha |$, т. е. не может достигаться меньшее заполнение, то α называют *наилучшим порядком исключения.* Эффективные алгоритмы для генерирования наилучших упорядочений неизвестны. Главное свойство лексикографического поиска — то, что он порождает минимальное упорядочение произвольного графа.

Лексикографический поиск требует для своей реализации времени $O(|V||E|)$. Его результатом будет не только минимальное упорядочение, но также, если нужно, полный портрет верхнего треугольного множителя *U.* Кроме того, список столбцовых индексов ненулевых элементов каждой строки (т. е. окончательная метка соответствующей вершины) будет получен упорядоченным по убыванию. Если заданный граф является совершенным графом исключения, то при лексикографическом поиске будет найден совершенный порядок исключения, т. е. наилучший порядок. Таким образом, если мы не знаем, будет ли наш граф совершенным, мы можем выяснить это за время $O(|V||E|)$ и вдобавок получить совершенное упорядочение. Если, однако, мы наперед знаем, что имеем дело с совершенным графом исключения, то можно модифицировать шаг 3 следующим образом:

Шаг 3' (Построение частичного множества триангуляции). Найти множество S ненумерованных вершин, смежных с *v.*

Модифицированный алгоритм выполняется за время $O(|V| + |E|)$. Разумеется, если граф не является совершенным, полученное упорядочение может не быть минимальным.

При реализации этого алгоритма в действительности не вычисляют метки вершин. Вместо этого для каждого значения метки заводят список всех вершин, имеющих эту метку. Списки хранятся в виде очереди, упорядоченной лексикографически, от наибольшей метки к наименьшей. Этот подход очень хорошо согласуется с компактной схемой хранения Шермана, обсуждавшейся в § 1.10. Другой алгоритм вычисления минимальных упорядочений предложен в [Ohtsuki, 1976]. Время работы этого алгоритма также $O(|V||E|)$, но он не находит позиции заполнения, порождаемого данным упорядочением. Минимальные упорядочения необязательно близки к наилучшим. Так, для квадратной сетки *nxn* заполнение, соответствующее лексикографическому упорядочению, равно $O(n^3)$, в то время как для метода вложенных сечений — $O(n^2 \log n)$, что является с точностью до постоянного множителя наилучшим возможным значением.

Проиллюстрируем лексикографический поиск на примере графа рис. 4.27 (a). Это тот же граф, что и на рис. 4.1 (b); только вершины во избежание путаницы помечены буквами. Начнем поиск с вершины *A.* Ей присваивается номер 11. Так как в этот момент все вершины имеют пустые метки, то множество *S* содержит только вершины *J* и *F.* Добавляем 11 к меткам вершин *J* и *F.*

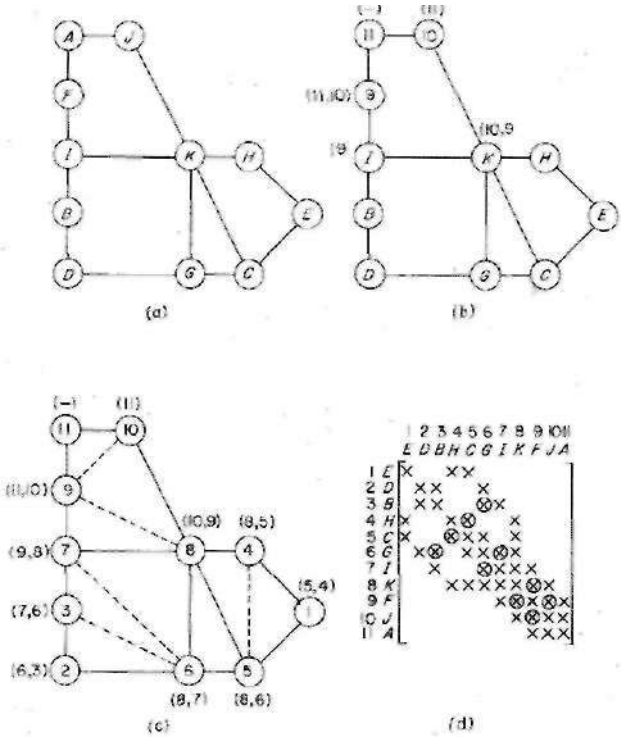


Рис. 4.27. Лексикографический поиск для графа рис. 4.1 (b), начинаемый с вершины *A*.

При следующем выполнении цикла приходится делать выбор, поскольку вершины *J* и *F* имеют одинаковые метки. Выберем для определенности вершину *J* и присвоим ей номер 10. Теперь $S = \{K, F\}$, так как *K* смежна с *J*, а *F* связана с *J* путем (*J*, *K*, *I*, *F*), в котором промежуточные вершины *K* и *J* имеют меньшие, чем у *F*, метки [1]); при этом *K*, *I* и *F* не нумерованы. Добавляем 10 к меткам *K* и *F*. Заметим, что результат этой операции не будет зависеть от порядка, в каком изменяются метки. Если вначале изменить метку *K*, то и в этом случае метка (*K*) меньше метки (*F*).

Теперь наибольшую метку, а именно (11, 10), имеет *F*. Присваиваем ей номер 9 и добавляем 9 к меткам вершин *I* и *K*, составляющих новое множество *S*. Ситуация в этот момент изображена на рис. 4.27 (b). Читатель самостоятельно закончит это упражнение. Результат показан на рис. 4.27 (c), где приведены минимальное упорядочение и конечные метки вершин. Соответ-

[1]) Точнее, пустые метки. — *Прим. перев.*

ствующая переупорядоченная матрица представлена на рис. 4.27 (d). При исключении в ней появятся пять новых ненулевых элементов, указанных взятыми в кружок крестиками; эти элементы соответствуют пяти ребрам, отмеченным на рис. 4.27 (c) пунктирными линиями; ненулевые элементы каждой строки имеют столбцовые индексы, описываемые метками. Исходный граф не триангулирован и не допускает совершенного порядка исключения. В то же время граф заполнения на рис. 4.27 (c) триангулирован, причем эта триангуляция минимальна, так как ни одно из добавленных ребер нельзя удалить без потери свойства триангулированности.

Другое полезное упражнение — применение лексикографического поиска к графу рис. 4.2 (a). Этот граф триангулирован, и если начать с вершины 12, будет получен следующий порядок: (6, 5, 20, 15, 13, 3, 8, 7, 1, 16, 14, 19, 9, 11, 10, 2, 18, 17, 4, 12). Читатель может проверить, что это совершенный порядок исключения, не порождающий заполнения. Сравним этот результат с рис. 4.25, где показан другой совершенный порядок исключения для той же матрицы.

4.17. СИММЕТРИЧНЫЕ НЕЗНАКООПРЕДЕЛЕННЫЕ МАТРИЦЫ

Если исключение проводится для симметричной положительно определенной матрицы и главные элементы выбираются на главной диагонали, то независимо от их порядка численная устойчивость обеспечена. Для матриц общего вида это уже не так, и для получения надежных результатов следует применять методы выбора главных элементов, описанные в § 3.6.

Если заданная матрица A симметрична, но незнакоопределена, то можно игнорировать симметрию и пользоваться методами для матриц общего вида, обсуждаемыми в гл. 5. В работе [Duff, Reid, 1976] показано, что по крайней мере в некоторых случаях этот путь выгоднее, чем попытки сохранить симметрию.

Если A симметрична и почти положительно определена и мы хотели бы сохранить симметрию, то можно, следуя предложению Даффа [Duff, 1980b], представить A в виде

$$A = M - N, \tag{4.33}$$

где M — симметричная положительно определенная, а N— симметричная матрица, и затем решать линейную систему $Ax = b$ итерационным процессом

$$Mx^{(k+1)} = Nx^{(k)} + b. \tag{4.34}$$

Линейную систему (4.34) приходится решать на каждом шаге, для чего можно использовать эффективные прямые методы для положительно определенных матриц или какой-нибудь итера-

ционный метод типа сопряженных градиентов [Concus et al., 1976].

В более общем случае, когда A симметрична и незнакоопределена и нужно сохранять симметрию, рекомендуется метод блочной диагонализации. Этот метод был кратко описан в § 3.6 [1]). Метод распространен на разреженный случай в работе [Duff et al., 1977]; соответствующая программа описана в [Munksgaard, 1977]. Дальнейшее обсуждение этого метода и родственных вопросов можно найти в [Duff, 1978, 1981a]. Можно применить и другие методы, упоминаемые в конце § 3.6. Так как, однако, они основаны на конгруэнтных преобразованиях, а не на гауссовом исключении, то разреженность может быть утрачена.

[1]) См. также книгу Икрамов Х. Д. Численное решение симметричных линейных систем. — М.: Наука, 1988.

Глава 5

Упорядочение для гауссова исключения: матрицы общего вида

5.1. ВВЕДЕНИЕ: ПОСТАНОВКА ЗАДАЧИ

В этой главе мы исследуем, как меняется разреженность матрицы B общего вида при применении к ней гауссова исключения. Его результатом будет треугольное разложение B в одной из следующих форм:

$$B = LDU, \tag{5.1}$$

$$B = L'U. \tag{5.2}$$

Здесь U — верхняя треугольная матрица с единичной диагональю; D — диагональная; L — нижняя треугольная с единичной диагональю; $L' = LD$ — нижняя треугольная; L и L' имеют одинаковый портрет, и $L \neq U^T$, *если* B несимметрична.

Как обычно, мы не принимаем в расчет случайное взаимное уничтожение, которое может произойти в ходе исключения. Взаимные сокращения редки, и попытки использовать их обычно не окупают себя. Таким образом, портреты L и U включают в себя позиции всех ненулевых элементов соответственно нижнего и верхнего треугольников B плюс заполнение, которое представляет собой множество позиций новых ненулевых элементов, созданных исключением в L и U. Когда мы говорим об уменьшении

заполнения, то имеем в виду общее заполнение в $L + U$. Здесь мы связаны с позициями ненулевых элементов, но не с их численными значениями.

Чтобы сформулировать нашу задачу, обозначим через

$$Ax = b \tag{5.3}$$

систему линейных уравнений, которую хотим решить; здесь A — разреженная матрица общего вида. Пусть P, Q — произвольные матрицы перестановок, так что, в частности $QQ^T = I$. Уравнение (5.3) можно переписать следующим образом:

$$PAQQ^Tx = Pb, \tag{5.4}$$

или

$$By = c, \tag{5.5}$$

где $y = Q^Tx$, $c = Pb$ и $B = PAQ$ — переупорядоченная форма A. В общем случае $Q \neq P^T$, так что B получена несимметричными перестановками строк и столбцов A. Мы хотим найти P и Q, такие что разложение B в форме (5.1) или (5.2) будет численно устойчивым, а заполнение — малым.

Даже если свойства матрицы A неизвестны, можно сразу приступить к исключению, используя, например, стандартную комбинацию: метод Марковица для сохранения разреженности (см. § 5.11) и выбор главного элемента с учетом барьера, обеспечивающего устойчивость вычислений (§ 3.6). Выбор главных элементов автоматически порождает P и Q. Если A вырождена, этот факт будет обнаружен в ходе разложения, поскольку на каком-то этапе появится нулевой или очень малый главный элемент. Система $Ax = b$ с вырожденной матрицей A может не иметь решения; например:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}. \tag{5.6}$$

Если же она совместна, то решений будет бесконечно много; например:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{5.7}$$

Читатель, желающий немедленно приступить к разложению, может пропустить § 5.2—5.10 и перейти к § 5.11, где описываются способы выбора главных элементов.

Однако, если свойства матрицы A неизвестны, возможно, имеет смысл попытаться до начала исключения переупорядочить ее к блочной нижней треугольной форме. Линейная система

с блочной нижней треугольной матрицей блочного порядка N имеет вид

$$\begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \cdot & \cdot & & \\ \cdot & & \cdot & \\ \cdot & & & \cdot \\ A_{N1} & & & A_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_N \end{bmatrix}; \qquad (5.8)$$

элементами являются матрицы, каждый диагональный блок A_{ii} - квадратный порядка n_i, и

$$\sum_{i=1}^{N} n_i = n.$$

Систему (5.8) можно решить как последовательность N меньших задач. Задача i имеет порядок n_i и матрицу коэффициентов A_{ii}, $i = 1, 2, \ldots, N$. При этом заполнение будет происходить только в диагональных блоках, даже если выбор главных элементов вызвал несимметричные перестановки строк и столбцов каждого блока. Процедура такова:

(1) Решить первую подсистему, для которой A_{11} является матрицей коэффициентов, относительно первых n_1 неизвестных. Вычислить вектор x_1 порядка n_1.

(2) Вычесть произведение $A_{j1}x_1$ из j-й правой части для $j = 2, \ldots, N$. Будет получена блочная нижняя треугольная матрица порядка $N - 1$, и процедура повторяется, пока не будет вычислено полное решение.

Очевидно, что следует сделать предположение о невырожденности диагональных блоков в системе (5.8).

Мы скажем, что матрица имеет *полную трансверсаль,* если все ее диагональные элементы не равны нулю. Трансверсаль есть в точности множество ненулевых диагональных элементов. Всякую невырожденную матрицу A посредством несимметричных перестановок с подходящими матрицами P и Q можно переупорядочить так, чтобы PAQ имела полную трансверсаль. Обратное, однако, неверно; например, вырожденная матрица системы (5.6) имеет полную трансверсаль.

Когда матрица A заданной линейной системы (5.3) имеет полную трансверсаль, блочную нижнюю треугольную форму (5.8) можно получить путем симметричных перестановок вида PAP^T. Если существует блочная форма [1]), то A называют *разложимой* матрицей. Некоторые матрицы не могут быть приведены к (нетривиальной) блочной нижней треугольной форме. Это почти всегда так для систем, полученных дискретизацией дифференциаль-

[1]) Нетривиальная (т. е. $N > 1$). — *Прим. перед.*

ных уравнений с частными производными. В других случаях разложимость матрицы может быть известна из ее построения. Процедуры приведения матрицы [1]) обсуждаются в § 5.9—5.10.

Если A содержит нули на главной диагонали, то необходимо получить полную трансверсаль, прежде чем пытаться переупорядочить матрицу к блочной форме. Получение полной трансверсали требует несимметричных перестановок. Матрицу, допускающую несимметричное переупорядочение к форме с полной трансверсалью, а затем симметричное переупорядочение к блочной нижней треугольной форме, будем называть *двояко разложимой* [2]). Процедуры получения полной трансверсали разбираются в § 5.7—5.8.

Теория графов играет важную роль в анализе разреженных несимметричных матриц. Со всякой несимметричной матрицей, имеющей полную трансверсаль, можно связать ориентированный граф, или *орграф.* Орграф инвариантен относительно симметричных перестановок строк и столбцов матрицы; меняется лишь его помечивание. Орграф может быть разбит на *сильные компоненты,* соответствующие диагональным блокам блочной нижней треугольной формы (5.8) данной матрицы. Можно показать, что матрица разложима тогда и только тогда, когде ее орграф может быть разбит на сильные компоненты [Harary, 1959]. Обе задачи — приведение разреженной матрицы и разбиение орграфа — эквивалентны, и для обеих используются одни и те же алгоритмы.

Если матрица не имеет полной трансверсали, то с ней можно ассоциировать *двудольный граф.* Двудольный граф инвариантен относительно несимметричных перестановок матрицы, и получение трансверсали эквивалентно отысканию паросочетаний в двудольном графе. Эта совокупность важных приложений теории графов к анализу разреженных несимметричных матриц мотивировала § 5.2, где даны необходимые определения и обсуждаются нужные свойства. Теория графов используется затем всюду в этой главе.

5.2. ТЕОРИЯ ГРАФОВ ДЛЯ НЕСИММЕТРИЧНЫХ МАТРИЦ

В § 4.2 было показано, что всякой симметричной матрице с ненулевой диагональю можно поставить в соответствие неориентированный граф. Были обсуждены многие определения и свойства, связанные с неориентированными графами, в частности то, что граф не меняется при симметричных перестановках строк и столбцов матрицы.

В этом параграфе мы рассмотрим другие типы графов, используемых в связи с матрицами общего вида. Граф $G = (V, E)$ называется *ориентированным графом* или *орграфом* [Harary et

[1]) К блочно-треугольному виду. — *Прим. перев.*
[2]) В оригинале — bireducible. — ***Прим. перев.***

al,. 1965], когда пары его вершин, представляющие ребра, упорядочены. Для ориентированного ребра мы часто будем пользоваться обозначением $(u \to v)$.

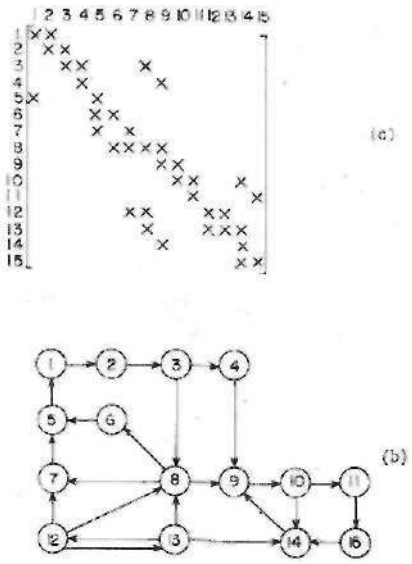


Рис. 5.1. Несимметричная матрица и ее ориентированный граф.

Всякой квадратной матрице A общего вида можно поставить в соответствие орграф [Rose, Bunch, 1972]. Пусть A — матрица порядка n с элементами a_{ij}: тогда орграф имеет n вершин $v_1, v_2, \ldots, v_n$ и $(v_i \to v_j)$ является ребром тогда и только тогда, когда $a_{ij} \neq 0$. Каждый ненулевой диагональный элемент a_{ij} соответствует петле $(v_i \to v_i)$. В тех случаях, когда на диагонали A нет нулей, присутствуют все петли, и обычно нет необходимости явно принимать их в расчет. Множество ненулевых диагональных элементов называется *трансверсалью.* Ассоциированный с A орграф остается тем же, если симметрично переставить строки и столбцы. Симметричная перестановка оставляет все диагональные элементы на диагонали. Другими словами, если $B = PAP^T$, где P — матрица перестановки, а G_A и G_B — орграфы, соответствующие A и B, то G_A и G_B идентичны с точностью до помечивания. На рис. 5.1 показаны несимметричная разреженная матрица с полной трансверсалью и отвечающий ей орграф.

Говорят, что ориентированное ребро $(u \to v)$ *исходит* из вершины u (или *начинается* в вершине u) и заходит в вершину v (или *кончается* в вершине v). Будем еще говорить, что ребро $(u \to v)$ *ведет* из u в v. *Полустепенью захода* вершины v называется число ребер, заходящих в v. *Полустепень исхода* — это число ребер, исходящих из v. *Источник* — это вершина с нулевой полустепенью захода и положительной полустепенью исхода; *сток* — вершина с нулевой полустепенью исхода и положительной полустепенью захода.

Если имеется ориентированное ребро $(u \to v)$, то говорят, что вершина v *смежна с* u. Если W — подмножество вершин G, то *смежное множество* для W, обозначаемое через Adj (W), — это множество вершин, не принадлежащих W, но смежных с вершинами из W. Именно пусть даны $G = (V, E)$ и $W \subset V$:

$$\mathrm{Adj}\,(W) = \{v \in V - W \mid \exists u \in W\colon\ \exists\,(u \to v) \in E\}. \qquad (5.9)$$

Если в орграфе существует путь из вершины *u* в вершину c/, то говорят, что *v достижима* из *u. Матрица достижимостей R* орграфа — это булева матрица, определяемая следующим образом: $r_{ij} = 1$, если v_j достижима из v_i в орграфе, и $r_{ij} = 0$ в противном случае. *Ориентированным циклом* называется путь по меньшей мере с двумя вершинами, у которого начальная и конечная вершины совпадают. Если *v* достижима в *G* из *u,* то расстояние от *u* до *v* — это длина кратчайшего пути из к в к; заметим, что расстояние от *v* до *u* может быть не определено или отличаться от расстояния от *u* до *v*.

Орграф, ассоциированный с активной подматрицей, полученной из матрицы *A* после *k* шагов гауссова исключения, называется *k*-м *орграфом исключения.* Каждый орграф исключения получается из предыдущего удалением вершины, отвечающей исключаемой на данном шаге переменной, скажем, вершины *v,* удалением всех ребер, исходящих или заходящих в *v,* и добавлением ориентированного ребра (*u* → *w)* для всякого ориентированного пути *(u, v, w)*. В качестве упражнения читатель может построить несколько первых орграфов исключения для матрицы рис. 5.1. *Орграф заполнения* — это граф матрицы *L* + *U.* Многие результаты о неориентированных графах, приведенные в предыдущих главах, могут быть распространены на ориентированный случай; см., например, [Rose et al., 1976, p. 281].

Орграф называют *слабо связным,* если связен соответствующий неориентированный граф, получаемый удалением ориентации ребер. Орграф на рис. 5.1 слабо связан. Вершины б, 7 и 9 смежны с вершиной 8. Если *W* = (8, 13), то Adj *(W)* = (6, 7, 9, 12, 14). Вершина 5 имеет полустепень захода 2 и полустепень исхода 1. Вершина 4 достижима из вершины 8, и в данном орграфе нет источников и стоков.

Слабо связный орграф может быть факторизован путем разбиения множества его вершин на непересекающиеся подмножества. В § 5.3 будет рассмотрено разбиение на сильные компоненты и будут также введены структуры уровней связности. Структуры уровней смежности, используемые для получения трансверсали разреженной матрицы, обсуждаются в § 5.5.

Имея дело с матрицами общего вида, мы будем допускать несимметричные перестановки строк и столбцов *A,* т. е. переупорядочения типа $B = PAQ$, где $Q \neq P^T$. Орграф *A* не инвариантен относительно несимметричных перестановок. В этом причина обращения к *двудольным графам* [Dulmage, Mendelsohn, 1959, 1967]. Двудольный граф состоит из двух различных множеств вершин *R* и *C*, по *n* вершин в каждом, и неориентированных ребер, соединяющих вершины *R* с вершинами *C.* Множество *R* ассоциировано со строками матрицы, а множество *C* — с ее столбцами. Вершины *R* назовем *юношами,* вершины *C* — *девушками.* Ребро

(r_i, c_j) присутствует в двудольном графе тогда и только тогда, когда $r_i \in R$, $c_j \in C$ и $a_{ij} \neq 0$. Ребра вида (r_i, c_i) [1]) всегда указываются, поскольку одно из важнейших приложений двудольных графов заключается в отыскании перестановок, максимизирующих трансверсаль A, что эквивалентно максимизации числа таких ребер. В § 5.8 показано, что эта задача эквивалентна также разысканию *наибольшего паросочетания* или *соответствия* между юношами и девушками. Применение к A различных перестановок строк и столбцов меняет лишь домечивание двудольного графа. Другими словами, матрицы A и $B = PAQ$ имеют с точностью до помечивания один и тот же двудольный граф, даже если $Q \neq P^T$. Матрица и соответствующий ей двудольный граф показаны на рис. 5.11.

Существуют и другие типы графов, используемых в связи с разреженными матрицами, например *строчные графы* и *столбцовые графы* [Mayoh, 1965; Tewarson, 1967a]. Строчный граф матрицы A — это неориентированный граф симметричной матрицы $B = A * A^T$, где $*$ обозначает умножение матриц без учета возможных взаимных уничтожений; иначе говоря, если какой-то элемент B равен нулю вследствие численного взаимного сокращения, он рассматривается как ненулевой, и соответствующее ребро включается в строчный граф. Перестановки строк A изменяют лишь помечивание вершин строчного графа, в то время как перестановки столбцов вообще его не меняют. Столбцовый граф A — это неориентированный граф симметричной матрицы $C = A^T * A$. Столбцовый граф матрицы A не меняется при перестановках ее строк, но изменяет помечивание при перестановках столбцов.

Дополнительный материал на эту тему можно найти в [Tewarson, 1973; Harary, 1971; Tarjan, 1976].

5.3. СИЛЬНЫЕ КОМПОНЕНТЫ ОРГРАФА

Рассмотрим слабо связный орграф $G = (V, E)$. Орграф G называют *сильно связным,* если для любой пары вершин $v, w \in V$ существуют путь из v в w и путь из w в v, т. е. v и w взаимно достижимы. Поскольку путь из v в w вместе с путем из w в v составляют цикл, то сильно связный орграф можно определить и как орграф, в котором для всякой пары вершин существует цикл, содержащий обе вершины. Матрица достижимостей сильно связного орграфа состоит из одних единиц. Орграф, показанный на рис. 5.1, не является сильно связным: если в качестве тестовых взять вершины 12 и 9, то мы найдем путь (12, 8, 9) из 12 в 9, но пути из 9 в 12 нет.

[1]) Если таковые имеются. — *Прим. перев.*

В общем случае орграф G может не быть сильно связным. *Сильно связной компонентой* или, короче, *сильной компонентой* называется G частичный граф в G, являющийся сильно связным и не допускающий расширения без потери этого свойства. Напомним (см. § 4.2), что частичный граф содержит все ребра G вида (v, w), такие что обе вершины v, w принадлежат ему. Из определения сильной компоненты вытекает существование цикла, которому принадлежат любые две заданные вершины компоненты. Из него вытекает также, что любой цикл в G должен быть целиком составлен из вершин данной сильной компоненты, либо, наоборот, целиком составлен из вершин G, ей не принадлежащих. В самом деле, если бы существовал цикл, содержащий вершину v сильной компоненты и вершину w, не принадлежащую ей, то мы могли бы добавлением w расширить сильную компоненту без потери сильной связности, что противоречит определению. Это свойство используется для выделения сильных компонент: вначале в G находят цикл, а затем насколько возможно расширяют его, исследуя другие циклы. Например, исследование графа на рис. 5.1 обнаруживает циклы (1, 2, 3, 8, 7, 5, 1) и (1, 2, 3, 8, 6, б, 1), имеющие общие вершины 1, 2, 3, 8 и 5. Множество вершин {1, 2, 3, 5, 6, 7, 8} и соответствующие ребра образуют сильную компоненту, потому что никакой другой цикл графа не содержит вершины этого множества.

Свойства, которые мы только что обсудили, означают, что. слабо связный орграф $G = (V, E)$ можно разбить на сильные компоненты C_1, C_2, ... C_s с непересекающимися множествами вершин. Если G сам сильно связан, то $s = 1$, и сильная компонента только одна. В противном случае G допускает разбиение и $s > 1$. Мы скажем, что ребро $(v \rightarrow w)$ является *выходом* или что оно *исходит* из сильной компоненты $C = (V_c, E_c)$, если $v \in V_c$, а $w \notin V_c$. Ребро $(v \rightarrow w)$ есть *вход* (или же оно *заходит* в C), если $v \notin V_C$, а $w \in V_C$. Так как сильная компонента — это частичный граф, то входы и выходы не принадлежат никакой сильной компоненте.

Если G допускает разбиение, то, как легко видеть, должна существовать хотя бы одна сильная компонента, не имеющая выходов. Действительно, если бы каждая компонента имела выход, то мы смогли бы проследить путь от одной компоненты к другой, затем к третьей и т. д., пока наконец не достигли бы одной из компонент, где уже были прежде; цикл, полученный таким образом, противоречит определению сильных компонент. В общем случае может существовать несколько компонент без выходов. Будем говорить, что все сильные компоненты без выходов принадлежат *уровню* 1.

Продолжая рассуждать подобным образом, мы можем заключить, что среди оставшихся сильных компонент G должна суще-

ствовать хотя бы одна такая, что каждое исходящее из нее ребро заходит в некоторую компоненту уровня 1. Вообще говоря, таких компонент может быть несколько, и мы будем говорить, что все они принадлежат *уровню* 2. Среди компонент G, не входящих ни в уровень 1, ни в уровень 2, должна быть хоть одна, из которой все выходы ведут в уровни 1 и 2; по крайней мере один из этих выходов должен заходить в уровень 2, иначе компонента принадлежала бы уровню 2, а, по нашему предположению, это не так. Названная компонента и, если найдутся, все другие с теми же свойствами образуют *уровень* 3. Заметим, что у компонент уровня 3 могут отсутствовать выходы, ведущие в уровень 1. Если продолжить описанную процедуру, то будет получена *структура уровней связности;* вот ее строгое определение:

$$L_1 = \{C_i \mid C_i \text{ не имеет выходов}\},$$
$$L_l = \{C_i \mid \text{если } (u \to v) \text{ — выход из } C_i \text{ и } v \in C_j, \text{ то } C_j \in L_{l'}, \text{ где } l' < l\},\ l = 2, 3, \ldots, m.$$

Определение подразумевает, что любая сильная компонента уровня $l > 1$ должна иметь по меньшей мере один выход, ведущий в уровень l — 1. Число m называется *длиной* разбиения. В отличие от структуры уровней смежности, структура уровней связности орграфа единственна. Мы будем говорить, что вершина v графа G находится в уровне l или что Уровень $(v) = l$, если v принадлежит сильной компоненте уровня l. Если $(v \to w)$ — ребро орграфа, то Уровень (v) = Уровень (w) тогда и только тогда, когда v и w принадлежат одной и той же сильной компоненте; в противном случае, Уровень (w) < Уровень (v). Сильная компонента уровня l может быть достигнута только ребрами, исходящими из уровней $l + 1, \ldots m$; однако может случиться, что сильная компонента не имеет входов, даже если ее уровень $l < m$.

Другую структуру уровней связности можно определить, помещая в уровень 1 все компоненты, не имеющие входов, и продолжая вышеописанный процесс с заменой слова «выход» на слово «вход». Говорят, что две структуры уровней являются *двойственными.* Мы сосредоточим внимание на структуре уровней первого типа.

Ациклические орграфы заслуживают специального упоминания. Орграф называется ациклическим, если он не имеет циклов: если путь начинается в произвольной вершине v, он никогда в v не вернется. Это значит, что каждая вершина ациклического орграфа представляет собой сильную компоненту. Поэтому число сильных компонент ациклического орграфа равно числу его вершин, хотя бы одна из вершин имеет нулевую полустепень

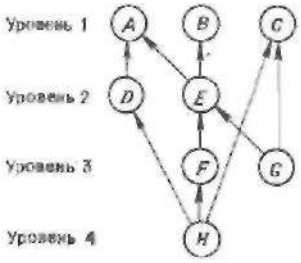


Рис. 5.2. Конденсация и структура уровней связности орграфа с восемью сильными компонентами.

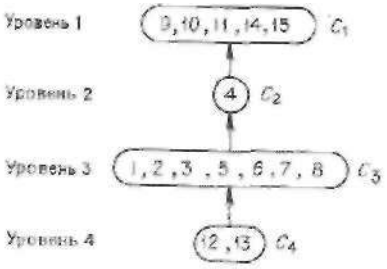


Рис. 5.3. Конденсация и структура уровней связности для орграфа на рис. 5.1 (b).

исхода и по крайней мере у одной вершины полустепень захода равна нулю. Одно из приложений ациклических орграфов к технологии разреженных матриц рассматривается в конце данного параграфа. Другое приложение будет рассмотрено в § 5.6 в связи с алгоритмом Хопкрофта—Карпа.

Разбиение произвольного орграфа на сильные компоненты удобно изображать посредством соответствующего факторграфа или *конденсации;* вершинами здесь являются сильные компоненты [Нагагу et al., 19651. Конденсация орграфа — это ациклический орграф; как и всякий ациклический орграф, он имеет хотя бы одну вершину с нулевой полустепенью исхода, т. е. хотя бы одну сильную компоненту без выходов. На рис. 5.2 показан пример конденсации и структуры уровней связности для орграфа с 8 сильными компонентами. Рис. 5.3 показывает конденсацию и структуру уровней, соответствующие орграфу на рис. 5.1.

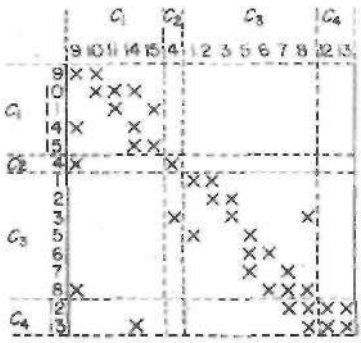


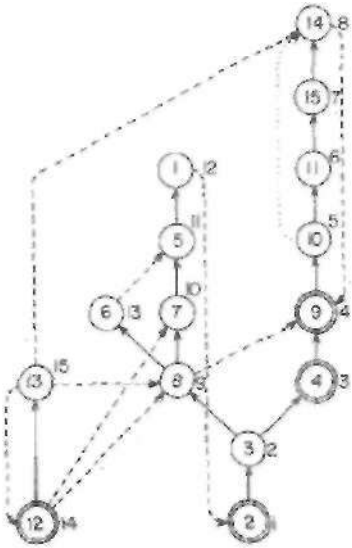
Рис. 5.4. Блочная нижняя треугольная матрица, полученная приведением матрицы на рис. 5.1 (a).

Пусть теперь G — орграф, ассоциированный с несимметричной матрицей *A.* Разобьем *G* на сильные компоненты и построим структуру уровней, как описано выше. Пусть вершины *G* упорядочены, и упорядочение совместимо с разбиением на компоненты и монотонно относительно уровней, т. е. все вершины каждой компоненты нумеруются последовательными числами и все компоненты каждого уровня l, $l = 1, 2, \ldots, m-1$, нумеруются раньше любой компоненты уровня $l' > l$. Тогда соответствующая симметрично переупорядоченная матрица будет блочной нижней

треугольной; каждой сильной компоненте с n_i вершинами отвечает квадратный диагональный блок порядка n_i; внедиагональные блоки соответствуют связям между сильными компонентами. Эти идеи были сформулированы в [Harary, 1969, p. 205]. Рис. 5.4 иллюстрирует сказанное для матрицы и графа, изображенных на рис. 5.1.

5.4. ПОИСК В ГЛУБИНУ НА ОРГРАФЕ

Поиск в глубину описан в § 4.15 как процедура обхода вершин и ребер графа: начинают с произвольной вершины s_1 и пытаются по возможности придерживаться последовательности вершина—ребро—вершина—ребро...; если в текущей вершине больше нет неисследованных ребер, то *возвращаются* к предыдущей вершине. В этом параграфе мы рассмотрим структуры, получаемые при выполнении поиска в глубину на орграфе, где фиксировано направление, в котором можно исследовать каждое ребро [Tarjan, 1972]. Предположим, что всякая вершина содержит петлю. При поиске действуют прежние правила. Пусть v — текущая вершина. Мы вошли в v либо впервые (и тогда v — последняя посещенная вершина), либо повторно при возвращении. Пусть исследуется ребро $(v \rightarrow w)$, причем w не посещалась прежде. Тогда w становится текущей вершиной, а ребро $(v \rightarrow w)$ квалифицируется как *древесное.* Если сохранять только древесные ребра и пренебрегать всеми остальными, то будет получено дерево с корнем в начальной вершине s_1. Поскольку теперь ребра ориентированы, то продолжение поиска может оказаться невозможным еще до того, как посещены все вершины. Так заведомо произойдет, если, например, s_1 принадлежит сильной компоненте С, не имеющей выхода (см. предыдущий параграф): и исследование ребер, и возвращение будут всегда приводить к вершинам из С, и дерево с корнем s_1 будет точным остовом С. В более общем случае при произвольном выборе s_1 будет получено дерево с корнем s_1, остовное лишь для некоторых сильных компонент графа. Тогда поиск возобновляется из новой начальной вершины s_2, которая пока что не посещалась и, следовательно, не принадлежит никакой из сильных компонент, натянутых на первое дерево. Снова пренебрегая всяким недревесным ребром, построим второе дерево с корнем в s_2, остовное для новой группы сильных компонент. Процедура продолжается, пока не будут посещены все вершины. Результатом будет семейство несвязанных деревьев, называемое *остовным лесом,* которое содержит все вершины орграфа и ребра, квалифицированные как древесные. Кроме того, при поиске нумеруются узлы в том порядке, в каком они посещаются. Будем обозначать через Номер (v) номер, приписанный вершине v в ходе поиска. В технологии разреженных матриц это упорядочение



Вершина v	Номер(v)	Связка(v)	
1	12	1	
2	1	1	*
3	2	1	
4	3	3	*
5	11	1	
6	13	1	
7	10	1	
8	9	1	
9	4	4	*
10	5	4	
11	6	4	
12	14	14	*
13	15	14	
14	8	4	
15	7	4	

Рис. 5.5. Джунгли, полученные применением алгоритма Тарьяна к орграфу на рис. 5.1 (b). Последовательность поиска указывают числа возле кружков. Корни сильных компонент отмечены двойными кружками.

и соответствие между деревьями леса и сильными компонентами используются алгоритмами, приводящими матрицу к блочной нижней треугольной форме.

Рассмотрим построение остовного леса для орграфа на рис. 5.1. Выбрав в качестве начальной вершину 2, можем проследить путь (2, 3, 4, 9, 10, 11, 15, 14). Все встреченные до сих пор ребра — древесные, что указано на рис. 5.5 сплошными стрелками. Ограничивая пока внимание древесными ребрами, видим, что из вершины 14 никакое древесное ребро не исходит. Приходится вернуться в вершину 3, где есть древесное ребро (3 → 8). Далее находим путь (3, 8, 7, 5, 1), снова возвращаемся к вершине 8 и обнаруживаем древесное ребро (8 → 6). Теперь первое дерево построено полностью. Сравнивая с рис. 5.3, замечаем, что это дерево остовное для компонент C_1, C_2 и C_3 орграфа. Так как некоторые вершины еще не посещались, нужно выбрать новую начальную вершину, скажем 12. Находим древесное ребро (12 → 13), и этим поиск завершен. Второе дерево остовное для компоненты C_4. Остовной лес в этом примере состоит из двух деревьев; на них натянуты все четыре сильные компоненты орграфа.

Исследуем теперь более детально связь между деревьями и натянутыми на них сильными компонентами. Пусть s — начальная вершина; T — дерево с корнем s, построенное посредством поиска в глубину. Покажем прежде всего, что T можно разбить на не-

связанные поддеревья, каждое из которых служит остовом ровно одной сильной компоненты. Пусть C = (V_C, E_C) — одна из сильных компонент, натянутых на T, и пусть T_C — *наименьшее* поддерево Т, содержащее *все* вершины С. Пусть $v \in V_C$ — корень T_C. Легко показать, что дереву T_C принадлежат *только* вершины из C. Действительно, предположим, что $w \notin V_c$ — вершина T_c. Вершина w не является корнем T_c; не может она быть и конечной вершиной, так как по предположению T_c — наименьшее поддерево, содержащее все вершины C. Тогда v — предок для w, и w должна иметь потомка (скажем, x) в C. В T_c имеется путь из v в x, содержащий w; поскольку С — сильная компонента, то в С существует путь из x в v. Оба пути образуют цикл, которому принадлежит w; *но* тогда w должна быть вершиной C вопреки нашему предположению.

Таким образом, T можно разбить на несвязанные поддеревья, каждое из которых остовное ровно для одной сильной компоненты. Корень поддерева — это первая вершина соответствующей компоненты, посещаемая в ходе поиска; он называется *корнем сильной компоненты*. В частности, начальная вершина s является корнем, той сильной компоненты, которой принадлежит. Эти результаты доказаны как теоремы в [Tarjan, 1972]. Следовательно, задача отыскания сильных компонент орграфа сводится к задаче отыскания корней поддеревьев. Будем говорить, что поддерево относится к уровню l, если соответствующая компонента находится в уровне l структуры уровней связности данного орграфа (см. § 5.3). Чтобы классифицировать поддеревья графа, показанного на рис. 5.1, сравним рисунки 5.5 и 5.3. Вершина 12 — корень поддерева (12, 13) уровня 4. Вершина 2 — корень поддерева (2, 3, 8, 7, 5, 1, 6) уровня 3. Вершина 4 сама образует поддерево, относящееся к уровню 2; вершина 9 — корень поддерева (9, 10, 11, 15, 14) уровня 1. Каждое из этих поддеревьев остовное ровно для одной сильной компоненты. Отметим, что поддерево уровня 1 не имеет выхода.

В заключение исследуем соотношения между недревесными ребрами, с одной стороны, и деревьями леса и сильными компонентами орграфа — с другой. Это будет сделано с учетом следующих существенных свойств поиска в глубину. Пусть T — одно из деревьев леса; u — произвольная вершина T; T_v — поддерево с корнем v, содержащее v и всех ее потомков в T. Тогда v — первая вершина T_n, посещаемая при поиске, поэтому она имеет наименьший номер в T_v; если w — потомок v, то Номер $(v) <$ $<$ Номер (w). Кроме того, все вершины T_v будут посещены до возвращения через v. В частности, если v — корень сильной компоненты, то номер v меньше номера любой другой вершины этой компоненты, и все вершины последней будут посещены, прежде чем покинуть ее возвращением через v.

Рассмотрим теперь некоторый шаг поиска. Пусть v — текущая вершина, $(v \rightarrow w)$ — ребро, не являющееся древесным. Это означает, что w посещалась до того, как v стала текущей вершиной. Следовательно, в этот момент определены и Номер (v), и Номер (w). Кроме того, если v относится к некоторому уровню l, то, как мы знаем, уровень l' вершины да должен удовлетворять условию $l' \leq l$; знак равенства достигается только, если v и w принадлежат одной и той же сильной компоненте. Если обнаружено недревесное ребро $(v \rightarrow w)$, то существуют следующие возможности:

(1) w — предок для v. В этом случае Номер (w) < Номер (v), а ребро $(v \rightarrow w)$ называется *обратным* [1]). Поскольку уже существует путь из w в v, образованный древесными ребрами, то роль обратного ребра состоит в замыкании цикла. Таким образом, v и ее предки вплоть до (и включая) w принадлежат одной и той же сильной компоненте. На рис. 5.5 ребро $(14 \rightarrow 9)$ обратное, поскольку 9 — предок для 14, и вершины 9, 10, 11, 15, 14 принадлежат сильной компоненте C_1.

(2) w — потомок v. В этом случае Номер (w) > Номер (v), а ребро $(v \rightarrow w)$ называется *лишним древесным ребром* [2]) [Gustavson, 1976b]. Лишнее древесное ребро соединяет две вершины, уже связанные путем из древесных ребер. Следовательно, оно не помогает поиску сильных компонент, и если наша цель состоит именно в этом, им можно пренебречь. На рис. 5.5 лишним древесным ребром является $(10 \rightarrow 14)$.

(3) w — ни потомок, ни предок для v. Ребро $(v \rightarrow w)$ называется *поперечным* [3]). Рассмотрим поддерево T_v с корнем v, содержащее все потомство v. В ходе поиска, после прихода в v, посещаются и нумеруются все вершины T_v и только они, пока T_v не будет покинуто возвращением через V. Вершина да не принадлежит T_v, но номер ей был присвоен раньше, чем v стала текущей вершиной. Это значит, что да получила номер еще до того, как мы впервые вошли в u; следовательно, Номер (w) < Номер (v). Вершины v и w могут принадлежать одной или разным сильным компонентам; в первом случае Уровень (w) = Уровень (v); во втором — Уровень (w) < Уровень (v). На рис. 5.5 поперечными ребрами будут $(8 \rightarrow 9)$ и $(6 \rightarrow 5)$. Вершины 6 и 5 принадлежат одной и той же сильной компоненте C_3 уровня 3. Вершины 8 и 9 относятся соответственно к C_3 и C_1 причем Уровень (9) = 1 < < Уровень (8) = 3.

Структура, составленная из остовного леса и добавочных ребер, которые могут быть обратными, лишними древесными и поперечными, называется *джунглями*.

---

[1]) В оригинале — frond. — *Прим. перев.*
[2]) В оригинале — redundant tree arc. — *Прим. перев.*
[3]) В оригинале — cross-link. — *Прим. перев.*

Если поиск в глубину проводится на ациклическом орграфе, где каждая вершина является сильной компонентой, то получится более простая структура. Каждая вершина остовного леса — корень своей собственной сильной компоненты. Нет ни обратных, ни активных [1]) поперечных ребер. Все ребра будут либо древесными, либо стерильными [2]) поперечными. Поиск в глубину на ациклическом орграфе использован в [Hopcroft, Karp, 1973] в связи с задачей отыскания наибольших паросочетаний в двудольных графах.

5.5; ПОИСК В ШИРИНУ НА ОРГРАФЕ И СТРУКТУРЫ УРОВНЕЙ ОРИЕНТИРОВАННОЙ СМЕЖНОСТИ

В этом параграфе мы рассмотрим класс разбиений слабо связного орграфа $G = (V, E)$, которые могут быть построены посредством поиска в ширину. *Подструктура уровней ориентированной смежности* $L_0, L_1, \ldots, L_m$ получается, если $L_0 \subset V$ — заданное подмножество вершин G, а каждый из остальных уровней является смежным множеством для объединения предыдущих уровней:

$$L_i = \mathrm{Adj}\left(\bigcup_{j=0}^{i-1} L_j\right), \quad i = 1, 2, \ldots, m. \tag{5.10}$$

Число m называется *длиной* подструктуры уровней; *ширина* определяется как максимальное число вершин в произвольном уровне. L_0 — *корень* подструктуры. Если L_0 состоит из единственной вершины u, т. е. $L_0 = \{u\}$, то будем говорить, что вершина u — корень подструктуры. Множество вершин подструктуры

$$V_s = \bigcup_{i=0}^{m} L_i \tag{5.11}$$

содержит все вершины L_0 и те вершины, которые из них достижимы. В V_s, возможно, входят не все вершины G. В этом случае оставшееся подмножество $V—V_s$ может быть разбито аналогичным образом, и так далее, пока не будет получена *структура уровней ориентированной смежности,* охватывающая все вершины G Нумерация уровней, однако, в каждой подструктуре своя.

Структуры уровней смежности строятся с использованием поиска в глубину. Процедура схожа с описанной в § 4.15 про-

[1]) Активным называется поперечное ребро, концы которого принадлежат одной и той же сильной компоненте (см. с. 222). —*Прим. перев.*

[2]) Стерильным называется поперечное ребро, концы которого принадлежат разным сильным компонентам (см. определение на с. 216).—*Прим. перев.*

цедурой для неориентированных графов, но существуют некоторые важные различия. Вершины при поиске группируются по уровням, а ориентированные ребра квалифицируются как *древесные* либо как *поперечные.* Предположим, что поиск начинается с некоторой (произвольно выбранной) начальной вершины *s.* Тогда уровень L_0 состоит из единственной вершины *s,* и будет получена подструктура уровней с корнем в *s.* Пусть на некотором шаге поиска уже опознаны все вершины уровня L_i. Все вершины из L_i и предыдущих уровней помечены как посещенные. Далее для некоторого $v \in L_i$ мы исследуем ребра, ведущие из *v* к другим вершинам. Если найдено ребро $(v \rightarrow w)$, причем вершина *w* еще не посещалась, то это ребро древесное, а *w* принадлежит уровню L_{i+1}. Если найдено ребро $(v \rightarrow x)$ и *x* — посещенная вершина, то это ребро поперечное. Поскольку посещенные вершины имеются лишь в уровнях L_j, $j \leq i + 1$, мы приходим к заключению, что поперечное ребро, исходящее из вершины уровня L_i, может вести только в вершину уровня L_j, $j \leq i + 1$. В более общем случае, когда уже построены некоторые другие подструктуры, поперечное ребро может вести также и в любую вершину этих подструктур. Древесное ребро может соединять лишь вершину уровня L_i и вершину уровня L_{i+1}; если $j > i + 1$, то вершина уровня L_i и вершина уровня L_j не могут быть соединены. Граф $G_8 = (V_s, E_s)$, где E_s— множество древесных ребер, представляет собой дерево с корнем s, остовное для подструктуры. Если по завершении очередного дерева продолжать поиск, начиная с новой, не посещавшейся до сих пор вершины, и так до тех пор, пока не будут исчерпаны все вершины *G,* то в конечном счете будут получены джунгли и остовной лес (см. § 5.4). Описанный в § 4.3 алгоритм [Rose et al., 1976].после небольших изменений может быть использован для проведения поиска.

Для подструктуры уровней с корнем в s имеет место следующее свойство: если $v \in L_i$, то расстояние в *G* от s до *v* (т. е. длина кратчайшего пути из s в *v*) равно *i*. Кроме того, в остовном дереве существует ровно один кратчайший путь длины *i* [1]); он составлен из древесных ребер. Это свойство будет использовано в § 5.8 при отыскании трансверсали разреженной матрицы.

Рассмотрим для примера орграф на рис. 5.1 (b). Для этого графа подструктура уровней ориентированной смежности с корнем в вершине 8 изображена на рис. 5.6. Древесные ребра указаны сплошными линиями, поперечные ребра — пунктирными. Расстояние от вершины 8 до вершины 10 равно 2, поскольку вершина 10 принадлежит L_2. Кратчайший путь из 8 в 10 — (8, 9, 10); он содержится в дереве. Оба других пути из 8 в 10, а именно (8, 6, 5, 1, 2, 3, 4, 9, 10) и (8, 7, 5, 1, 2, 3, 4, 9, 10) длиннее. Все вершины,

[1]) Из *s* в *v*. — *Прим. перев.*

достижимые в графе из вершины 8, принадлежат данной подструктуре [1])

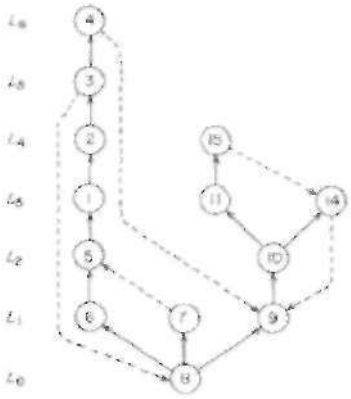


Рис. 5.6. Подструктура уровней ориентированной смежности с корнем в вершине 8 для орграфа на рис. 5.1 (b).

5.6. ОТЫСКАНИЕ МАКСИМАЛЬНОГО МНОЖЕСТВА ПУТЕЙ С РАЗЛИЧНЫМИ ВЕРШИНАМИ В АЦИКЛИЧЕСКОМ ОРГРАФЕ

Поиск в глубину имеет несколько приложений в технологии разреженных матриц. То, которое мы рассмотрим в этом параграфе, связано с приведением разреженной матрицы общего вида к блочной нижней треугольной форме посредством несимметричных перестановок (см. обсуждение в § 5.8). Задача формулируется следующим образом: заданы ациклический орграф $G = (V, E)$ и две его вершины s, $t \in V$; найти максимальное множество путей из s в t таких, что ни одна вершина, кроме s и t, не принадлежит более чем одному пути. Будем говорить, что множество *максимально* в смысле некоторого свойства, если для его элементов это свойство выполнено, и оно (множество) не включается строгим образом ни в какое другое множество элементов, имеющих данное свойство. С другой стороны, *наибольшим* будем называть множество с наибольшим числом элементов. На рис. 5.7 приведен пример. Пути 1, 2, 4, 6, 7 и 1, 3, 5, 7 составляют максимальное множество путей с различными вершинами, ведущих из 1 в 7. Число элементов этого множества равно 2. Множество, образованное единственным путем 1, 2, 5, 7, также максимально (хоть и состоит из одного элемента), потому что нельзя найти большее множество путей с различными вершинами, которое включало бы в себя путь 1, 2, 5, 7.

Использование поиска в глубину для решения обсуждаемой задачи было предложено в [Hopcroft, Karp, 1973]. Если поиск начинается с вершины s и существует хотя бы один путь из s в t, то t принадлежит дереву с корнем в s. Поэтому достаточно исследовать только это дерево. Алгоритм пользуется стеком (определение и терминологию см. в § 1.2), где хранится последовательность вершин от s до текущей вершины. Каждое ребро исследуется только один раз, и либо оно становится частью конструируемого пути, либо не существует пути из s в t, содержащего

[1]) Для обсуждаемого примера все вершины G принадлежат подструктуре и, следовательно, она совпадает со структурой уровней. — *Прим. перев.*

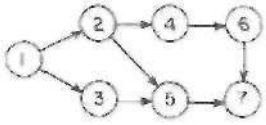


Рис. 5.7. Пути в ациклическом графе.

это ребро. Рассматриваются только ребра, ведущие к непосещавшимся вершинам, потому что мы ищем пути, у которых кроме концов s и t нет общих вершин. Так как возможно, что алгоритму придется исследовать несколько ребер, ведущих к t, принимается соглашение о том, что t не должна становиться посещенной вершиной. Если ход поиска приведет к вершине t, она опускается в стек, который в этом случае содержит полный путь из s в t. Все вершины поднимаются из стека и печатаются; затем в стек снова опускается s, и начинается поиск нового пути. Алгоритм заканчивает работу, когда s поднимается из стека после того, как были исследованы все выходы из s. Пусть V_v — множество посещенных вершин, E_e — множество исследованных ребер. Пусть граф представлен своей структурой смежности (см. § 1.4). Тогда алгоритм можно описать следующим образом:

Шаг 0 (Инициализация). Положить $V_v \leftarrow \varnothing$, $E_e \leftarrow \varnothing$.

Шаг 1 (Начало пути). Опустить s в стек.

Шаг 2 (Посещение вершины). Положить $v \leftarrow$ вершина наверху стека.

Шаг 3 (Исследование ребра). Найти в списке смежности вершины v ребро ($v \rightarrow w$), E_e, где $w \notin V_0$. Тогда:

(3а) Если такое ребро нашлось и $w \neq t$, положить $V_v \leftarrow V_v \cup \{w\}$, $E_e \leftarrow E_e \cup (v \rightarrow w)$, опустить до в стек и перейти к шагу 2 для продолжения поиска.

(3б) Если такое ребро нашлось и $w = t$, положить $E_e \leftarrow E_e \cup (v \rightarrow w)$, опустить w в стек и перейти к шагу 5 для сборки пути.

(3в) Если v не имеет неисследованных выходов, ведущих к непосещавшимся вершинам, перейти к шагу 4 для возвращения.

Шаг 4 (Возвращение). Поднять стек на одну позицию. Если стек пуст, то остановка. В противном случае перейти к шагу 2.

Шаг 5 (Формирование пути). Поднять все содержимое стека, напечатать путь, а затем перейти к шагу 1, чтобы начать новый путь.

Этот алгоритм описан в [Hopcroft, Karp, 1973]. Отметим, что на стр. 230 этой работы пропущен оператор DELETE; он должен быть поставлен сразу после строки FIRST = первый элемент LIST (TOP). Алгоритм требует памяти и времени $O(|V|, |E|)$.

Чтобы показать на примере работу алгоритма, построим для ациклического графа на рис. 5.7 множество путей, ведущих из $s = 1$ в $t = 7$ и имеющих различные внутренние вершины. В стек опускается вершина 1, затем при последовательных исполнениях

шагов 2, 3 и 3а опускаются, скажем, вершины 2, 4 и 6. Теперь на *шаге 2* $v = 6$, а на шаге 3 $w = t = 7$. В стек опускается вершина 7, и управление передается от шага 3 к шагу 5. Здесь стек очищается, и печатается (в обратном порядке) путь 1, 2, 4, 6, 7. Снова опускается вершина 1, и на этот раз прослеживается путь 1, 3, 5, 7. Алгоритм заканчивает работу, так как список смежности вершины 1 исчерпан.

Предположим теперь, что в первой попытке был найден путь 1, 2, 5, 7. Тогда на шаге I опускается вершина s = 1, на шаге 2 полагаем $v = 1$, а на шаге 3 $w = 3$; w опускается на верх стека (см. 3а). Теперь на шаге 2 $v = 3$, но на шаге 3 не удается найти выход из вершины 3, ведущий к нелосещавшейся вершине, поскольку вершина 5 уже посещалась. Управление передается к шагу 4, где поднимается вершина 3. Наконец, на шаге 2 $v = 1$, на шаге 3 не обнаруживается неисследованных выходов, вершина 1 поднимается на шаге 4, и алгоритм заканчивает работу. Заметим, что в этот раз найдено максимальное множество, состоящее только из одного пути, и вершины 4, 6 ни разу не посещались: поиск здесь неполный, поскольку, как указано выше, достаточно исследовать дерево с корнем в $s = 1$.

5.7. ОТЫСКАНИЕ ТРАНСВЕРСАЛИ: АЛГОРИТМ ХОЛЛА

Если у заданной n x n матрицы A все диагональные элементы не равны нулю, то ей можно сопоставить орграф, как это объяснено в § 5.2. Однако если на главной диагонали имеются нули, то вначале переставляются строки и столбцы с тем, чтобы получить ненулевую диагональ. Найденная этим путем последовательность ненулевых элементов называется *трансверсалыо.* Ниже мы покажем, что у невырожденной матрицы A полная трансверсаль всегда существует. Однако полную трансверсаль может иметь и вырожденная матрица. Назовем *структурно вырожденной* матрицу порядка n, которую нельзя переупорядочить так, чтобы получилась полная трансверсаль. Если максимальная возможная трансверсаль имеет длину $k < n$, то k — это *структурный ранг,* а $n - k$ — *структурный дефект* матрицы.

Максимальную трансверсаль матрицы A можно найти посредством алгоритма Холла [Hall, 1956]. Мы опишем здесь вариант с перестановками строк, который удобен для разреженных матриц, хранимых в строчном формате. Вариант с перестановками столбцов обсуждается в [Duff, 1976; Reid, 1977], а версия с поиском в глубину изложена в [Gustavson, 1976b]. Алгоритм состоит из n шагов. Цель k-го шага — поместить ненулевой элемент в k-*ю* позицию диагонали. По завершении k шагов в первых k позициях главной диагонали стоят ненулевые элементы. Теперь на шаге $k + 1$ имеются следующие возможности:

(а) Элемент $a_{k+1,\ k+1} \neq 0$; в этом случае шаг $k + 1$ закончен.

(б) Существует ненулевой элемент, строчный и столбцовый индексы которого находятся в пределах от $k + 1$ до n. В таком случае перестановкой строк и/или столбцов этот элемент можно перевести в $(k + 1)$-ю диагональную позицию. Квадратная подматрица, расположенная на пересечении строк и столбцов с номерами $1, \ldots, k$, не меняется при этих перестановках, и ненулевые элементы в первых k позициях диагонали останутся на своих местах.

(в) Квадратная подматрица, стоящая на пересечении строк и столбцов с номерами $k + 1, \ldots, n$, — нулевая. Мы можем все же надеяться отыскать комбинацию строчных перестановок, помещающую ненулевой элемент в $(k + 1)$-ю диагональную позицию. Чтобы найти требуемые перестановки, проследим в матрице *увеличивающий путь*. Он начинается с позиции $(k + 1, k + 1)$, идет вдоль строки $k + 1$ до ненулевого элемента, стоящего, скажем, в столбце l (хотя бы один такой элемент должен быть, иначе строка $k + 1$ целиком состоит из нулей, и A — вырожденная матрица), затем вдоль столбца l до позиции (l, l), далее к внедиагональному ненулевому элементу строки l, расположенному, скажем, в столбце m, и т. д. Другими словами, путь попеременно проходит через диагональный и внедиагональный ненулевой элементы. Путь не может дважды идти вдоль какой-либо строки или столбца и заканчивается на ненулевом элементе подматрицы, образованной строками $1, \ldots, k$ и столбцами $k + 1, \ldots, n$. В памяти машины путь может регистрироваться записью номеров диагональных позиций в том порядке, в каком они посещались, т. е. $k + 1, l, m, \ldots$. Поскольку некоторые диагональные позиции могли быть посещены, а впоследствии удалены из пути, нужно завести и список посещавшихся позиций, чтобы не войти в них снова. Ниже будет приведен соответствующий пример.

Чтобы построить нужный путь, мы начинаем с позиции $(k + 1, k + 1)$ и ищем в строке $k + 1$ ненулевой элемент. Для невырожденной матрицы A такой должен найтись, потому что в противном случае строка $k + 1$ целиком состояла бы из нулей. Если ненулевой элемент стоит в столбце l, то мы переходим к строке l и ищем в ней ненулевой элемент в столбцах $k + 1, \ldots, n$. Если такой имеется, то путь закончен; если нет, то мы ищем в первых k позициях строки l ненулевой элемент, который бы стоял в столбце, ранее не посещавшемся, скажем, в столбце m. Далее мы переходим к строке m и повторяем вышеописанные операции, пока путь не будет завершен. Если в некоторой строке не удается найти внедиагональный ненулевой элемент в непосещавшемся столбце, то эта строка удаляется из пути (но не из списка посещавшихся позиций), и мы возвращаемся к предыдущей строке. Если на некотором шаге после посещения r позиций в пределах от 1 до k наш путь становится пустым (т. е. мы вернулись к на-

чальному пункту), это означает, что A вырождена: действительно, мы нашли $r + 1$ строк (r посещавшихся строк плюс строка $k + 1$), имеющих ненулевые элементы только в r столбцах.

Как только искомый путь построен (скажем, $k + 1$, l_1, l_2, ..., l_r, где $l_r > k$), мы переставляем $r + 1$ строк и два столбца, чтобы вывести последний найденный ненулевой элемент в позицию ($k + 1$, $k + 1$). Нужны такие перестановки строк:
строка $k + 1$ становится строкой l_1
строка l_1 становится строкой l_2,

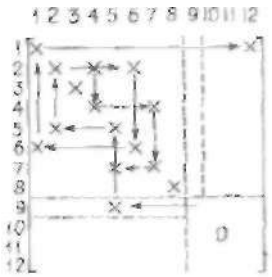


Рис. 5.8. Алгоритм Холла для отыскания трансверсали.

строка l_{r-1} становится строкой $k + 1$.

Поскольку строки выбирались так, чтобы ненулевой элемент строки l_i стоял в позиции l_{i+1}, то при замене строки l_i строкой l_{i+1} этот ненулевой элемент попадет в диагональную позицию (l_{i+1}, l_{i+1}). Следовательно, после строчных перестановок элементы в первых k позициях диагонали останутся ненулевыми. Кроме того, последний ненулевой элемент пути будет переведен в позицию ($k + 1$, l_r). Перестановка столбцов $k + 1$ и l_r перемещает его в позицию ($k + 1$, $k + 1$), чем ($k + 1$)-й шаг завершается. Разумеется, если $l_r = k + 1$, то перестановка столбцов не нужна.

Полезно пояснить алгоритм на примере. Рассмотрим матрицу на рис. 5.8, где уже проведены $k = 8$ шагов.

Начинаем с позиции (9, 9) и продолжаем следующим образом: строка 9, столбец 5, строка 5, столбец 2, строка 2, столбец 4, строка 4, столбец 7, строка 7. К этому моменту мы посетили позиции 9, 5, 2, 4 и 7. Единственный внедиагональный ненулевой элемент строки 7 стоит в столбце 5, уже посещавшемся. Это вынуждает нас удалить из текущего пути (но не из списка посещавшихся позиций) 7 и 4 и продолжать от 2 к 6, 1 и, наконец, 12. В результате получен путь 9, 5, 2, 6, 1, 12.

Теперь, когда путь построен, нужно выполнить следующие перестановки:

строка 9 становится строкой 5,
строка 5 становится строкой 2,
строка 2 становится строкой 6,
строка 6 становится строкой 1,
строка 1 становится строкой 9.

При этом диагональ остается ненулевой, а последний ненулевой элемент пути переходит из позиции (1, 12) в позицию (9, 12). Наконец, переставляем столбцы 9 и 12, что выводит ненулевой элемент в позицию (9, 9). Шаг 9 закончен.

Посмотрим теперь, что произойдет, если элемент (6, 1) равен нулю. В этом случае мы обнаружили бы, что строки 9, 5, 2, 4, 7 и 6 имеют ненулевые элементы только в столбцах 5, 2, 4, 7 и 6. Это означает, что A — вырожденная матрица.

Относительно эффективности этого алгоритма заметим, что на каждом шаге каждая строка просматривается самое большое дважды: первый раз при поиске ненулевого элемента, стоящего правее позиции k; второй раз при необходимости найти ненулевой элемент в непосещавшемся столбце с номером от 1 до k. Поэтому если в матрице ζ ненулевых элементов, то на шаге k выполняется $O(z)$ операций, а в алгоритме в целом $O(nz)$ операций. Эта оценка достигается на специально сконструированных примерах [Duff, 1976], но на практике число операций обычно является малым кратным z.

5.8. ОТЫСКАНИЕ ТРАНСВЕРСАЛИ: АЛГОРИТМ ХОПКРОФТА - КАРПА

Алгоритм Хопкрофта—Карпа [Hopcroft, Karp, 1973] находит максимальную трансверсаль разреженной матрицы. Основная идея та же, что и в алгоритме Холла: прослеживаются увеличивающие пути, и от шага к шагу улучшаются назначения. Однако проводится тщательный анализ увеличивающих путей, и их свойства используются, чтобы разбить последовательность назначений на малое число фаз. Внутри каждой фазы одновременно находят много коротких увеличивающих путей; все они имеют одинаковую, причем минимальную длину, и с их помощью улучшается текущее назначение. Этим способом повышается эффективность алгоритма.

Начнем анализ с рассмотрения неориентированного графа $G = (V, E)$ и подмножества $M \subseteq E$ его ребер. Подмножество M называется *паросочетанием* или *назначением*[1]), если никакая вершина G не инцидентна более чем одному ребру из M. Паросочетание с наибольшим кардинальным числом $m = |M|$ называется *наибольшим паросочетанием.* Вершина, не инцидентная никакому ребру из М, называется *свободной.* Путь в графе G, не имеющий повторяющихся вершин, называется *увеличивающим путем* паросочетания M, если оба его конца — свободные вершины, а ребра попеременно принадлежат $E - M$ и M. Увеличивающие пути служат для повышения кардинального числа паросочетания. Если известен увеличивающий путь паросочетания M, то можно получить новое паросочетание M' с числом ребер, на единицу большим, чем у M (т. е. $|M'| = |M| + 1$). Для этого нужно взять все ребра пути, не принадлежащие исходному паро-

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

сочетанию *M,* и добавить все ребра *M,* не вошедшие в увеличивающий путь. Это свойство имеет место для любого увеличивающего пути, какова бы ни была его длина. В особенности нас будут интересовать *кратчайшие увеличивающие пути,* т. е. пути, имеющие наименьшее кардинальное число среди всех увеличивающих путей паросочетания *M.* Чтобы более точно сформулировать сказанное, напомним сперва некоторые обозначения, используемые в теории множеств. Рассмотрим два множества: *S* и *T.* Запись *S—T* обозначает множество элементов *S,* не принадлежащих *T. Таким* образом,

$$S - T = S - (S \cap T). \quad (5.12)$$

Запись S $\oplus$ *T* обозначает симметричную разность множеств *S* и *T.* Элемент принадлежит S $\oplus$ *T,* если он находится либо в S, либо в *T,* но не в *S* $\cap$ *T.* Следовательно,

$$S \oplus T = S \cup T - S \cap T. \quad (5.13)$$

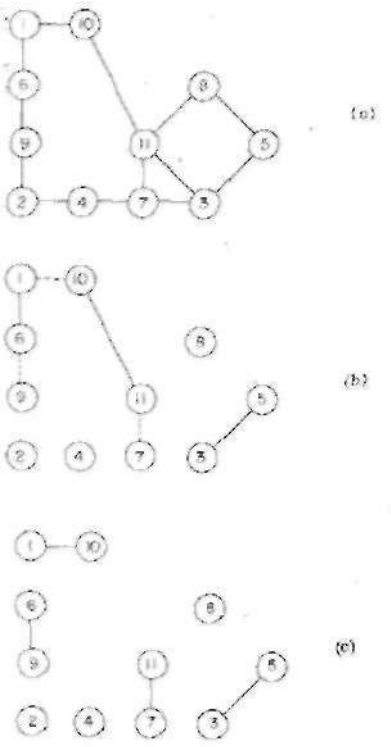


Рис. 5.9. (a) Граф $G = (V, E)$. (b) Паросочетание M и увеличивающий путь P. (c) Увеличенное паросочетание $M \oplus P$.

Пусть теперь *P* обозначает множество ребер увеличивающего пути паросочетания *M* в графе *G.* Тогда $M \oplus P$ также паросочетание, причем $| M \oplus P | = | M | + 1$. Рассмотрим, например, граф рис. 4.1 (Ь), для удобства воспроизведенный на рис. 5.9(a). Паросочетание *M* с 3 ребрами указано на рис. 5.9(b) сплошными линиями. Вершины 9 и 7 — свободные, и путь 9,6, 1, 10, 11, 7 является увеличивающим, поскольку оба его конца свободны, ребра (6, 1) и (10, 11) принадлежат *M,* а ребра (9, 6), (1, 10) и (11, 7) входят в *E,* но не в *M.* На рис. 5.9 (c) показано увеличенное паросочетание $M \oplus P$; оно содержит 4 (= 3 + 1) ребра. Мы могли бы, для получения паросочетания с 4 ребрами, использовать и кратчайший увеличивающий путь, например 2,9 или 4,7 (см. рис. 5.9 (b)).

Продолжим наш анализ. Мы ищем наибольшее паросочетание в графе $G = (V, E)$. Отправляясь от пустого паросочетания $M_0 = \emptyset$, мы будем вычислять последовательность паросочетаний M_0, M_1, M_2, ... с возрастающими кардинальными числами, пока не

будет достигнуто наибольшее паросочетание. Каждое паросочетание M_{i+1} получается из предыдущего паросочетания M_i посредством построения увеличивающего пути P_i. Именно, $M_{i+1} = M_i \oplus P_i$, $i = 0, 1, 2, \ldots$. Если G — ассоциированный с разреженной матрицей двудольный граф, то это в точности та процедура, на которой основан обсуждавшийся в § 5.7 алгоритм Холла. Для матрицы порядка n с числом ненулевых элементов z этот алгоритм требует $O(nz)$ операций.

В отличие от алгоритма Холла, алгоритм Хопкрофта—Карпа использует только кратчайшие увеличивающие пути. Если P_i — кратчайший увеличивающий путь паросочетания M_i и $M_{i+1} = M_i \oplus P_i$, $i = 0, 1, 2, \ldots$, то имеют место следующие утверждения:

(I) $|P_{i+1}| \geq |P_i|$, т. е. кратчайший увеличивающий путь для M_{i+1} не может иметь меньше ребер, чем кратчайший увеличивающий путь для M_i.

(II) Для всех j и k таких, что $|P_j| = |P_k|$, пути P_j и P_k не имеют общих вершин.

(III) Для всех j и k таких, что $j < k$ и $|P_j| = |P_k|$, P_k является кратчайшим увеличивающим путем паросочетания M_j.

(IV) Если m — кардинальное число наибольшего паросочетания, то количество различных целых чисел в последовательности $|P_0|, |P_j|, |P_2|, \ldots$ не может превышать $2m^{1/2} + 2$.

Кроме того, если m — кардинальное число наибольшего паросочетания, а M — произвольное паросочетание, для которого $|M| < m$, то справедливо и

(V) Существует увеличивающий путь длины, не превышающей $2|M|/(m - |M|) + 1$.

В силу этих свойств вычисление последовательности паросочетаний разбивается не более чем на $2m^{1/2} + 2$ фаз. Все увеличивающие пути, разыскиваемые внутри фазы, имеют различные вершины и одинаковую длину. Кроме того, все эти пути — кратчайшие увеличивающие для того паросочетания, с которым начинается данная фаза. Итак, вместо того чтобы вычислять всю последовательность паросочетаний, Хопкрофт и Карп предлагают следующий алгоритм:

Шаг 0 (Инициализация). Положить $M \leftarrow \emptyset$.

Шаг I (Поиск путей). Найти максимальное множество $\{P_1, P_2, \ldots, P_t\}$ кратчайших увеличивающих путей паросочетания M, не имеющих общих вершин. Если таких путей нет, то остановка.

Шаг 2 (Увеличение паросочетания). Положить $M \leftarrow M \oplus P_1 \oplus P_2 \oplus \ldots \oplus P_t$ и перейти к шагу 1.

Рис. 5.10 дает иллюстрацию к применению этого алгоритма для графа рис. 5.9 (а). Пусть M — пустое паросочетание: $M = \emptyset$. Максимальное множество кратчайших увеличивающих путей

с различными вершинами указано на рис. 5.10 пунктирными линиями; все эти пути (их 5) имеют длину I. Если увеличить M в соответствии с описанием шага 2, то сразу будет получено наибольшее паросочетание. Вершина 3 не может быть включена в паросочетание и остается свободной. Для данного примера оказалось достаточно провести только одну фазу, чтобы породить наибольшее паросочетание.

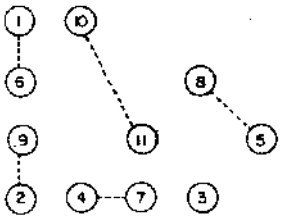


Рис. 5.10. Паросочетания в неориентированном графе рис. 5.9 (а). Пунктирными линиями указано максимальное множество увеличивающих путей с различными вершинами.

Остается показать, как использовать этот алгоритм для разреженных матриц. Пусть A — квадратная разреженная матрица порядка n. Процедура отыскания трансверсали состоит в следующем:

(1) Построить ассоциированный с A двудольный граф $G = (V, E)$ (см. § 5.2). Граф G неориентированный и имеет два множества вершин R и C, по n вершин в каждом. Вершины R называются юношами и соответствуют строкам A; вершины C называются девушками и соответствуют столбцам A. В G имеется ребро (r_i, c_j), если $a_{ij} \neq 0$.

(2) Пусть M — паросочетание в G. Вначале $M = \emptyset$, но в ходе работы алгоритма M становится непустым. Теперь при фиксированном G и текущем M мы приписываем ориентацию ребрам G: каждое ребро из M направлено от юноши к девушке, а каждое из оставшихся ребер G — от девушки к юноше. Пусть $\overrightarrow{G} = (V, \overrightarrow{E})$ — полученный этим способом ориентированный граф. Именно такой способ построения $\overrightarrow{G}$ мотивируется следующим его свойством: ребра любого ориентированного пути в $\overrightarrow{G}$ попеременно принадлежат $\overrightarrow{E} - M$ и M. Поэтому если в $\overrightarrow{G}$ найден путь, связывающий свободную девушку со свободным юношей (имеется в виду свобода по отношению к M), то этот путь необходимо увеличивающий.

(3) Пусть L_0 — множество свободных девушек. Тогда нужно построить подструктуру уровней ориентированной смежности с корнем в L_0, пользуясь поиском в ширину (см. § 5.5). Нет необходимости строить всю подструктуру. Достаточно сохранять ребра, идущие от каждого уровня L_i к уровню L_{i+1}; при этом нужны только уровни $L_0, L_1, \ldots, L_p$, где $p = \min \{i | L_i \cap \cap \{\text{свободные юноши}\} \neq \emptyset\}$. Построенная подструктура представляет собой ациклический орграф с девушками в четных уровнях и юношами — в нечетных. Каждый путь от свободной девушки к свободному юноше — это кратчайший увеличивающий путь для M, причем его длина равна p. Чтобы найти максимальное множество путей с различными вершинами, обращается ориента-

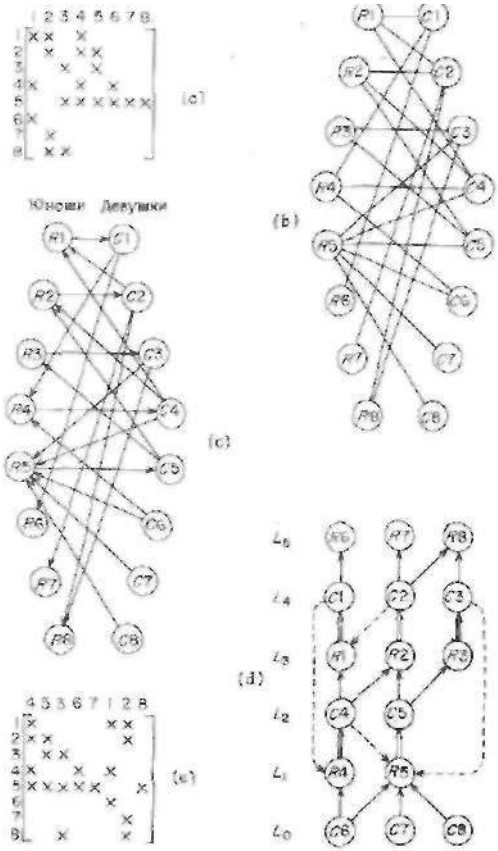


Рис. 5.11. Алгоритм Хопкрофта — Карпа для отыскания трансверсали: (a) портрет разреженной матрицы; (b) двудольный граф; (c) ориентированное паросочетание; (d) подструктура уровней; (e) переупорядоченная матрица.

ция каждого ребра, и на подструктуре, отправляясь от свободных юношей уровня L_p, выполняется поиск в глубину, как это объяснено в § 5.6. Заметим, что если бы ориентация ребер не обращалась, а поиск начинался от свободных девушек уровня L_0, то пришлось бы исследовать много ребер, ведущих к несвободным юношам уровня L_p, что потребовало бы значительно большей вычислительной работы. Наконец, если $\{P_1, P_2, \ldots, P_t\}$ — найденное множество путей, то можно увеличить текущее паросочетание M, и новое паросочетание имеет кардинальное число $|M| + t$.

Рассмотрим, в качестве примера, применение алгоритма к разреженной матрице на рис. 5.11 (a). Соответствующий ей двудоль-

ный граф изображен на рис. 5.11 (b). Поначалу M — пустое паросочетание, все юноши и девушки свободны, и каждое ребро направлено от девушки к юноше. Корнем подструктуры уровней является все множество девушек, и в этой подструктуре только два уровня, В данном конкретном случае всего за одну фазу можно было бы найти наибольшее (с кардинальным числом 7) паросочетание (12, 24, 35, 46, 58, 61, 83); здесь первая цифра каждой пары указывает строку, вторая — столбец. Предположим, однако, что мы были не так удачливы и обнаружили только пять древесных ребер, порождающих паросочетание (11, 22, 33, 44, 55) с кардинальным числом 5. Теперь начинаются вычисления второй фазы. Ребра, представляющие паросочетание, направлены от юношей К девушкам; остальные ребра — *от* девушек к юношам (см. рис. 5.11 (c)). За корень берется множество, состоящее из свободных девушек $C6$, $C7$ и $C8$, строится подструктура уровней и сохраняются только те ребра, что идут от уровня L_i к уровню L_{i+1}. Результат построения показан на рис. 5.11 (d). Ребра, принадлежащие подструктуре, указаны сплошными линиями, а те ребра, которые, помимо подструктуры, принадлежат и паросочетанию, *отмечены* двойными линиями. *Остальные ребра двудольного* графа проведены в иллюстративных целях пунктирными линиями, но сейчас они не представляют для нас интереса. Теперь обращается ориентация каждого ребра, и проводится, отправляясь от $R6$, $R7$ и $R8$, поиск в глубину. Пусть на этот раз нам сопутствовала удача, и мы нашли следующее максимальное множество путей с различными вершинами: ($R6$, $C1$, $R1$, $C4$, $R4$, $C6$) и ($R7$, $C2$, $R2$, $C5$, $R5$, $C7$), оба имеют длину 5. Тогда новым паросочетанием с кардинальным числом 7 будет (14, 25, 33, 46, 57, 61, 72); снова первая цифра каждой пары указывает строку, вторая — столбец. Соответствующая переупорядоченная матрица изображена на рис. 5.11 (e). Это наибольшее паросочетание, так как для него нельзя найти увеличивающий путь.

Если разреженная матрица порядка η имеет г ненулевых элементов, то алгоритм, строящий внутри фазы максимальное множество путей, требует памяти и времени $O(n, z)$ [1]). Поскольку кардинальное число наибольшего паросочетания не превышает n, то фаз будет не больше чем $2n^{1/2} + 2$. Следовательно, весь алгоритм вычисления наибольшего паросочетания требует времени не более $O(n^{3/2}, zn^{1/2})$ и памяти не более $O(n, z)$. Если $z = O(n)$, что часто выполняется для разреженных матриц, то эти оценки превращаются в $O(n^{3/2})$ для времени и $O(n)$ для памяти.

Изложенный выше алгоритм был разработан для двудольных графов. В [Even, Kariv, 1975] он обобщен на ориентированные графы. Несколько более простой и эффективный алгоритм для

[1]) То есть $O(n) + 0(z)$. — *Прим. перев.*

той же задачи предложен в [Micali, Vazirani, 1979]; его временная сложность $O(|V|^{1/2} \cdot |E|)$. Как выяснилось, для многих практических задач алгоритм Холла, несмотря на худшую асимптотическую оценку, работает быстрее, чем алгоритм Хопкрофта—Карпа.

5.9. АЛГОРИТМ САРДЖЕНТА - УЭСТЕРБЕРГА ДЛЯ ОТЫСКАНИЯ СИЛЬНЫХ КОМПОНЕНТ ОРГРАФА

В § 5.1 отмечалось, что несимметричная разреженная матрица, имеющая полную трансверсаль (т. е. диагональ без нулей) может быть симметрично переупорядочена к блочной нижней треугольной форме, если известны сильные компоненты соответствующего орграфа. Сарджент и Уэстерберг [Sargent, Westerberg, 1964] предложили простой алгоритм отыскания сильных компонент произвольного орграфа. Алгоритм основан на процедуре поиска в глубину, поскольку в нем прослеживается путь при сохранении насколько это возможно последовательности вершина—ребро—вершина—ребро... Путь начинается из произвольно взятой вершины и продолжается до тех пор, пока не будут обнаружены либо цикл, либо вершина без выхода.

Цикл опознается благодаря тому, что путь входит в вершину, посещавшуюся прежде. Как известно, все вершины цикла принадлежат одной и той же сильной компоненте. В случае обнаружения цикла применяется процедура, называемая *стягиванием вершин* [1]): все вершины цикла стягиваются в единую *составную вершину,* все ребра цикла игнорируются, а ребра, соединяющие вершины цикла с прочими вершинами графа, рассматриваются теперь как соединяющие составную вершину с этими прочими вершинами. Составная вершина становится последней вершиной текущего пути, и в модифицированном графе поиск возобновляется с этого места.

Если найдена вершина, или составная вершина, не имеющая выхода, то эта вершина или составная вершина является сильной компонентой уровня 1. Вершина или все вершины, образующие составную, удаляются из графа вместе со всеми инцидентными им ребрами. Поиск затем возобновляется в оставшемся подграфе, причем отправляются от последней вершины текущего пути либо от новой начальной вершины, если путь исчерпан. Так продолжается до тех пор, пока не будут посещены все вершины и исследованы все ребра исходного графа.

В этом алгоритме каждое ребро графа исследуется только один раз. .Однако необходимы частые перестройки графа, что состав-

[1]) В оригинале — vertex collapsing. — *Прим. перев.*

ляет серьезную дополнительную работу. Стягивание вершин при обнаружении очередного цикла эквивалентно формированию нового факторграфа, а удаление сильной компоненты — выделению частичного графа. Приходится вести записи, указывающие, из каких вершин состоит каждая составная вершина и какой составной вершине принадлежит каждая вершина исходного графа. Неудачная реализация может повлечь $O(|V|^2)$ перепомечиваний вершин (V — множество вершин графа $G = (V, E)$). В [Munro, 1971a] предлагается проводить стягивание вершин, перепомечивая только составную вершину с наименьшим числом составляющих; в [Munro, 1971b] показано, что модифицированный алгоритм требует времени $O(|V| \log |V|, |E|)$. Если применить к алгоритму Сарджента—Уэстерберга соображения Тарьяна [Tarjan, 1975] по поводу операций (Объединение и Поиск) с непересекающимися множествами (см. § 1.13), то оценку временной сложности можно снизить до $O(|V| \log^* |V|, |E|)$, где

$$\log^* a = \min \{i \mid \overbrace{\log \log \ldots \log}^{i \text{ раз}} (a) \leqslant 1\}. \tag{5.14}$$

Функция $\log^* a$ растет очень медленно при возрастании a; например, $\log^* a < 4$, если $a < 3 \times 10^6$. Это означает, что для практических задач алгоритм линеен относительно $|V|$ и $|E|$. Алгоритм Сарджента—Уэстерберга излагается в [Reid, 1977; Duff, Reid, 1978a]; в последнюю работу включена четкая блок-схема метода.

Рассмотрим применение этого алгоритма к графу на рис. 5.1 (b). Если начать, например, с вершины 2, то может быть построен путь 2, 3, 4, 9, 10, 11, 15, 14,9. Обнаруживается цикл, так как вершина 9 встречается повторно. Принадлежащие циклу вершины 9, 10, 11, 15 и 14 стягиваются, и формируется составная вершина, скажем A. Когда поиск возобновляется, то выясняется, что вершина в конце пути, т. е. в точности вершина A, не имеет выхода и, следовательно, является сильной компонентой уровня 1. Вершины 9, 10, 11, 15 и 14 удаляются из графа и помещаются в первую компоненту, скажем C_1. Поиск продолжается, исходя из вершины 4, которая снова не имеет выхода и становится компонентой C_2. Остается путь 2, 3. При поиске от вершины 3 выявляется цикл 2, 3, 8, 7, 5, 1, 2, порождающий составную вершину B. Продолжая поиск от вершины B, найдем цикл B, 6, B, не имеющий выхода и составляющий компоненту C_3. Наконец, поскольку теперь путь исчерпан, выбираем вершину 12 в качестве новой начальной вершины, прослеживаем цикл 12, 13, 12 и определяем компоненту C_4.

5.10. АЛГОРИТМ ТАРЬЯНА ДЛЯ ОТЫСКАНИЯ СИЛЬНЫХ КОМПОНЕНТ ОРГРАФА

При поиске в глубину на орграфе $G = (V, E)$ формируется остовной лес (см. § 5.4), состоящий из одного или нескольких деревьев. Вершины каждой сильной компоненты C орграфа определяют поддерево некоторого дерева из леса. Поддерево содержит все вершины C и только вершины С; его корень — это первая вершина С, посещаемая в холе поиска. Таким образом, задача отыскания сильных компонент сводится к отысканию корней поддеревьев в остовном лесу, порождаемом поиском в глубину.

Алгоритм Тарьяна [Tarjan, 1972] сконструирован таким образом, чтобы возвращение через корень компоненты С в другую компоненту более высокого уровня было невозможно до тех пор, пока все вершины C не будут посещены и опознаны как элементы C. Это означает, в частности, что если v — текущая вершина поиска и найдено ребро $(v \rightarrow w)$, где w нумерована, но не приписана какой-либо компоненте, то v и w принадлежат одной и той же компоненте. Доказательство несложно: пусть Уровень $(w) = l$; так как до еще не приписана, то текущая вершина v должна находиться в уровне $l' \leq l$. Однако поскольку $(v \rightarrow w)$ — ребро графа G, то $l' \geq l$. Следовательно, $l' = l$, что возможно, лишь если v и до относятся к одной компоненте.

В алгоритме Тарьяна проводится поиск, и каждой вершине v сопоставляется метка. Последняя состоит из двух чисел, называемых Номер (v) и Связка (v) [1]). Номер (v) определяется в ходе поиска простой нумерацией вершин от 1 до $n = |V|$ соответственно порядку, в каком они посещаются. Для произвольной вершины v, не являющейся корнем сильной компоненты. Связка (v) — это номер некоторой вершины до из той же, что и для v, сильной компоненты; однако до нумерована раньше, чем v, т. е. Связка (v) = Номер (w) < Номер (v). Для корня r любой сильной компоненты мы, напротив, полагаем Связка (r) = Номер (r). Это свойство используется для опознания корней сильных компонент: если число Связка (v) правильно вычислено, то v тогда и только тогда будет корнем сильной компоненты, когда Связка (v) = = Номер (v). В ходе поиска вычисления, связанные с признаком Связка, производятся на шагах следующих трех типов:

Шаг 1 (Инициализация). Если вершина v посещается впервые, то положить Связка (v) = Номер (v).

Шаг 2 (Модификация). Если v — текущая вершина и найдено ребро $(v \rightarrow w)$, где до нумерована раньше, чем и, но еще не приписана какой-либо компоненте, то положить Связка (v) = min [Связка (v), Связка (w)].

[1]) В оригинале — Lowlink (v). — *Прим. перев.*

Шаг 3 (Возвращение). Если $(v \to w)$—древесное ребро, причем и w принадлежат одной компоненте и происходит возвращение из w в v, то положить Связка (v) = min [Связка (v), Связка (w)].

На шаге 1 признаку Связка присваивается максимальное возможное для данной вершины значение. На шаге 2 Связка (v) переопределяется на меньшее значение, если найдена связь v с некоторой подходящей вершиной w. Вследствие требования Номер (w) < Номер (v) здесь учитываются только обратные и поперечные ребра. Кроме того, поскольку w нумерована, но еще не приписана [1]), то, как мы знаем, v и w должны быть в одной и той же компоненте. Это устраняет «стерильные» поперечные ребра, т. е. поперечные ребра $(v \to w)$, у которых v и w относятся к разным компонентам [Gustavson, 1976b]. Далее, Связка (w) — это номер некоторой вершины из той сильной компоненты, какой принадлежат v и w; этот номер, возможно, меньше, чем у w. Стратегия шага 2 состоит в том, чтобы переопределять признак Связка (v) на наименьший из известных пока номеров вершин в данной компоненте.

Наконец, на шаге 3 минимальное значение признака Связка спускается вдоль пути при возвращении. Стратегия этого шага та же, что у шага 2: мы знаем, что v связана с w, а w связана с x= Связка (w) [2]); поэтому v связана с x. Если номер x меньше, чем текущее значение числа Связка (v), то последнее переопределяется. Стоит отметить, что алгоритм будет работать правильно и при ослаблении требования о том, чтобы v и w принадлежали одной компоненте. Так сделано, например, в реализации Густавсона [Gustavson, 1976b]. Действительно, если $(v \to w)$ — древесное ребро, а v и w относятся к разным компонентам, то w — корень своей компоненты. Поскольку имеет место возвращение, то Связка (w) = Номер (w) > Номер (v). Так как Связка $(v) \leq$ $\leq$ Номер (v), то значение признака Связка (v) в данном случае шагом 3 не будет изменено. Разумеется, во избежание лишних проверок есть смысл использовать информацию, какая у нас имеется.

Теперь нужно показать, что признак Связка вычисляется верно. Пусть v — некоторая вершина, относящаяся к сильной компоненте C, и пусть T_v — поддерево с корнем v, содержащее потомство v в С. Если v не является корнем C, то T_v должно иметь выход в эту компоненту, т. е. должно существовать ребро $(w \to x)$, где $w \in T_0$, $x \in C$, причем это ребро недревесное. Следовательно, x должна быть нумерована раньше, чем w. Более того,

[1]) Какой-либо компоненте. — *Прим. перев.*

[2]) Имеется в виду равенство номеров. — *Прим. перев.*

x должна быть нумерована раньше, чем v, чтобы ребро $(w \rightarrow x)$ могло быть выходом из T_v; поэтому Номер (x) < Номер (v). Ребро $(w \rightarrow x)$ было найдено в ходе поиска, так что Связка $(w) \leq$ $\leq$ Номер (x). Позже, при возвращении, это значение было спущено к v, откуда Связка $(v) \leq$ Номер (x) < Номер (v). Это как раз тот тест, который определяет, что v не является корнем C. Заметим, что полученный результат будет справедлив независимо от того, полагаем мы Связка (w) = Номер (x) или Связка (w) — = Связка (x) в момент обнаружения ребра $(w \rightarrow x)$. Первый случай соответствует первоначальной стратегии Тарьяна и Густавсона. Вторая стратегия, примененная в [Duff, Reid, 1978a], подразумевает использование возможной информации о наличии у x еще более низкой связки в С и переопределение числа Связка (w) на наименьшее значение, какое мы имеем в данный момент. Так как Связка $(x) \leq$ Номер (x), то эта стратегия может помочь избежать некоторых присваиваний признаку Связка.

Если v — корень сильной компоненты, то T_v не имеет выхода, и Связка (v) = Номер (v). В джунглях, изображенных на рис. 5.5 (a), вершины, являющиеся корнями сильных компонент, обведены двойными кружками; на рис. 5.5 (b) приведена таблица значений признаков Номер и Связка.

Алгоритм Тарьяна по существу совпадает с алгоритмом Сарджента—Уэстерберга. Главное отличие заключается в том, что вместо понятия «стягивание вершин», использовавшегося Сарджентом и Уэстербергом, в алгоритме Тарьяна вводится признак Связка, позволяющий хранить составные вершины в стеке. Действительно, если Связка $(v) = w$, то все вершины, находящиеся в стеке между (также помещенными туда) вершинами w и v, образуют составную вершину.

Теперь мы приведем развернутую формулировку алгоритма Тарьяна. Полный алгоритм представляет собой комбинацию поиска в глубину и описанной выше процедуры вычисления признака Связка. Используются два стека (определение стека см. в § 1.2); мы будем придерживаться терминологии Тарьяна и называть их соответственно «стеком» и «маршрутом». Маршрут содержит список вершин, образующих путь от начальной вершины к текущей. Каждая новая вершина, найденная при поиске, опускается в маршрут; каждый раз, когда выполняется возвращение, соответствующая вершина поднимается с верха маршрута. Таким образом, список вершин, хранимых в маршруте, — это в действительности путь в теоретико-графовом смысле.

Стек содержит список вершин, принадлежащих частично сформированным сильным компонентам. Каждая новая вершина, найденная при поиске, опускается в стек. Вершины не удаляются из стека индивидуально. Когда сильная компонента в верхней части стека сформирована полностью, она удаляется вся.

Алгоритм использует также массивы Номер и Связка, булевский массив для проверки за фиксированное время, находится ли вершина в стеке, и представления множества V_v посещенных вершин и множества E_c исследованных ребер. Точная формулировка алгоритма такова:

Шаг 0 (Инициализация). Положить $E_e \leftarrow \emptyset$, $V_v \leftarrow \emptyset$, $i \leftarrow 0$.

Шаг 1 (Выбор начальной вершины). Выбрать произвольную вершину $v \notin V_v$. Если такой нет, то — остановка.

Шаг 2 (Посещение вершины). Опустить v в стек и маршрут. Положить $V_v \leftarrow V_v \cup \{v\}$, $i \leftarrow i + 1$, Номер $(v) \leftarrow i$, Связка $(v) \leftarrow i$.

Шаг 3 (Исследование ребра). Найти в списке смежности вершины v ребро $(v \rightarrow w) \notin E_e$. Возможны следующие случаи:

(3а) Найдено ребро $(v \rightarrow w)$, причем $w \notin V_v$, т. е. $(v \rightarrow w)$ — древесное ребро. Положить $E_e \leftarrow E_e \cup (v \rightarrow w)$, $v \leftarrow w$ и перейти к шагу 2.

(3б) Найдено ребро $(v \rightarrow w)$, причем $w \in V_v$. Положить $E_e \leftarrow E_e \cup (v \rightarrow w)$ и перейти к шагу 4 для пересчета признака Связка (v).

(3в) Вершина v не имеет неисследованных выходов и Связка $(v) <$ Номер (v). Перейти к шагу 5 для возвращения.

(3г) Вершина v не имеет неисследованных выходов и Связка $(v) =$ Номер (v). Перейти к шагу 6 для сборки сильной компоненты.

Шаг 4 (Пересчет признака Связка). Если Номер $(w) <$ Номер (v) и w находится в стеке, то положить Связка $(v) \leftarrow \min$ [Связка (v), Связка (w)] и перейти к шагу 3. В противном случае, сразу перейти к шагу 3.

Шаг 5 (Возвращение). Поднять v из маршрута. Положить: $u \leftarrow$ вершина на верхе маршрута, Связка $(u) \leftarrow \min$ [Связка (u), Связка (v)], $v \leftarrow u$. Перейти к шагу 3.

Шаг 6 (Формирование сильной компоненты). Поднять v и все вершины над v из стека и поместить их в текущую сильную компоненту. Поднять v из маршрута.

Если маршрут пустой, перейти к шагу 1. В противном случае положить $v \leftarrow$ вершина на верхе маршрута и перейти к шагу 3.

Этот алгоритм требует памяти и времени $O(|V|, |E|)$. Реализация алгоритма, обеспечивающая внушительную экономию памяти, описана в [Gustavson, 1976b]; *там* же приведен тщательный подсчет числа операций (оператор 8с на стр. 287 этой статьи содержит опечатку; правильная его запись: $V \leftarrow W$). Программа Густавсона работает непосредственно с неупорядоченным строч-

ным представлением разреженной матрицы, которое, как показано в § 1.8, эквивалентно структуре смежности соответствующего орграфа. Программа требует, чтобы все петли, т. е. ребра вида $(v \rightarrow v)$, хранились в явном виде, как и будет в случае полного строчного представления матрицы. Это требование эквивалентно предположению, что известна полная трансверсаль и матрица переупорядочена (или может быть переупорядочена) таким образом, что все диагональные элементы не равны нулю. Это предположение неявно присутствует во всех наших обсуждениях поиска в глубину.

Другая эффективная реализация описана в [Duff, Reid, 1978b]; она включает в себя рассмотренную выше улучшенную стратегию пересчета признака Связка. Даны также анализ сложности и сравнение с алгоритмом Сарджента—Уэстерберга.

Рассмотрим применение алгоритма Тарьяна к графу на рис. 5.1 (b). Соответствующие джунгли показаны на рис. 5.5 (a). За начальную выбирается вершина 2 и прослеживается путь 2, 3, 4, 9, 10, 11, 15, 14. Номера, получаемые вершинами, указаны рядом с кружками. Вначале для каждой вершины значение Связка полагается равным значению Номер. Затем выявляется обратное ребро (14 → 9), и Связка (14) исправляется на значение Связка (9) — 4. Поскольку других выходов из вершины 14 нет, то происходит возвращение сначала к вершине 10, где устраняется ребро (10 → 14), а потом к вершине 9. При возвращении признак Связка для вершин 15, 11 и 10) переопределяется на 4; это означает, что ни одна из них не является корнем какой-либо компоненты. Однако, когда текущей вершиной становится вершина 9, выясняется, что Связка (9) = Номер (9) = 4, т. е. 9 — корень компоненты. Сама компонента состоит из вершин 9, 10, 11, 15 и 14, находящихся сейчас в верхней части стека. Это множество удаляется из стека для формирования компоненты.

Возвращаясь к вершине 4, устанавливаем, что Связка (4) = Номер (4) = 3; следовательно, 4 — это корень еще одной сильной компоненты. Вершина 4 удаляется с верха стека, и возвращение продолжается теперь к вершине 3, где выявляется путь 8, 7, 5, 1, образуемый древесными ребрами. Вершины этого пути нумеруются, признаку Связка присваиваются начальные значения. Далее находим ребро (1 → 2), и исправляем значение Связка (1) на 1. При возвращении эта единица передается вдоль пути вплоть до вершины 8. Когда текущей становится вершина 8, текущий путь сокращается до 2, 3, 8, но в стеке остаются 2, 3, 8, 7, 5, 1. Ребро (8 → 9) устраняется, потому что вершина 9, хотя и посещалась, не присутствует в стеке. Затем обнаруживаем ребро (8 → 6) и присваиваем признакам Связка (6) и Номер (б) значение 13. После этого регистрируется поперечное ребро (6 → 5), активное, так как вершина 5 посещалась и находится в стеке. Поэтому

Связка (6) исправляется на Связка (5) = 1. Возвращаясь к вершине 2, выбираем из верхней части стека сильную компоненту 2, 3, 8, 7, 5, 1, 6.

Теперь, когда построение первого дерева закончено, маршрут и стек опустели. Но посещались пока еще не все вершины. Если в качестве новой начальной вершины взять вершину 12, то будет получено второе дерево, как легко может проверить читатель.

5.11. СТРАТЕГИИ ВЫБОРА ГЛАВНЫХ ЭЛЕМЕНТОВ ДЛЯ НЕСИММЕТРИЧНЫХ МАТРИЦ

В этом параграфе мы изучим стратегии выбора главных элементов, имеющие целью локальную минимизацию заполнения и числа операций при гауссовом исключении для разреженной матрицы общего вида. Методы данного параграфа приложимы к любой матрице A, но если A большая и известно (или есть основания полагать), что она разложима, то почти наверное имеет смысл вначале привести ее к блочной нижней треугольной форме, а уже затем использовать эти методы для каждого блока по отдельности. Все методы являются вариантами стратегии, первоначально предложенной Марковицем [Markowitz, 1957]. Стратегия Марковица заключается в том, чтобы на каждом шаге брать в качестве главного ненулевой элемент, для которого минимально произведение числа остальных ненулевых элементов той же строки и числа остальных ненулевых элементов того же столбца активной подматрицы. Этот простой эвристический алгоритм в сочетании с каким-нибудь хорошим критерием устойчивости дает прекрасные результаты, значительно лучшие, чем у других, более сложных схем.

Алгоритм Марковица локален в том смысле, что минимизирует на каждом шаге заполнение и число операций, не считаясь с возможностью, что другая последовательность предыдущих главных элементов могла бы породить еще меньшее общее заполнение и потребовать меньшего общего числа операций.

Идею стратегии Марковица, ее вариантов и других родственных методов легче всего понять с помощью следующей картины гауссова исключения. В начале k-го шага мы принимаем за главный элемент $a_{kk}^{(k)}$ и нормируем строку k активной подматрицы делением всех ее элементов на $a_{kk}^{(k)}$. Как обычно, активной называется подматрица, стоящая на пересечении строк и столбцов с номерами $k, \ldots, n$. Ситуация в этот момент иллюстрируется рисунком 5.12 для случая $n = 7$, $k = 3$. Векторы r и c имеют размерность $n - k$. Рассмотрим матрицу cr^{T}; это квадратная матрица порядка $n-k$ и ранга 1 (см. § 2.2), и k-й

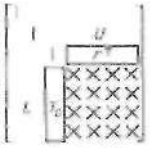


Рис. 5.12. Гауссово исключение для матрицы общего вида; порядок $n = 7$, шаг $k = 3$.

шаг гауссова исключения состоит в вычитании cr^T из подматрицы, расположенной в строках и столбцах $k + 1, \ldots, n$. Вектор c становится k-м подстолбцом L, а вектор $(1, r^T)$ — k-й строкой U[1]). Пусть $n(v)$ обозначает число ненулевых элементов вектора v. Стратегию Марковица можно теперь переформулировать следующим образом: выбрать в активной подматрице и перевести в позицию (k, k) посредством соответствующих перестановок строк и столбцов тот ненулевой элемент, для которого произведение $n(c)\, n(r)$ минимально. Так как $n(c)\, n(r)$ — это число ненулевых элементов матрицы cr^T, то становится понятно, в каком смысле стратегия Марковица пытается локально уменьшить заполнение.

Следует сделать три замечания. Можно рассудить, что произведение $n(c)\, n(r)$ минимально, когда минимален каждый из сомножителей, так что было бы достаточно выбрать на роль главного элемент, стоящий в строке и столбце с наименьшим числом ненулевых элементов. Такой элемент, однако, может оказаться нулем и, следовательно, непригодным в качестве главного. В действительности, во многих случаях все шансы за то, что такой элемент будет нулевым как раз потому, что он находится в строке и столбце с наибольшим числом нулей. Ситуация становится еще хуже, когда в расчет принимаются соображения численной устойчивости, так как теперь отвергаются не только нули, но и слишком малые ненулевые элементы. Однако все же верно то, что приемлемый ненулевой элемент, минимизирующий произведение $n(c)\, n(r)$, часто соответствует *малым* $n(c)$ и $n(r)$. Учтем, что $n(c)$ и $n(r)$ — это в точности число внедиагональных ненулевых элементов соответственно в k-м столбце L и k-й строке U. Это проясняет то обстоятельство, что стратегия Марковица по существу пытается сохранить разреженность L и U.

Пусть C — множество позиций ненулевых элементов в матрице cr^T; D — аналогичное множество для подматрицы, расположенной в $A^{(k)}$ в строках и столбцах $k + 1, \ldots, n$ (см. рис. 5.12). Обычно $C \cap D \neq \emptyset$, поэтому множество F элементов заполнения, внесенного в $A^{(k+1)}$ на k-м шаге, выражается формулой

$$F = C - C \cap D. \tag{5.15}$$

Второе наше замечание относится к тому, что стратегия Марковица минимизирует $|C| = n(c)\, n(r)$, но не $|F|$. Однако минимизация $|C|$ имеет тенденцию уменьшать к $|F|$. Предлагалось основывать выбор главного элемента на требовании минимизации $|F|$ [Berry, 1971], но такой подход связан с очень большой вычислительной работой, поскольку предполагает определение числа $|F|$ для каждого из кандидатов. Кроме того, он не дает лучших результатов, чем простая стратегия Марковица [Duff, Reid,

[1]) Считая от главной диагонали. — *Прим. перев.*

1974]. С другой стороны, число $|C| - n\ (c)\ n\ (r)$ равно числу умножений и числу сложений (если засчитывать сложения с нулями), выполняемых на *k-м* шаге. Следовательно, метод Марковица локально минимизирует число умножений и сложений. Число делений за шаг равно $n\ (r)$; как объяснено выше, это число обычно мало.

Наше третье замечание касается вопроса об устойчивости. Стратегия Марковича сама по себе не гарантирует численной устойчивости. Типичным является следующее правило, сочетающее стратегию Марковица с одним из критериев устойчивости, обсуждавшихся в § 3.6: на каждом шаге среди всех ненулевых элементов активной подматрицы, удовлетворяющих принятому критерию устойчивости, выбрать в качестве главного тот, для которого минимально произведение $n\ (c)\ n\ (r)$. Стандартный критерий устойчивости — это отбор по барьеру. В терминах множеств, определенных в § 3.6: S_{pc} содержит все ненулевые элементы активной подматрицы; S_{st} — те элементы из S_{pc}, которые подчиняются принятому критерию устойчивости; S_{sp} — это подмножество S_{st}, выделяемое условием Марковица. В общем случае S_{sp} может содержать несколько ненулевых элементов. Наконец, главный элемент — это любой элемент из $S_{piv} \equiv S_{sp}$.

Как указано Даффом [Duff, 1981a], одна из сильных сторон стратегии Марковица — ее симметрия. Последняя означает, что (если не учитывать влияние критерия устойчивости и неоднозначности выбора) для обеих матриц A и A^T получается одно и *го* же упорядочение. Стоимость алгоритма Марковица существенно зависит от размера множества S_{st} на каждом шаге, поскольку произведение $n\ (c)\ n\ (r)$ должно вычисляться для любого элемента из S_{st}. Дафф предложил более дешевую стратегию, называемую r_i в c_j, или *минимальная строка в минимальном столбце.* Однако эта стратегия не имеет симметрии. Формулируется она так. На каждом шаге выбирается столбец с наименьшим числом ненулевых элементов. Затем среди ненулевых элементов этого столбца, удовлетворяющих принятому критерию устойчивости, выбирается тот, что стоит в строке с наименьшим числом ненулевых элементов. Здесь S_{pc} содержит все ненулевые элементы активной подматрицы; S_{sp} — столбец с наименьшим числом ненулевых элементов [1]); S_{sp} — это подмножество S_{sp}, выделяемое критерием устойчивости, и $S_{piv} \equiv S_{sp}$. Наконец, главный элемент выбирается из S_{piv} исходя из соображений разреженности. Как отмечено Даффом [Duff, 1981a, p. 9], в некоторых случаях эта стратегия работает гораздо хуже, тем стратегия Марковица. Ценой некоторой дополнительной работы ее можно симметризовать. На каждом шаге выполняется предписание стратегии r_t в c_j, а затем независимо —

[1]) То есть ненулевые элементы этого столбца. — *Прим. перев.*

предписание стратегии c_k в r_i (т. е. выбирается строка с наименьшим числом ненулевых элементов, скажем строка l; выбираются ненулевые элементы строки l, подчиняющиеся критерию устойчивости; выбирается ненулевой элемент, стоящий в столбце (скажем, k) с наименьшим числом ненулевых элементов). Наконец, из двух найденных главных элементов берется тот, для которого произведение $n\ (c)\ n\ (r)$ меньше.

Существуют и другие стратегии выбора главных элементов, но ни одна, по-видимому, не дает значительно лучших результатов [1]). Разными авторами проводилось численное тестирование различных методов. Обзор этой тематики дан Даффом [Duff, 1977, p. 505]. Интересным обобщением стратегии Марковица является обсуждавшаяся в § 3.6 стратегия Златева.

Для реализации метода Марковица требуются два целых массива (назовем их *NR* и *NC*), в которые поначалу записывается число ненулевых внедиагональных элементов соответственно каждой строки и каждого столбца A. Эти массивы переопределяются от шага к шагу с учетом исключаемых элементов и заполнения; процедура аналогична той, что используется в алгоритме минимальной степени (см. § 4.8). При реализации метода возникают трудности, если активная подматрица хранится во внешней памяти, а в оперативную память может поместиться лишь часть ее. Одна из альтернатив [Duff, Reid, 1974; Reid, 1977] состоит в том, чтобы упорядочить столбцы A заранее и обрабатывать их затем в установленном порядке, не выполняя новых перестановок столбцов. Из ряда возможных стратегий выбора этого первоначального порядка упорядочение столбцов по возрастанию числа ненулевых элементов, как выяснилось, не хуже любого другого решения. На каждом шаге выполняются перестановки строк, обеспечивающие, что главным будет тот элемент ведущего столбца который удовлетворяет принятому критерию устойчивости и которому отвечает наименьшее число ненулевых элементов в одноименной строке *исходной* матрицы A. Такой способ выбора вынужден, поскольку число ненулевых элементов в строках активной подматрицы здесь не пересчитывается. Практика показывает, что он дает удовлетворительные результаты. Другие методы априорного упорядочения освещены в [Duff, 1977].

5.12. ДРУГИЕ МЕТОДЫ И ИМЕЮЩЕЕСЯ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Метод Мартина — Уилкинсона [Martin, Wilkinson, 1967) для несимметричных ленточных матриц сохраняет исходный порядок столбцов и выбирает главные элементы из ведущих столбцов

[1]) Чем описанные. —*Прим. перев.*

посредством строчных перестановок. Нижняя полулента сохраняет свою ширину, а верхняя может, в худшем случае, удвоиться. Этот метод связан с априорным упорядочением Даффа и Рида, о котором шла речь в конце § 5.11.

В § 4.14 было дано краткое описание метода конечных элементов и фронтальных схем упорядочения для симметричного положительно определенного случая. Обобщение фронтального метода на случай матриц общего вида предложено Худом (Hood, 1976]. Мы будем ссылаться на рис. 4.22 гл. 4. Подход Худа заключается в том, чтобы откладывать исключение до тех пор, пока частично собранная подматрица не достигнет максимального допускаемого порядка, а затем проводить в полностью собранной подматрице исключение в сочетании со схемой полного выбора главного элемента. Этот подход еще обобщен в (Cliffe et al., 19781, где рекомендуется отбор по барьеру в полностью собранной подматрице и выполнение шага исключения сразу по появлении приемлемого главного элемента. Так как ненулевой элемент, выбранный на роль главного, принадлежит полностью собранной подматрице (область G— G на рис. 4.22), то соответствующие строка и столбец также собраны полностью, и необходимые перестановки могут быть выполнены. Если сравнивать со случаем, когда исключение без всякого выбора главного элемента, проводится немедленно по окончании сборки строки и столбца, то в обсуждаемом методе исключения откладываются ненадолго, а фронт вырастает незначительно. Имеется подпрограмма, основанная на этих принципах, под названием MA32AD [Hopper, 1980]. В работах Duff, 1980b; Reid, 1981] предлагается в симметричном незнакоопределенном случае аналогичным образом использовать технику блочной диагонализации (см. § 3.6).

Читатель, интересующийся программами решения несимметричных или симметричных незнакоопределенных систем, должен заглянуть в Каталог программного обеспечения для разреженных матриц [Heath, 1982] или Каталог Харуэлла [Hopper, 19801 или, конкретно, в работу [Duff, 1980a I, где анонсируется версия пакета для комплексных уравнений.

Глава 6

Разреженные задачи на собственные значения

6.1. ВВЕДЕНИЕ

Стандартная задача на собственные значения определяется уравнением

$$Ax = \lambda x, \tag{6.1}$$

где A — заданная n x n-матрица. Требуется определить *собственные пары* (λ, x) матрицы A, где λ — *собственное значение,* а x — соответствующий *собственный вектор.* Обобщенная задача на собственные значения формулируется в виде

$$Ax = \lambda Bx, \tag{6.2}$$

где A и B — заданные n x n-матрицы, и снова требуется найти λ и x. Пару матриц A, B принято называть *пучком* [Gantmacher, 1959]. В случае когда $B = I,$ обобщенная задача на собственные значения сводится к стандартной задаче.

С целью упрощения изложения, а также с тем, чтобы оставаться в рамках общих тенденций, характерных для большинства литературных источников и существующего математического обеспечения, мы ограничимся (если не оговаривается противное) анализом того случая, когда матрица A —вещественная симметричная и B — вещественная симметричная положительно определен-

ная. Почти все результаты останутся справедливыми и для эрмитовых матриц, если только заменить знак транспонирования T на знак транспонирования с сопряжением H. С другой стороны, задача на собственные значения в случае, когда матрица A или матрицы A и B — эрмитовы, может быть решена с использованием математического обеспечения, предназначенного для задач с вещественными матрицами (см. §6.15).

Уравнение (6.1) имеет ненулевое решение x, если выполнено условие

$$\text{Det } (A - \lambda I) = 0, \tag{6.3}$$

которое есть не что иное, как полиномиальное уравнение n-й степени относительно λ, имеющее n корней $\lambda_1, \lambda_2, \ldots, \lambda_n$. Эти корни и являются собственными значениями матрицы A; они могут быть попарно различными, а могут быть и кратными с некоторой *кратностью*. Если матрица A — вещественная симметричная, то все собственные значения вещественны. Простейшим примером является единичная матрица I, имеющая собственное значение $\lambda = 1$ с кратностью n. Каждому собственному значению λi вещественной симметричной матрицы A соответствует ненулевой вещественный вектор x_i, являющийся решением уравнения (6.1). Эти n линейно независимых решений образуют множество собственных векторов матрицы A. Если x_i — собственный вектор, то ясно, что при любом вещественном α вектор αx_i также будет собственным, и поэтому всегда можно предполагать, что собственные векторы нормированы. Собственные векторы, отвечающие различным собственным значениям, попарно ортогональны. Собственному значению кратности m отвечают m собственных векторов, которые не определены однозначно; однако, их можно выбрать так, чтобы они были попарно ортогональными. Если X — n x n-матрица, столбцы которой суть n собственных векторов, определенных указанным образом, то нетрудно установить, что матрица X — ортогональная, т. е.

$$X^T = X^{-1}. \tag{6.4}$$

Теперь уравнение (6.1) можно переписать в виде $AX = X\Lambda$, где Λ — диагональная n x n-матрица, на диагонали которой расположены собственные значения матрицы A, упорядоченные тем же образом, что и столбцы матрицы X. Используя (6.4), получаем *спектральное разложение* матрицы A:

$$A = X\Lambda X^T \tag{6.5}$$

определяемое единственным образом [1]). Понятно, что справедливо

[1]) С оговорками, касающимися порядка собственных значений и выбора собственных векторов для кратного собственного значения. — *Прим. перев.*

также соотношение $X^T AX = \Lambda$, которое определяет преобразование *ортогонального подобия* (или преобразование *конгруэнтности),* приводящее матрицу A к диагональной матрице Λ. Существуют также и другие преобразования матрицы $A,$ при которых собственные значения не изменяются. Если G—произвольная ортогональная $n \times n$-матрица, т. е. $G^T = G^{-1},$ то стандартную задачу на собственные значения можно записать в виде

$$(GAG^T)\ (Gx) = \lambda\ (Gx). \tag{6.6}$$

Обозначив $A' = GAG^T,$ отсюда можно заключить, что если (λ, x) — собственная пара матрицы $A,$ то (λ, Gx) — собственная пара матрицы $A'.$

Аналогичные результаты справедливы для обобщенной задачи на собственные значения (6.2). Уравнение (6.2) имеет ненулевое решение $x,$ если λ удовлетворяет уравнению

$$\text{Det}\ (A - \lambda B) = 0. \tag{6.7}$$

Если матрицы A и B — вещественные и симметричные, а матрица B положительно определена, то уравнение (6.7) имеет n вещественных корней $\lambda_i,$ $i = 1, 2, \ldots, n,$ простых или кратных, и каждому из этих собственных значений отвечает вещественный собственный вектор $x_i.$ Указанные пары (λ_i, x_i) являются собственными парами пучка $A, B.$ Собственные векторы $x_i,$ $i = 1, 2, \ldots, n,$ линейно независимы и каждый из них может быть B-нормирован путем умножения на константу, такую что $x_i^T Bx_i = 1.$ Собственные векторы, отвечающие различным собственным значениям, попарно B-ортогональны: $x_i^T Bx_j = 0$ при $\lambda_i \neq \lambda_j;$ собственные векторы, отвечающие кратному собственному значению, можно выбрать так, чтобы они были B-ортогональны. Матрица $X,$ столбцами которой являются выбранные указанным образом собственные векторы, удовлетворяет соотношению

$$X^T BX = I, \tag{6.8}$$

и обобщенная задача на собственные значения может быть записана в виде $AX = BX\Lambda,$ где Λ — диагональная матрица, образованная собственными значениями.

Пусть G — произвольная обратимая $n \times n$-матрица, не обязательно ортогональная или B-ортогональная. Пусть $A' — G^T AG$ и $B' = G^T BG.$ Тогда, обобщенная задача на собственные значения (6.2) может быть записана в виде

$$(G^T AG)\ (G^{-1}x) = \lambda \qquad (G^T BG) \qquad (G^{-1}x) \tag{6.9}$$

откуда следует, что если (λ, x) — собственная пара пучка $A, B,$ то $(\lambda, G^{-1}x)$ — собственная пара $A', B'.$ Пучки A, B и A', B' называют *конгруэнтными.* Существует много различных способов

выбора матриц G, таких что A' и B' — диагональные; в частности, при $G = X$ из (6.8) следует, что обобщенная задача на собственные значения приводится к виду $X^TAX = \Lambda$.

Обобщенная задача на собственные значения может быть преобразована к стандартной форме. Простейший способ заключается в том, чтобы переписать уравнение (6.2) следующим образом:

$$B^{-1}Ax = \lambda x; \qquad (6.10)$$

здесь матрица B^{-1} существует в силу предположения о положительной определенности B. Матрица $B^{-1}A$ не является, вообще говоря, ни симметричной, ни разреженной, и приведение к виду (6.10) не рекомендуется применять для больших разреженных матриц A и B, если произведение $B^{-1}A$ должно быть сформировано явно. Формулировка (6.10) удобна, когда она используется" неявно в комбинации с методами, для реализации которых требуется лишь умение эффективно умножать матрицу на вектор; так, если u — произвольный вектор, и требуется вычислить $w = B^{-1}Au$, то формируется вектор $v = Au$, а затем решается система $Bw = v$ относительно w. В случае когда для матрицы B можно выполнить факторизацию Холецкого $B = U^T_BU_B$, уравнение (6.2) можно переписать следующим образом:

$$(U_B^{-T}AU_B^{-1})(U_Bx) = \lambda (U_Bx). \qquad (6.11)$$

Если же положительно определена матрица A и для нее можно получить факторизацию $A = U^T_AU_A$, то уравнение (6.2) принимает вид

$$(U_A^{-T}BU_A^{-1})(U_Ax) = \frac{1}{\lambda}(U_Ax). \qquad (6.12)$$

Преобразования (6.11) и (6.12) сохраняют симметрию, однако этот факт не имеет особого значения для разреженных матриц большого размера, так как обе эти формулировки обычно используются неявным образом.

Хорошо известно, что уравнения типа (6.3) или (6.7) нельзя решить алгебраически при $n > 4$. Вследствие этого задачи на собственные значения при $n > 4$ могут быть решены исключительно при помощи итерационных процедур. Перед тем как начать итерации, данную задачу можно модифицировать или привести к упрощенному виду с целью уменьшения затрат на выполнение итерационного шага, ускорения сходимости итераций и улучшения численной устойчивости. Сама по себе процедура приведения дает ощутимый вклад в общие вычислительные затраты, и этот факт должен учитываться пользователем. Вообще говоря, можно рекомендовать приведение больших разреженных матриц к специальным формам только в некоторых случаях, когда требуется вычис-

лить большое количество собственных пар. Важно проводить четкое различие между методами, которые лишь модифицируют рассматриваемую задачу на собственные значения, и методами, которые в действительности решают эту задачу путем выполнения итераций.

Среди итерационных методов, применяющихся для решения задач на собственные значения, встречаются такие, которые используют данную матрицу только для вычисления ее произведений на векторы, принадлежащие некоторой последовательности. Эти методы, являются, вообще говоря, наилучшими для задач с разреженными матрицами большого размера. Весьма маловероятно, что потребуется вычислить полную систему собственных пар для большой матрицы; в самом деле, множество собственных векторов образует заполненную $n \times n$-матрицу X. Обычные постановки задач формулируются следующим образом: (i) вычислить некоторые собственные значения, принадлежащие заданному интервалу или крайние собственные значения, или (ii) вычислить некоторые собственные пары, или (iii) вычислить все собственные значения или большое их количество, не вычисляя соответствующие собственные векторы. Свойства каждого метода определяют его область применения.

Все описываемые методы прочно укоренились в практике вычислений и документированы в обширной литературе. Ведущим научным трудом, по общему признанию, является монография «Алгебраическая проблема собственных значений» Уилкинсона [Wilkinson, 1965]. Превосходная современная книга «Симметричная проблема собственных значений» написана Парлеттом [Parlett, 1980]. С этой книгой должен ознакомиться всякий, кто имеет дело с задачами на собственные значения. Можно рекомендовать также обзор Стюарта [Stewart, 1976a], хотя он уже частично устарел. Решение задач на собственные значения для плотных матриц небольшого размера является, в сущности, закрытой проблемой, поскольку созданы очень эффективные программы, способные работать автоматически в том смысле, что они могут использоваться в режиме «черного ящика», не требуя, в частности, задания специальных параметров, определяемых пользователем. Основные алгоритмы собраны в справочнике Уилкинсона и Райнша [Wilkinson, Reinsch, 1971], а наиболее известные и тщательно оттестированные фортрановские версии этих алгоритмов были опубликованы в двух выпусках руководства для пользователей пакета программ EISPACK [Smith et al., 1976, Garbow et al., 1977].

Есть свидетельства в пользу того, что сравнимый уровень развития математического обеспечения скоро будет достигнут и в области решения задач с разреженными матрицами большого размера. Программы пакета EISPACK непригодны для решения

таких задач: исключение могут составить лишь частные случаи разреженных задач с ленточными и трехдиагональными матрицами. Разреженные задачи на собственные значения послужили поводом для выполнения обширных теоретических исследовании. (См., например, статью Руэ [Rune, 1977], обзор Дженнингса методов решения задач на собственные значения, возникающих при исследовании колебаний конструкций [Jennings, I98I], и § 5 обзора Даффа по математическому обеспечению для разреженных матриц [Duff, 1982].) Книга [Cullum, Willoughby, 1983] посвящена алгоритму Ланцоша, рассматриваемому в качестве наиболее перспективного метода решения задаче матрицами большого размера. Первые хорошие программы, однако, стали доступны лишь совсем недавно. Каталог [Heath, 1982] содержит 123 программы общего назначения для решения задач с разреженными матрицами, и лишь десять из них предназначены для решения больших задач на собственные значения. В этой главе будут рассмотрены те аспекты задач на собственные значения и методов их решения, которые имеют отношение к использованию разреженности. Ссылки на доступное программное обеспечение приводятся в конце каждого параграфа.

Прежде чем приступить к численному решению той или иной задачи на собственные значения, может оказаться удобным выполнить балансировку матрицы A с целью повышения точности вычисления собственных значений и (или) собственных векторов. Обычная процедура заключается в применении преобразования подобия вида

$$A' = D^{-1}AD,$$

где диагональная матрица D выбирается таким образом, чтобы суммы модулей элементов одноименных строк и столбцов матрицы A' были примерно равны. Матрицу D можно определить при помощи итерационной процедуры; при этом с целью предотвращения внесения погрешностей округления в результате балансировки накладывается дополнительное ограничение на матрицу D, а именно ее диагональные элементы выбираются в виде целых степеней основания системы счисления, используемой в данной ЭВМ. Эта процедура обсуждалась Парлеттом и Райншем [Parlett, Reinsch, 1969], а программа, реализующая ее, включена в пакет EISPACK.

6.2. ОТНОШЕНИЕ РЭЛЕЯ

Результаты, получаемые при помощи численных методов, являются лишь приближенными. Рассмотрим следующий вопрос: если мы располагаем приближением x к собственному вектору матрицы A, то можно ли вычислить приближение к соответствую-

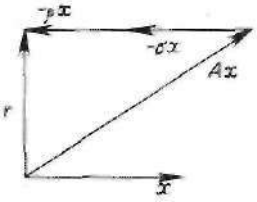


Рис. 6.1. Двумерный пример минимизированного вектора невязки.

щему собственному значению? Приведем предварительно следующий результат: если для любого ненулевого вектора χ и произвольного вещественного числа σ определить величину $s = \| Ax - \sigma x \| / \| x \|$, то существует собственное значение λ матрицы A, принадлежащее отрезку $[\sigma - s, \sigma + s]$ [1]), т. е.

$$\sigma - s \leq \lambda \leq \sigma + s. \qquad (6.13)$$

Длина отрезка равна $2s$, а его центр совпадает с точкой σ. Длина этого отрезка минимальна, если значение σ выбрано из условия ортогональности *вектора невязки* $r = Ax - \sigma x$ к вектору x, как показано на рис. 6.1, который иллюстрирует высказанное утверждение на простом двумерном примере. Указанное частное значение σ и есть *отношение Рэлея* ρ (x); при этом из условия $x^T r = 0$ получаем:

$$\rho(x) = \frac{x^T A x}{x^T x}. \qquad (6.14)$$

Отношение Рэлея представляет собой наилучшее приближение к собственному значению матрицы A, которую можно получить на основе доступной информации.

Одно из свойств отношения Рэлея заключается в том, что для любого вектора x, отличного от нулевого, оно заключено в следующих границах:

$$\lambda_{min} \leq \rho(x) \leq \lambda_{max}, \qquad (6.15)$$

где λ_{min} и λ_{max} — наименьшее и наибольшее собственное значение матрицы A соответственно. Соотношения (6.15) являются основой для построения методов отыскания крайних собственных значений посредством максимизации или минимизации отношения Рэлея ρ (x) на множестве всех нормированных векторов x. Для этой цели можно использовать метод сопряженных градиентов (см., например, работы [Geradin, 1971 1 или [Longsine, McCormick, 1981], где используется одновременная минимизация нескольких отношений Рэлея). Одним из наиболее употребительных методов минимизации отношения Рэлея является *покоординатная релаксация.* Если x_k — текущее приближение к собственному вектору, то следующее приближение определяется в этом методе в виде $x_{k+1} = x_k + \alpha e_j$, т. е. оно получается просто добавлением величины α к j-и компоненте вектора x_k. Значение α выбирается из условия минимизации величины ρ (x_{k+1}). На практике сходимость уско-

[1]) Напомним, что матрица A предполагается симметричной. — *Прим. перев*

ряют, применяя *покоординатную верхнюю релаксацию,* где $x_{k+1} = x_k + \beta\alpha e_j$, значение α выбирается из условия минимума $\rho(x_k + \alpha e_j)$, а *параметр верхней релаксации* β обычно выбирают в пределах $1 \leq \beta \leq 2$. Обзор методов релаксации можно найти в работе Руэ [Ruhe, 1977]. Шварц [Schwartz, 1977] рассматривал одновременное итерирование нескольких векторов.

Распространение указанных результатов на случай обобщенной задачи на собственные значения (6.2) осуществляется следующим образом. Для любого вектора u положим $\| u \|_{B^{-1}} = (u^T B^{-1} u)^{1/2}$. Для любого ненулевого вектора x и произвольного вещественного числа σ определим величину $s = \| Ax - \sigma Bx \|_{B^{-1}} / \| Bx \|_{B^{-1}}$; тогда найдется собственное значение λ пучка $A, B,$ принадлежащее отрезку $[\sigma - s, \sigma + s]$, т. е.

$$\sigma - s \leq \lambda \leq \sigma + s. \qquad (6.16)$$

Для любого ненулевого вектора χ отношение Рэлея определим как

$$\rho(x) = \frac{x^T Ax}{x^T Bx}. \qquad (6.17)$$

Оно обладает тем свойством, что

$$\lambda_{min} \leq \rho(x) \leq \lambda_{max}, \qquad (6.18)$$

где λ_{min} и λ_{max} — крайние собственные значения пучка A, B. Для фиксированных x и σ вектор невязки определяется в виде $r = Ax - \sigma Bx$, и при $\sigma = \rho(x)$ для него справедливо $r^T x = 0$. Методы минимизации отношения Рэлея легко распространяются на случай обобщенной задачи на собственные значения (см., например, [Schwartz, 1977] или работу [Longsine, McCormick, 1981], где разработана новая техника, использующая одновременную минимизацию отношения Рэлея по нескольким независимым векторам).

Концепция скалярного отношения Рэлея допускает обобщение, приводящее к понятию *матрицы Рэлея;* это понятие лежит в основе процедуры аппроксимации Рэлея — Ритца, широко используемой при решении задач на собственные значения для разреженных матриц. Эти вопросы обсуждаются в § 6.9.

6.3. ГРАНИЦЫ ДЛЯ СОБСТВЕННЫХ ЗНАЧЕНИЙ

Существует несколько способов получения границ для собственных значений. Определим сначала три матричные нормы (две из них уже использовались в гл. 3):

(1) 1-норма $\| A \|_1 = \max_j \sum_{i=1}^{n} | a_{ij} |$,

(2) ∞-норма $\| A \|_\infty = \max_i \sum_{j=1} | a_{ij} |$,

(3) евклидова норма $\| A \|_E = \left(\sum_{i,j=1}^{n} a_{ij}^2 \right)^{1/2}$;

здесь A — вещественная $n \times n$-матрица.

Тогда если λ — собственное значение матрицы A, то для любой из этих норм имеем

$$| \lambda | \leqslant \| A \|. \qquad (6.19)$$

Аналогично, для обобщенной проблемы собственных значений получаем оценку

$$| \lambda | \leqslant \| B^{-1} A \|, \qquad (6.20)$$

где λ — собственное значение пучка A, B, а матрица B — положительно определенная.

Теоремы Гершгорина [Gerschgorin, 1931] также удобны для получения границ, и иногда даже с их помощью удается выделить интервалы, содержащие известное количество собственных значений. Каждой строке матрицы A ставится в соответствие *круг Гершгорина* с центром a_{ii} и радиусом $\Sigma_{j \neq i} | a_{ij} |$. Эти круги, вообще говоря, определяют подмножества комплексной плоскости, хотя в случае, когда собственные значения вещественны, мы можем рассматривать их пересечение с вещественной осью и получить, таким образом, *интервалы Гершгорина.* Соответствующие теоремы устанавливают, что каждое собственное значение матрицы A содержится по крайней мере в одном из этих кругов, и что если объединение s кругов образует связную область, которая изолирована от всех остальных кругов, то указанная область содержит в точности s собственных значений матрицы A. Хотя в случае, когда размер матрицы велик, маловероятно, что объединения кругов Гершгорина будут образовывать изолированные друг от друга области (в частности при условии, что матрица A хорошо отбалансирована), эти теоремы обычно позволяют получить более точные границы, чем те, которые определяются из соотношения (6.19), и, кроме того, практическое использование теорем Гершгорина обходится весьма дешево.

Отношение Рэлея также дает способ получения границ для наименьшего и наибольшего собственных значений симметричной матрицы A. Для этого можно использовать неравенства (6.15) при том или ином выборе вектора y. Например, выбирая в качестве векторов y столбцы единичной матрицы, легко получить, что $\lambda_{min} \leq \min_i (a_{ii})$ и $\lambda_{max} \geq \max_i (a_{ii})$. Нижнюю границу для λ_{min} и верхнюю — для λ_{max} можно получить, используя интер-

валы Гершгорина; можно также взять эти границы равными двум значениям следующего выражения

$$\frac{1}{n}\left(t \pm \left((n-1)\left(n \| A \|_E^2 - t^2\right)\right)^{1/2}\right),$$

где $t = \Sigma_i a_{ii}$ — след матрицы A. Эти оценки получаются на основе итераций Лагерра для характеристического многочлена.

Полезно напомнить следующие элементарные свойства матриц. Если $\{\lambda_i\}$ — множество собственных значений матрицы A, то справедливы следующие соотношения:

$$\begin{aligned} &\sum_i \lambda_i = \sum_i a_{ii}, \text{ (след матрицы } A), \\ &\sum_i \lambda_i^2 = \| A \|_E^2, \\ &\prod_i \lambda_i = \mathrm{Det}\,(A) \text{ (определитель матрицы } A). \end{aligned} \tag{6.22}$$

В следующем параграфе будет рассмотрен метод бисекции, который также можно использовать для определения границ собственных значений. В качестве примера рассмотрим следующую матрицу, которая уже встречалась нам в § 2.13:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 2 & 4 \end{bmatrix}.$$

Пусть три собственных значения этой матрицы упорядочены по возрастанию: $\lambda_1 \leq \lambda_2 \leq \lambda_3$.

Нормы этой матрицы равны $\| A \|_1 = \| A \|_\infty = 7$, $\| A \|_E = \sqrt{41} = 6.4031\ldots$; отсюда следует, что $-6.404 < \lambda_1, \lambda_2, \lambda_3 < 6.404$. Интервалы Гершгорина равны (0, 4), (0, 6) и (1, 7); все эти интервалы взаимно перекрываются, и их объединение образует связный интервал (0, 7). Отсюда получаем, что $0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 7$. Проверяя значения диагональных элементов, мы замечаем, что $\lambda_1 \leq 2$ и $\lambda_3 \geq 4$. Отношение Рэлея для векторов (1, 1, 0), (1, 0, 1), (0, 1, 1) и (1, 1, 1) равны соответственно 3.5, 4, 5.5 и 5.666... Из итераций Лагерра получаем, что $-0.056 < \lambda_1, \lambda_2, \lambda_3 < 6.056$. Принимая во внимание все полученные выше оценки, можно заключить, что

$$0 \leq \lambda_1 \leq 2,$$

$$5.666 < \lambda_3 < 6.056.$$

Определитель равен 13, след равен 9, а собственные значения равны $\lambda_1 = 1.307979$, $\lambda_2 = 1.643104$, $\lambda_3 = 6.048917$, и все требуемые соотношения выполняются. Матрица A положительно определена, так как все ее собственные значения положительны.

Симметричные матрицы с заранее заданными «хорошими» собственными значениями и собственными векторами, полезные для примеров, могут быть легко построены [Heuvers, 1982].

6.4. МЕТОД БИСЕКЦИИ ДЛЯ ВЫЧИСЛЕНИЯ СОБСТВЕННЫХ ЗНАЧЕНИЙ

Для обобщенной задачи на собственные значения с симметричными матрицами A и B, так же как и для стандартной задачи с симметричной матрицей A, оказывается возможным вычислить количество собственных значений, меньших заданного числа σ. Соответствующая процедура основана на теореме Сильвестра о сохранении инерции (см., например, [Schwartz, 1977]). Пусть для обобщенной задачи на собственные значения (6.2) с положительно определенной матрицей B задано значение σ и пусть существует треугольная факторизация $A - \sigma B = U^T D U$, где U — верхняя треугольная матрица с единичной диагональю. Тогда количество собственных значений, меньших σ, равно количеству отрицательных элементов диагональной матрицы D. Аналогичное утверждение справедливо и для стандартной задачи при $B = I$. Заметим, что матрица $A - \sigma B$ симметрична, хотя и не является, вообще говоря, положительно определенной.

Величина

$$d_j = \prod_{i=1}^{j} d_{ii}, \qquad (6.23)$$

равна определителю j-й ведущей подматрицы в $A - \sigma B$. Последовательность $\{d_0 \equiv 1, d_1, \ldots, d_n\}$ есть *последовательность Штурма,* вычисленная для данного значения σ. Очевидно, что два соседних члена этой последовательности имеют разные знаки, если соответствующий элемент d_{ii} отрицателен, а это в свою очередь отвечает собственному значению, меньшему σ. Таким образом, количество перемен знаков в последовательности Штурма может быть использовано для подсчета числа собственных значений, меньших σ. В общем случае последовательности Штурма для разреженных матриц не используются, так как диагональная матрица D, получаемая в результате факторизации, уже содержит всю необходимую информацию. Использование этих последовательностей ограничено случаем стандартной задачи на собственные значения для трехдиагональной матрицы A, где можно применить рекуррентные соотношения, позволяющие вычислить члены последовательности Штурма, не прибегая к факторизации матрицы A [Wilkinson, 1965, p. 300].

Метод бисекции [Barth et. al., 1967] использует указанные свойства для отыскания вещественных собственных значений. Отрезок $[\alpha, \beta]$, содержащий все собственные значения, можно

определить, применяя методы, описанные в § 6.3. Затем можно взять $\sigma = (\alpha + \beta)/2$ и найти количество собственных значений, меньших σ, и количество собственных значений, больших σ. Заметим, что если σ окажется равным какому-либо собственному значению, то матрица $A - \sigma B$ будет вырожденной и этот факт будет установлен в процессе факторизации. Повторяя описанный процесс, можно локализовать любое наперед заданное собственное значение с точностью $(\beta - \alpha)/2^k$ за k шагов. С другой стороны, если заранее известно, что интересующее нас собственное значение лежит в заданном интервале, мы можем улучшить точность соответствующего приближения, используя бисекцию. Этот метод следует принимать во внимание при решении задач с разреженными матрицами A и B большого размера, если требуется найти не слишком большое число собственных значений. Одно очевидное ограничение заключается в том, что этим методом нельзя вычислить собственные векторы; однако если найдены собственные значения с некоторой точностью, не обязательно очень высокой, то для уточнения собственных значений и вычисления собственных векторов можно применить метод обратных итераций (см. ниже § 6.8). Метод бисекции реализован в пакете EISPACK [Smith et al., 1976] только для трехдиагональных матриц, так же как и его комбинация с обратными итерациями.

Для того чтобы проиллюстрировать метод бисекции, рассмотрим ту же матрицу, что и в предыдущем параграфе. При $\sigma = 3$ факторизация имеет вид

$$A - 3I = \underbrace{\begin{bmatrix} 1 & & \\ -1 & 1 & \\ -1 & 3 & 1 \end{bmatrix}}_{U^T} \underbrace{\begin{bmatrix} -1 & & \\ & 1 & \\ & & -7 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & -1 & -1 \\ & 1 & 3 \\ & & 1 \end{bmatrix}}_{U}. \tag{6.24}$$

Таким образом, матрица A имеет два собственных значения, которые меньше 3.

6.5. ПРИВЕДЕНИЕ МАТРИЦ ОБЩЕГО ВИДА

Выше уже упоминалось, что, перед тем как приступить к отысканию собственных значений каким-либо итерационным методом, решаемую задачу часто можно свести к задаче более простого вида. Приведение к упрощенному виду обычно осуществляется посредством конгруэнтных преобразований (6.6) или (6.9), применяемых к стандартной задаче на собственные значения или к обобщенной задаче соответственно. Приведенная форма матрицы является симметричной *трехдиагональной*, если исходная матрица симметрична, или *верхней хессенберговой,* если исходная матрица несимметрична.

Перейдем теперь к более детальному изучению определений, которые нам понадобятся, и вычислительных процедур.

Матрица C называется *трехдиагональной,* если ее ненулевые элементы расположены только на диагонали, на первой верхней кодиагонали (которую называют также наддиагональю) и на первой нижней кодиагонали (которую называют также поддиагональю); таким образом, $c_{ij} = 0$ при $|j - i| > 1$. Матрица C называется *верхней хессенберговой* или *верхней почти треугольной,* если ее ненулевые элементы расположены только в ее верхнем треугольнике, на диагонали и на поддиагонали, т. е. $c_{ij} = 0$ при $j < i - 1$. Симметричная хессенбергова матрица необходимо является трехдиагональной, хотя трехдиагональная матрица может быть, вообще говоря, как симметричной, так и несимметричной.

В n-мерном пространстве определим i, j-плоскость как линейную оболочку двух единичных векторов e_i и e_j: $\{x = \alpha e_i + \beta e_j \mid \alpha, \beta$ — любые вещественные числа$\}$; здесь e_i — i-й столбец единичной матрицы I. Рассмотрим матрицу $R(i, j, \theta)$, которая совпадает с единичной матрицей I, за исключением следующих четырех элементов:

$$\begin{aligned} r_{ii} &= r_{jj} = \cos\theta \equiv \gamma, \\ -r_{ij} &= r_{ji} = \sin\theta \equiv \sigma, \end{aligned} \tag{6.25}$$

где $i < j$. Матрица $R(i, j, \theta)$ является элементарной [1]) (см. § 2.3) и, кроме того, она ортогональна, т. е. $R^{-1} = R^{\mathrm{T}}$. Эта матрица соответствует преобразованию *плоского вращения* в i, j-плоскости на угол θ. Последовательность плоских вращений можно использовать для преобразования произвольной симметричной матрицы A к трехдиагональной форме. Рассмотрим преобразование ортогонального подобия

$$A \to A' = R(i, j, \theta)^{\mathrm{T}} A R(i, j, \theta). \tag{6.26}$$

Матрица A' симметрична и подобна матрице A. При этом A' совпадает с матрицей A, за исключением i-й и j-й линий *(линией* мы называем строку или столбец). Выбирая подходящим образом угол вращения θ, можно обратить в нуль любой желаемый элемент матрицы A', лежащий на этих линиях (а также и элемент, находящийся в позиции, симметричной относительно главной диагонали). Если в нуль обращаются элементы a'_{ij} и a'_{ji}, то такое преобразование называют *вращением Якоби,* в противном же случае это — *вращение Гивенса.* Приведение к трехдиагональной форме достигается посредством выполнения последовательности вращений, в результате каждого из которых появляется новая пара нулевых элементов. Предусмотрев, чтобы вращения выполнялись в соответствующем порядке, можно добиться сохранения нулевых

[1]) В действительности матрица вращения — не элементарная. — *Прим. перев.*

значений, полученных на предыдущих этапах [1]). Алгоритм, использующий вращения Якоби, можно распространить на случай обобщенной задачи на собственные значения с симметричной матрицей A и симметричной положительно определенной матрицей B; при этом элементы, расположенные в позиции (i, j) матриц A и B, аннулируются одним и тем же преобразованием конгруэнтности.

Эта же техника применима и к любой матрице A общего вида. В этом случае вращения используются для получения нулевых элементов в нижнем треугольнике, в результате чего матрица приводится к верхней хессенберговой форме. Каждое вращение требует извлечения квадратного корня при вычислении γ и σ и четырех умножений при модификации каждой пары элементов из i-й и j-й линии. Усовершенствованная процедура, получившая название *быстрых вращений Гивенса,* не требует извлечения квадратных корней и использует лишь две операции умножения на каждую пару элементов. Эта процедура была предложена Хэммерлингом [Hammerling, 1974], а ее весьма наглядное описание можно найти в книге Парлетта [Parlett, 1980, p. 100].

Для приведения матрицы общего вида к верхней хессенберговой форме или симметричной матрицы к трехдиагональной форме можно также использовать *отражения Хаусхолдера.* Как известно, гиперплоскость в n-мерном пространстве характеризуется вектором, u, ортогональным ей, и определяется как множество вида $\{x \mid u^T x = 0\}$ (отметим отличие этого определения от определения i, j-плоскости). *Матрица отражения,* или *матрица Хаусхолдера* H является элементарной ортогональной матрицей, которая «обращает» вектор u, т. е. $Hu = -u$, и оставляет без изменения любой вектор x, принадлежащий указанной гиперплоскости, т. е. $Hx = x$, если $u^T x = 0$. Тогда для любого вектора b гиперплоскость можно выбрать таким образом, что $Hb = \pm \| b \| e_1$, а именно:

$$H = I - 2\frac{uu^T}{u^T u}, \tag{6.27}$$

где

$$u = b \mp \| b \| e_1. \tag{6.28}$$

Это свойство позволяет обратить в нуль все требуемые элементы столбца матрицы при помощи одного преобразования отражения. Отражения более эффективны, чем вращения, в случае когда преобразуемая матрица A — плотная и может храниться в оперативной памяти. Заметим, однако, что если $до$ — произвольный вектор, то

$$Hw = w - 2\frac{u^T w}{u^T u}u, \tag{6.29}$$

[1]) Это относится только к вращениям Гивенса. — *Прим. перев.*

откуда видно, что в результате действия Я на матрицу A происходит вычитание вектора u, умноженного на соответствующую константу, из каждого столбца w матрицы A.

Процедуры на Алголе для симметричных матриц и их детальное обсуждение можно найти в работе [Martin et al., 1968]. Фортрановские версии этих программ включены в пакет EISPACK [Smith, et al., 1976] под именами TRED1 и TRED2. Процедуры на Алголе для матриц общего вида были опубликованы Мартином и Уилкинсоном [Martin, Wilkinson, 1968]; соответствующая фортрановская версия включена в пакет EISPACK под именем ORTHES. Область применимости этих программ ограничена в основном плотными матрицами небольшого размера, В результате как вращений, так и отражений происходит образование большого числа линейных комбинаций многих линий матрицы A. Это обстоятельство является крайне неудовлетворительным с точки зрения сохранения разреженности. Если зафиксирована определенная последовательность выбора главных элементов, то отражения никогда не приведут к меньшему заполнению, чем вращения, в то время как гауссовы преобразования подобия оказываются наилучшими в смысле величины заполнения [Duff, JReid, 1975]. Различные способы выбора главных элементов при выполнении вращений рассматривались в работах [Duff, 1974b, Rudolphi, 1973]. Эти методы, однако, должны приниматься во внимание при решении задач с очень разреженными несимметричными матрицами; кроме того, вращения Гивенса нашли важное применение в приведении ленточных матриц, о чем будет говориться в следующем параграфе. Вращения Гивенса используются также для решения разреженной линейной задачи наименьших квадратов [George, Heath, 1981]. Стратегии выбора главного элемента в случае плоских вращений рассматривались Златевым [Zlatev, 1982], и описанный им подход оказался весьма эффективным для некоторых прямоугольных матриц.

6.6. ПРИВЕДЕНИЕ СИММЕТРИЧНОЙ ЛЕНТОЧНОЙ МАТРИЦЫ К ТРЕХДИАГОНАЛЬНОЙ ФОРМЕ

Рассмотрим симметричную ленточную матрицу A порядка η с шириной ленты $2m + 1$, т. е. $a_{ij} = 0$ при $| i — j | > m$. Оказывается, что можно привести матрицу A к симметричной трехдиагональной форме при помощи вращений Гивенса (или быстрых вращений Гивенса) практически без привнесения ненулевых элементов за пределами ленты в процессе преобразований. При этом применяются вращения вида R (j— 1, j, θ), и на каждом шаге обращаются в нуль два элемента a_{kj} и a_{jk}, где $k < j$ — 1. Мы объясним работу этой процедуры, показав, как обрабатывается

первая строка матрицы *A;* при этом для наглядности будут делаться ссылки на рис. 6.2.

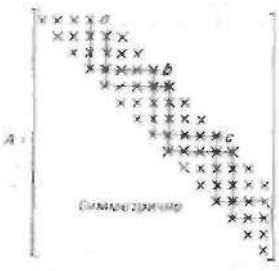


Рис. 6.2. Приведение ленточной матрицы с полушириной ленты $m = 4$ к трехдиагональной форме.

Сначала при помощи преобразования вращения R *(m.,* $m + 1$, θ_1) обращается в нуль элемент, расположенный в позиции (1, $m + 1$). На рис. 6.2 этот элемент обозначен буквой a. Преобразование R^TAR модифицирует m-е и $(m + 1)$-е линии матрицы A путем образования линейных комбинаций m-х *и* $(m + 1)$-х линий исходной матрицы, в результате чего возникает новый ненулевой элемент b в позиции *(m,* $2m + 1$).

Затем при помощи вращения R ($2m$, $2m + 1$, θ_2) обращается в нуль элемент *b;* при этом образуются линейные комбинации $2m$-х и $(2m + 1)$-х линий и возникает новый ненулевой элемент c в позиции ($2m$, $3m + 1$). Этот элемент «уничтожается» в свою очередь.вращением R ($3m$, $3m + 1$, θ_3). Для примера, показанного на рис. 6.2, на этом этапе новых ненулевых элементов не возникает, и, таким образом, исключение элемента исходной матрицы, стоявшего в позиции (1, $m + 1$), оказывается полностью завершенным.

В общем случае элементы первой строки исключаются в порядке (1, $m + 1$), (1, m), ... , (1, 3) с выполнением для каждого из них всех преобразований вращения, необходимых для того, чтобы уничтожить ненулевые элементы вне ленты. Затем по порядку обрабатываются тем же самым образом остальные строки.

Общее число вращений ограничено величиной n^2 $(m — 1)/(2m)$. Если используются вращения Гивенса, то каждое из них требует одного извлечения квадратного корня и выполнения $8m + 13$ операций, однако если применяются быстрые вращения, то число операций сокращается примерно вдвое, а извлечения квадратного корня не требуется. Соответствующая программа на Алголе была опубликована в [Schwartz, et al., 1971]. Фортрановская версия включена в пакет EISPACK [Smith et al., 1976, p. 532]. Перед тем, как приступить к выполнению вращений, можно произвести симметричную перестановку строк и столбцов матрицы A с целью уменьшения полуширины ленты m (см. выше § 4.5).

Описанный подход был распространен на случай обобщенной задачи на собственные значения (6.2) с симметричными ленточными матрицами A и B порядка n с полушириной ленты m при условии положительной определенности матрицы B [Crawford, 1973]. На первом шаге получается стандартная задача на собственные значения с матрицей, имеющей ту же ширину ленты;

при этом затрачивается $O(n^2m)$ операций. После этого на втором шаге ленточная матрица приводится к трехдиагональной форме. Эта процедура очень полезна в случаях, когда объем памяти ограничен. Ее дополнительное преимущество заключается в том, что существуют мощные методы решения задач на собственные значения с трехдиагональной матрицей. Метод рекомендуется применять, если требуется найти все собственные значения, однако он уступает другим методам при решении частичных задач на собственные значения, возникающих, например, при моделировании вибрации конструкций [Jennings, 1981]. Если матрицы A и B имеют большой размер и небольшую ширину ленты, то обычно оказываются предпочтительными другие методы, не использующие явное приведение к трехдиагональной форме.

6.7. РЕШЕНИЕ ЗАДАЧ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ ДЛЯ ТРЕХДИАГОНАЛЬНЫХ И ХЕССЕНБЕРГОВЫХ МАТРИЦ

В предыдущих параграфах обсуждались процедуры, применяемые для приведения симметричной матрицы к подобной ей симметричной трехдиагональной матрице или для приведения матрицы общего вида к подобной ей хессенберговой матрице. Трехдиагональные матрицы являются важным частным случаем разреженных матриц, в то время как хессенбергова матрица может и не быть разреженной. Рассматривался также случай ленточных матриц, и было отмечено, что симметричная обобщенная задача на собственные значения для ленточного пучка может быть сведена к стандартной задаче с симметричной трехдиагональной матрицей без использования дополнительной памяти. Кроме того, на практике часто случается, что матрица задачи на собственные значения с самого начала имеет трехдиагональный вид. В любом из этих случаев мы сталкиваемся с проблемой вычисления собственных значений приведенной матрицы.

Для решения этой задачи существуют устоявшиеся алгоритмы и легко доступные программы для ЭВМ. В известном смысле эти процедуры являются настолько мощным средством, что можно считать задачу на собственные значения решенной, если она приведена к трехдиагональной форме. В случае когда требуется найти все собственные значения или все собственные векторы и собственные значения, наиболее широко применяются QR- и QL-алгоритмы. Здесь эти алгоритмы не описываются, так как они достаточно подробно отражены в современной литературе по плотным матрицам (см., например, [Parlett, 1980]). Для определения собственных значений, принадлежащих заранее заданному интервалу, применяется также метод бисекции, основанный на использовании последовательностей Штурма; при этом для определения соответствующих собственных векторов применяется метод обратных итераций. Последние два метода обсуждаются

в § 6.4 и 6.8 ввиду того, что они представляют интерес и в случае разреженных матриц общего вида. Программы на Фортране, реализующие метод QR, метод QL, бисекцию с использованием последовательностей Штурма и метод обратных итераций для вычисления полной системы собственных пар или ее части в случае симметричной трехдиагональной или хессенберговой матрицы, включены в пакет EISPACK [Smith, 1976]. В случае хессенберговой матрицы эти программы требуют ее хранения в полном формате, и поэтому они удобны только для решения задач небольшого размера. С другой стороны, эти программы могут вполне удовлетворительно работать с симметричными трехдиагональными матрицами очень большого размера. Заметим, что собственные значения и собственные векторы хессенберговой матрицы являются, вообще говоря, комплексными.

Здесь полезно отметить, что если матрица имеет трехдиагональный вид, но несимметрична, то ее все же можно преобразованием подобия привести к симметричной трехдиагональной форме, если только произведение каждой пары взаимно симметричных кодиагональных элементов либо положительно, либо равно нулю, но при этом оба элемента — нулевые. Все собственные значения такой матрицы вещественны. Эта процедура обсуждалась Уилкинсоном [Wilkinson, 1965]; реализующая ее программа на Фортране включена в пакет EISPACK. Если несимметричная трехдиагональная матрица не имеет требуемого вида, то она должна рассматриваться как хессенбергова матрица.

6.8. ПРЯМЫЕ И ОБРАТНЫЕ ИТЕРАЦИИ

Как степенной метод, известный также под названием метода прямых итераций или итераций Стодолы, так и метод обратных итераций весьма привлекательны в качестве средства решения задач с разреженными матрицами большого размера, если требуется найти лишь несколько собственных значений *вместе* с соответствующими собственными векторами. Оба метода просто формулируются, и с их помощью можно получать достаточно точные результаты; метод обратных итераций используется также для плотных матриц небольшого размера. *Степенной метод* заключается в следующем. Выберем произвольный вектор-столбец, скажем, e_1. Умножим данную матрицу A на этот вектор и нормируем полученный вектор. Будем повторять указанную операцию до тех пор, пока получаемые векторы не обнаружат тенденцию к сходимости, скажем всего m раз. Выполнив m — 1 шагов, мы получили вектор $x = A^{m-1}e_1/\| A^{m-1}e_1\|$, где $x^Tx = 1$. Тогда на m-м шаге мы обнаруживаем, что $Ax/\| Ax \| = x' \approx x$. Поэтому можно считать, что величина $h = \| Ax \| = \frac{\| A^m e_1 \|}{\| A^{m-1} e_1 \|}$ является

приближением к собственному значению матрицы A, а вектор χ — приближением к соответствующему собственному вектору. Полученное приближение отвечает доминирующему собственному значению, однако можно легко получить и другие собственные векторы и собственные значения. Это делается следующим образом.

Предположим, что несколько собственных значений λ_1, λ_2, ... и соответствующих собственных векторов уже определены; предполагается, что эти собственные векторы ортонормированы: $x_i^T x_j = \delta_{ij}$. Выбрав любой вектор u, образуем вектор

$$w = u - (u^T x_1)\, x_1 - (u^T x_2)\, x_2 - \ldots , \qquad (6.30)$$

ортогональный каждому из собственных векторов x_1, x_2, ... , и используем этот вектор в качестве стартового при выполнении итераций. Тогда будет наблюдаться сходимость к следующему доминирующему собственному значению, не принадлежащему множеству λ_1, λ_2, ... , и к соответствующему собственному вектору. Если бы вычисления выполнялись точно, каждый новый вектор был бы в точности ортогонален всем уже известным собственным векторам. Однако в результате влияния ошибок округления в итерируемых векторах возникают «ложные» компоненты, возрастание которых за одну итерацию происходит примерно в $|\lambda_1/h|$ раз, где h — текущая оценка вычисляемого собственного значения. Поэтому необходимо время от времени выполнять переортогонализацию текущего итерируемого вектора; для определения того момента, когда возникает необходимость переортогонализации, можно использовать приведенную выше оценку роста погрешности совместно с оценкой начальной ошибки в ортогонализованном векторе.

Указанная процедура, при помощи которой устраняется повторная сходимость приближений к уже известным собственным парам, называется *исчерпыванием.* В данном случае исчерпывание осуществляется посредством проектирования итерируемого вектора на инвариантное подпространство матрицы A, дополнительное к подпространству, натянутому на уже известные собственные векторы. Описанный метод особенно удобен в разреженном случае, так как исходная матрица не подвергается никаким изменениям, и разреженность, таким образом, сохраняется. Существуют другие методы исчерпывания, однако они представляются менее удобными для использования в случае когда матрица A разрежена. (См. работы [Hotelling, 1943, Householder, 1961], или книгу Парлетта [Parlett, 1980], где дается более современное изложение.) Степенной метод можно очень легко запрограммировать; при этом могут оказаться полезными алгоритмы умножения матрицы на вектор, описанные в гл. 7.

Степенной метод может оказаться не очень удобным, когда требуется вычислить только собственные значения матрицы A. Причина заключается в том, что каждое новое собственное значение можно вычислить лишь в том случае, если доступны все собственные векторы, отвечающие ранее вычисленным собственным значениям. Эти собственные *векторы* являются плотными, и при хранении они могут потребовать даже больше памяти, чем матрица A, если требуется получить достаточно большое количество собственных значений. Однако, поскольку эти векторы используются только в процессе переортогонализации, их можно хранить на периферийных запоминающих устройствах и передавать в случае необходимости в оперативную память *по* нескольку векторов.

Осталось объяснить, почему и как сходятся прямые итерации. Рассмотрим множество ортонормированных собственных векторов $x_1, x_2, \ldots, x_n$ матрицы A как базис n-мерного пространства, и пусть произвольный вектор u имеет в этом базисе компоненты $(u_1, u_2, \ldots, u_n)$. Этот базис и компоненты разложения служат лишь теоретическим инструментом, так как к началу вычислений они неизвестны. При умножении матрицы A на вектор u полученный вектор Au имеет в этом базисе компоненты $(\lambda_1 u_1, \lambda_2 u_2, \ldots, \lambda_n u_n)$. где $\lambda_1, \ldots, \lambda_n$ — собственные значения матрицы A. Если, например, λ_1 — доминирующее собственное значение и $u_1 \neq 0$, то понятно, что $|\lambda_1 u_1| > |\lambda_i u_i| > |u_1| |u_i|$ для любого $i \neq 1$ такого, что $u_i \neq 0$. Новый вектор как бы поворачивается по направлению к вектору x_1, и если умножение на матрицу A с последующим нормированием результата повторить достаточное число раз, то возникающая последовательность векторов будет сходиться к x_1.

Теперь, если вектор x_1 уже известен, выбирается новый стартовый вектор v, ортогональный к x_1, и умножение матрицы A на этот вектор будет поворачивать его по направлению к собственному вектору, отвечающему следующему доминирующему собственному значению, скажем λ_2. Этот процесс можно продолжать до тех пор, пока не будут найдены требуемые собственные пары. Если собственное значение окажется кратным, например, $\lambda_2 = \lambda_3$, векторы v будут сходиться к одному из соответствующих собственных векторов, скажем x_2. Если затем выбрать вектор w ортогональным как к x_1 так и к x_2, то итерации будут сходиться к другому собственному вектору x_3. Конечно, любая линейная комбинация x_2 и x_3 также будет представлять собой собственный вектор, отвечающий собственному значению $\lambda_2 = \lambda_3$.

Степенной метод сходится со скоростью геометрической прогрессии, знаменатель которой равен

$$c_0 = |\lambda_q| |\lambda_s|, \tag{6.31}$$

где λ_S — доминирующее собственное значение, соответствующее инвариантному подпространству [1]), в котором ищется собственный вектор, а λ_q — следующее доминирующее собственное значение, отвечающее тому же подпространству.

Для ускорения сходимости рассматриваемых итерационных методов часто применяются *сдвиги.* Если $\{\lambda_i\}$ — множество собственных значений матрицы A, то матрица $A - \sigma I$, где σ — любое вещественное число, имеет собственные значения $\{\lambda_i - \sigma\}$ и те же самые собственные векторы. Если выбрано значение σ и степенной метод применяется к матрице $A - \sigma I$, то будет иметь место сходимость к собственному значению λ_S, наиболее удаленному от σ, которое необходимо является (алгебраически) наибольшим или наименьшим собственным значением матрицы A. В этом случае итерации сходятся со скоростью геометрической прогрессии, знаменатель которой равен

$$c_\sigma = \max_{i \neq s} |\lambda_i - \sigma| / |\lambda_s - \sigma|. \qquad (6.32)$$

Это значение нельзя значительно уменьшить за счет выбора значения σ, и поэтому при помощи сдвигов не удается существенно улучшить скорость сходимости степенного метода.

Метод обратных итераций есть не что иное, как степенной метод, примененный к матрице A^{-1}. С практической точки зрения, однако, этот метод сильно отличается От описанного выше. Если матрица A симметричная и невырожденная, то, умножая соотношение $Ax = \lambda x$ слева на A^{-1}, получаем

$$A^{-1}x = \frac{1}{\lambda} x, \qquad (6.33)$$

откуда следует, что собственные значения матрицы A^{-1} являются величинами, обратными к собственным значениям матрицы A, а собственные векторы этих матриц совпадают. Заметим, что $\lambda \neq 0$, так как по предположению матрица A невырожденна.

В методе обратных итераций, как и в степенном методе, вычисляется последовательность векторов $\{y_k\}$. При этом каждый вектор нормируется:

$$x_k = y_k / \| y_k \|, \qquad (6.34)$$

так что $x_k^T x_k = 1$. Стартовый вектор выбирается произвольным образом, например $x_0 = e_1$. Каждый следующий вектор определяется как произведение матрицы A^{-1} на нормированный предыдущий вектор:

$$y_{k+1} = A^{-1}x_k. \qquad (6.35)$$

[1]) Понятие инвариантного подпространства определяется в следующем параграфе. — *Прим. перев.*

На практике, однако, удобнее не умножать явно обратную матрицу A^{-1} на вектор, а получать каждый вектор y_{k+1} в результате решения системы линейных уравнений

$$Ay_{k+1} = x_k. \qquad (6.36)$$

Для решения систем вида (6.36) матрица A предварительно факторизуется (до начала выполнения итераций) с использованием эффективных методов, разработанных для разреженных матриц, которые обсуждались в гл. 4 и 5. Затем в процессе выполнения обратных итераций на каждом шаге уравнение (6.36) решается посредством прямой и обратной подстановки. Для решения систем (6.36) можно применять также итерационные методы (см., например, [Ruhe, Wiberg, 1972I).

Наиболее привлекательное свойство обратных итераций состоит в том, что они сходятся со скоростью геометрической прогрессии к ближайшему к нулю собственному значению λ_1 матрицы A и соответствующему собственному вектору. Если λ_2 — следующее собственное значение, ближайшее к нулю, то есть $|\lambda_1| < |\lambda_2| \leq |\lambda_i|$ при $i \neq 2$, то знаменатель геометрической прогрессии, характеризующей скорость сходимости приближений, равен

$$c_0 = |\lambda_1/\lambda_2|. \qquad (6.37)$$

Если ввести сдвиг σ, используя матрицу $A - \sigma I$ вместо A, то приближения метода будут сходиться к собственному значению матрицы A, наиболее близкому к σ. Таким образом, пользователь располагает эффективным способом выбора того собственного значения, которое он будет вычислять; этим свойством не обладает степенной метод. Если обозначить указанное собственное значение через λ_s, т. е. σ и λ_s таковы, что

$$|\lambda_s - \sigma| = \min_i |\lambda_i - \sigma|, \qquad (6.38)$$

то итерации сходятся со скоростью геометрической прогрессии, знаменатель которой равен

$$c_\sigma = |\lambda_s - \sigma| / \min_{i \neq s} |\lambda_i - \sigma|. \qquad (6.39)$$

В качестве значения σ, определяющего величину сдвига, обычно выбирается какая-либо аппроксимация собственного значения λ_s. Если используется хорошее приближение к собственному значению, то величина c_σ мала и сходимость обратных итераций будет очень быстрой. Конечно, можно применять на каждой итерации свой сдвиг, используя преимущество, возникающее за счет улучшения приближения к λ_s, достигнутого на предыдущей итерации. В § 6.2 было показано, что если известен нормированный вектор x_k, представляющий собой аппроксимацию к собственному вектору,

то наилучшей аппроксимацией к соответствующему собственному значению является отношение Рэлея $\rho_k = x_k^T A x_k$; заметим, что вычисление этой величины не требует больших затрат в случае, когда матрица A разрежена. Идея использования величины ρ_k в качестве сдвига на следующей итерации, т. е. $\sigma_{k+1} = \rho_k$, лежит в основе метода *итераций с отношением Рэлея.* Хотя здесь требуется при решении системы (6.36) выполнять факторизацию матрицы $A - \sigma_k I$ на каждой итерации, идея все же оказывается удачной, так как удается достичь асимптотически кубической скорости сходимости, т. е. если число выполненных итераций k достаточно велико, то (обычно уже при умеренных значениях k) число верных цифр в компонентах вектора x_k утраивается на каждой итерации. По мере того как значение σ_k приближается к λ_s, матрица $A - \sigma_k I$ становится плохо обусловленной; однако на самом деле это дает дополнительные выгоды, так как систему с матрицей $A - \sigma_k I$ оказывается возможным решать, не заботясь о вычислительных ошибках даже в том случае, когда они велики, поскольку возникающий вектор погрешности оказывается почти в точности пропорциональным искомому собственному вектору, и его влияние сводится лишь к улучшению скорости сходимости. Кроме того, в случае когда значение σ_k близко к λ_s, и матрица $A - \sigma_k I$ плохо обусловлена, решение y_{k+1} уравнения (6.36) оказывается очень велико по норме, и так как эта норма $\| y_{k+1} \|$ в любом случае должна вычисляться (согласно (6.36)), то ее можно использовать как удобное средство для контроля сходимости.

Метод обратных итераций со сдвигом находит важное применение в ситуациях, когда множество собственных значений уже найдено при помощи какой-либо другой процедуры, например бисекции. Одной итерации часто оказывается достаточно для получения собственного вектора с требуемой точностью [Peters, Wilkinson, 1971].

Как прямые, так и обратные итерации можно использовать для несимметричных матриц, имеющих комплексные собственные значения [1]).

Прямые и обратные итерации можно применять также и для решения обобщенной задачи на собственные значения (6.2). Степенной метод со сдвигом формулируется в этом случае следующим образом:

$$B y_{k+1} = (A - \sigma B)\, x_k,$$
$$x_{k+1} = y_{k+1} / \| y_{k+1} \|. \qquad (6.40)$$

[1]) Это утверждение в общем случае неверно, так как, например, уже для матрицы второго порядка $A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ ни прямые, ни обратные итерации в том виде, в каком они описаны здесь, не сходятся. Для таких задач следует привлекать методы, изложенные в 6.9 и 6.10. —*Прим. перев.*

На каждой итерации этого метода требуется решать систему линейных уравнений с матрицей B. Обобщение метода обратных итераций со сдвигом можно записать в следующем виде:

$$(A - \sigma B)\, y_{k+1} = Bx_k,$$
$$x_{k+1} = y_{k+1}/\| y_{k+1} \|; \qquad (6.41)$$

здесь также на каждой итерации необходимо решать систему линейных уравнений. Расширение метода итераций с отношением Рэлея на случай обобщенной задачи на собственные значения можно получить, полагая на каждой итерации сдвиг равным отношению Рэлея (6.17) для соответствующего вектора. Если последовательность $\{x_k\}$ сходится к собственному вектору, то скорость сходимости будет кубической, однако последовательность $\{x_k\}$, вообще говоря, не обязана сходиться к собственному вектору.

6.9. ПОДПРОСТРАНСТВА И ИНВАРИАНТНЫЕ ПОДПРОСТРАНСТВА

Множество всех векторов-столбцов размера η называется n-мерным пространством. Пространство векторов с вещественными компонентами обозначается $\mathscr{R}^n$, а пространство векторов с комплексными компонентами — через $\mathscr{C}^n$. Вектор представляется точкой в $\mathscr{R}^n$ или $\mathscr{C}^n$. Если, кроме того, определить на пространстве скалярное произведение, то оно становится евклидовым пространством $\mathscr{E}^n$. Скалярное произведение двух векторов $\chi = (x_1, \ldots, x_n)^T$ и $y = (y_1, \ldots, y_n)^T$ определяется следующим образом:

$$(x, y) = y^H x \equiv \sum_{i=1}^{n} y_i^* x_i, \qquad (6.42)$$

где y_i^* — величина, комплексно сопряженная к y_i а символом H обозначается операция транспонирования с сопряжением. Отметим отличие (6.42) от «вещественного» скалярного произведения

$$y^T x = \sum_{i=1}^{n} y_i x_i. \qquad (6.43)$$

В случае вещественных векторов эти выражения совпадают. Более гибкое определение скалярного произведения, подходящее для анализа обобщенной задачи на собственные значения $Ax = \lambda Bx$ с симметричной положительно определенной матрицей B, дается в виде $(x, y) = y^H Bx$, а соответствующее пространство обозначается $\mathscr{M}^n$. Можно также рассматривать скалярное произведение вида $y^H B^{-1} x$.

Потенциальные возможности алгоритмов типа Ланцоша или одновременных итераций нельзя реализовать полностью, если не использовать понятие подпространства. Множество $S = \{s_1, s_2, \ldots\}$

векторов размера n определяет *подпространство* $\mathscr{P}$ пространства $\mathscr{E}^n$, являющееся множеством всех векторов, представимых в виде линейной комбинации s_1, s_2,.... Указанному определению соответствует сокращенная запись

$$\mathscr{P} = \text{span}\ (s_1,\ s_2,\ \ldots) = \text{span}\ (S)\ ^{1}). \qquad (6.44)$$

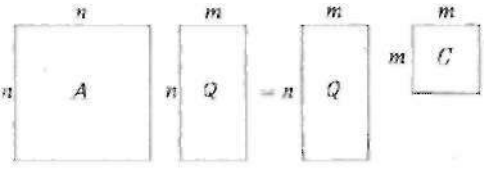
Множество S называют *образующим* множеством подпространства. Всякое подпространство имеет бесконечно много образующих множеств, содержащих различное число векторов. Образующее множество, содержащее наименьшее количество векторов, называется *базисом* подпространства, а количество m векторов базиса называется размерностью подпространства. Векторы базиса линейно независимы. Если, кроме того, эти векторы ортонормированы, то *базис* называют *ортонормированным.* Подпространство $\mathscr{P}$ (при $m > 1$) имеет бесконечно много ортонормированных базисов, однако вектор любого базиса можно выразить в виде линейной комбинации векторов всякого другого базиса. Примером подпространства может служить плоскость в трехмерном пространстве $\mathscr{R}^3$. Размерность этого подпространства равна 2, и любые два вектора, лежащие в этой плоскости и не являющиеся параллельными, образуют его базис.

Пусть теперь $A = A^{\mathrm{T}}$ — симметричная матрица (рассуждения остаются справедливыми и в комплексном случае для эрмитовой матрицы A). Подпространство $\mathscr{P}$ *инвариантно относительно* A, если для любого x из $\mathscr{P}$ вектор Ax также принадлежит $\mathscr{P}$. Собственный вектор матрицы A определяет инвариантное подпространство размерности единица, а множество m ортонормированных собственных векторов матрицы A образует базис инвариантного подпространства размерности m, натянутого на эти векторы. Важный результат, имеющий отношение к обсуждаемым вопросам, формулируется следующим образом: *любое* инвариантное подпространство является линейной оболочкой некоторого подмножества собственных векторов матрицы A.

Если $Q = (q_1, q_2, \ldots, q_m)$ — векторы некоторого базиса инвариантного подпространства $\mathscr{P}$, упорядоченные в виде $n \times m$-матрицы Q, то действием матрицы A на Q мы получаем новую $n \times m$-матрицу AQ, столбцы которой суть линейные комбинации столбцов матрицы Q. Это утверждение справедливо в силу инвариантности $\mathscr{P}$: действительно, каждый вектор Aq_i принадлежит $\mathscr{P}$. Указанные m линейных комбинаций удобно записать в виде произведения QC, где $m \times$ m-матрица C называется сужением A на $\mathscr{P}$. Таким образом, $AQ = QC$, или, более наглядно,

---

[1]) Принято говорить, что $\mathscr{P}$ — линейная оболочка, натянутая на векторы s_1, s_2, ... — *Прим. перее.*



Случай, когда столбцы матрицы Q образуют ортонормированный базис $\mathscr{P}$, представляет особый интерес. Из соотношения $Q^T Q = I_m$ (здесь I_m — единичная матрица порядка m) нетрудно получить, что $C = Q^T AQ$, где C — симметричная *матрица Рэлея.* Рассмотрим теперь задачу на собственные значения для матрицы C и пусть (λ, y) — собственная пара C, где y — m-вектор. Таким образом,

$$Cy = \lambda y, \tag{6.46}$$

Умножая слева на Q, получаем $QCy = \lambda Qy$, или

$$A\,(Qy) = \lambda\,(Qy), \tag{6.47}$$

откуда следует, что λ является также и собственным значением матрицы A, а Qy — соответствующим собственным вектором Этот результат очень важен, так как показывает, что собственные пары матрицы A большого размера можно определить, решая задачу на собственные значения для матрицы C существенно меньшего размера.

Конечно, полученный выше результат можно непосредственно использовать только в том случае, когда подпространство $\mathscr{P} =$ $=$ span (Q) инвариантно относительно A. Однако такое подпространство $\mathscr{P}$ нельзя выбрать заранее, так как это было бы равносильно решению задачи на собственные значения для матрицы A. Обычно подпространство span (Q) оказывается лишь «почти» инвариантным, и естественный вопрос заключается в том, как определить хорошие приближения к собственным парам матрицы A в этом случае. *Процедура Рэлея — Ритца,* обеспечивающая получение наилучших приближений, заключается в следующем:

(1) Выполняется ортонормализация матрицы Q (если это необходимо) и вычисляется m x m-*матрица Рэлея* $H = Q^T AQ$. Матрица Рэлея обобщает понятие скалярного отношения Рэлея, упоминавшееся выше в § 6.2. Здесь она обозначена символом H вместо C для того, чтобы подчеркнуть, что соотношение (6.45) уже не является точным равенством.

(2) Определяется требуемое количество $k \leq m$ собственных пар матрицы H, скажем, (μ_i, h_i), $i = 1, \ldots, k$. Таким образом, $Hh_i = \mu_i h_i$.

(3) Полученные *значения Ритца* μ_i суть наилучшие приближения к собственным значениям матрицы A, а *векторы Ритца*

$x_i = Qh_i$ — соответствующие приближения к собственным векторам этой матрицы.

Для того чтобы получить границы ошибок полученных приближений к собственным значениям, можно использовать результаты, приведенные в § 6.2. Если $r_i = Ax_i - \mu_i x_i$ — вектор невязки, отвечающий паре Ритца (μ_i, x_i), то отрезок $[\mu_i - \|r_i\|, \mu_i + \|r_i\|]$ содержит собственное значение матрицы A [1]). Если указанные k интервалов не пересекаются, то это гарантирует, что нами найдены приближения к k собственным значениям матрицы A. Однако, если два или более отрезков перекрывается, в них может быть заключено одно и то же собственное значение. Но даже в этом случае все же оказывается возможным идентифицировать k собственных значений матрицы A [Kahan, 1967; Parlett, 1980].

Расширение процедуры Рэлея — Ритца на случай обобщенной задачи на собственные значения производится следующим образом. Пусть столбцы n x m-матрицы Q образуют базис подпространства $\mathcal{S} = \text{span}(Q)$. Матрица Q не обязательно должна быть ортонормальной или B-ортонормальной. Тогда

(1) Вычисляются две m x m-матрицы Рэлея $H_A = Q^T AQ$ и $H_B = Q^T BQ$.
(2) Вычисляется требуемое количество k собственных пар (μ_i, h_i) задачи на собственные значения $H_A h_i = \mu_i H_B h_i$ с матрицами H_A, H_B небольшого размера.
(3) Полученные значения Ритца μ; суть наилучшие приближения к собственным значениям пучка A, B, а векторы Ритца $x_i = Qh_i$ — соответствующие приближения к собственным векторам этого пучка.

6.10. ОДНОВРЕМЕННЫЕ ИТЕРАЦИИ

Если повторно умножать матрицу A на произвольный вектор u с последующим нормированием результата, то возникающая последовательность векторов сходится к собственному вектору x_1 матрицы A. Применяя ту же процедуру к вектору v, ортогональному к x_1, а в остальном произвольному, мы получаем последовательность векторов, сходящуюся к другому собственному вектору x_2 матрицы A. Повторяя указанный процесс, можно вычислить требуемое количество собственных векторов. Такова общая схема степенного метода, описанного выше в § 6.8. *Метод одновременных итераций,* или *метод итерирования подпространства,* представляет собой естественное обобщение этой идеи. Множество из k нормированных векторов-столбцов, упорядоченных в виде n x k-матрицы U, одновременно умножаются слева на матрицу A.

[1]) Векторы h_i (а тогда и x_i) предполагаются ортонормированными. — *Прим. перев.*

Эти векторы должны быть попарно ортогональны, так как в противном случае все они будут сходиться к одному и тому же собственному вектору, т. е. матрица U должна быть ортонормальной: $U^T U = I_k$. Таким образом, столбцы матрицы U образуют ортонормированный базис подпространства $\mathscr{S} = \text{span}\ (U)$. Это подпространство не является инвариантным, хотя становится «почти» инвариантным с увеличением числа итераций алгоритма. Из предыдущего параграфа мы знаем, что наилучшие приближения к собственным векторам матрицы A, которые можно получить в подпространстве $\mathscr{S}$, — это векторы Ритца. Поэтому желательно вычислять векторы Ритца на каждом шаге. Более того, если эти векторы вычислены, то именно их будет разумно использовать вместо базиса U в качестве набора k ортонормированных векторов на следующем шаге итераций. Фактически мы хотели бы, чтобы наш пока что произвольный базис U на каждом шаге состоял в точности из множества k векторов Ритца. Эта красивая и утонченная идея приводит к построению мощного метода, который можно теперь переформулировать в следующем виде. Пусть к началу m-го шага известен ортонормированный базис U_{m-1}. Тогда на m-м шаге производится вычисление матрицы AU_{m-1} и определение нового базиса U_m подпространства span (AU_{m-1}), причем в качестве столбцов матрицы U_m выбираются векторы Ритца. Райншем [Reinsch, 1971] был предложен элегантный алгоритм, позволяющий выполнить требуемые вычисления. В соответствии с этим алгоритмом на m-й итерации

(1) Предполагается, что уже вычислена ортонормальная $n \times k$-матрица U_{m_1} так что

$$\underset{k\times n}{U_{m-1}^{\tau}}\ \underset{n\times k}{U_{m-1}} = \underset{k\times k}{I_k}. \qquad (6.48)$$

(2) Выполняется умножение A на U_{m-1}:

$$\underset{n\times k}{C} = \underset{n\times n}{A}\ \underset{n\times k}{U_{m-1}} \qquad (6.49)$$

и матрица C запоминается на месте U_{m-1}

(3) Матрица C, вообще говоря, не является ортонормальной. При помощи процедуры Грама — Шмидта выполняется ортонормализация столбцов матрицы C, в результате чего получаются матрицы Q и R, такие что

$$\underset{n\times k}{C} = \underset{n\times k}{Q}\ \underset{k\times k}{R}, \qquad (6.50)$$

где матрица Q — ортонормальная, так что

$$Q^{\mathrm{T}}Q = I_k, \qquad (6.51)$$

а матрица R — верхняя треугольная. Матрица Q запоминается на месте матрицы C.

(4) Вычисляется произведение RR^{T}, и для полученной симметричной положительно определенной матрицы небольшого размера решается задача на собственные значения. В результате получается спектральное разложение вида

$$\underset{k\times k}{R}\ \underset{k\times k}{R^{\mathrm{T}}} = \underset{k\times k}{P}\ \underset{k\times k}{D}\ \underset{k\times k}{P^{\mathrm{T}}}, \tag{6.52}$$

где D — диагональная матрица с положительными диагональными элементами, которые равны квадратам новых значений Ритца, причем

$$P^{\mathrm{T}}P = I_k. \tag{6.53}$$

(5) Вычисляются новые векторы Ритца, из которых формируется новый базис:

$$\underset{n\times k}{U_m} = \underset{n\times k}{Q}\ \underset{k\times k}{P} \tag{6.54}$$

и матрица U_m запоминается на месте Q. Затем выполняется проверка на сходимость итераций, и в случае необходимости осуществляется возврат к этапу (1).

Докажем, что столбцы матрицы U_m являются векторами Ритца, отвечающими подпространству $\mathscr{S} = \mathrm{span}\,(U_m)$. Сначала, используя последовательно соотношения (6.54), (6.51) и (6.53), покажем, что столбцы матрицы U_m ортонормальны:

$$U_m^{\mathrm{T}}U_m = (QP)^{\mathrm{T}}\,QP = P^{\mathrm{T}}\,(Q^{\mathrm{T}}Q)\,P = I_k.$$

Нам потребуется также следующий результат:

$$QP = AU_{m-1}R^{\mathrm{T}}PD^{-1}. \tag{6.55}$$

Соотношение (6.55) нетрудно получить, если заметить, что из (6.53) следует $DP^{\mathrm{T}}PD^{-1} = I_k$, так что $QP = QPDP^{\mathrm{T}}PD^{-1}$; тогда, используя последовательно соотношения (6.52), (6.50) и (6.49), нетрудно вывести требуемое равенство. Используем теперь еще одно свойство, заключающееся в том, что матрица A^{-2} имеет те же самые собственные векторы, что и матрица A, и, следовательно, аппроксимации Рэлея — Ритца из подпространства $\mathscr{S}$, отвечающие матрицам A и A^{-2}, совпадают [1]. Используя последовательно (6.54), (6.55), (6.48), (6.52) и (6.53), получаем, что сужение матрицы A^{-2} на $\mathscr{S}$ можно записать в виде

$$U_m^{\mathrm{T}}A^{-2}U_m = (QP)^{\mathrm{T}}\,A^{-2}QP = D^{-1}, \tag{6.56}$$

Таким образом, матрицей Рэлея для A^{-2} служит диагональная матрица D^{-1}. Обычная процедура для отыскания векторов Ритца

[1]) Это утверждение автора неверно; см. § 14.2 монографии [Parlett, 1980]. В общем случае можно рассчитывать не более чем на приближенное равенство этих аппроксимаций, — ***Прим. перев.***

заключается в выполнении спектрального разложения D^{-1} и последующем умножении матрицы U_m на полученную матрицу из собственных векторов. Но спектральное разложение D^{-1} есть, $D^{-1} = I_k D^{-1} I_k$, матрица из собственных векторов равна I_k, откуда $U_m I_k = U_m$, т. е, U_m представляет собой матрицу, образованную векторами Ритца из подпространства $\mathscr{S}$, отвечающими матрице A или A^{-2} [1]. Кроме того, значения Ритца для A^{-2} суть диагональные элементы матрицы D^{-1}, так что диагональные элементы матрицы $D^{1/2}$ равны значениям Ритца для A, т. е. наилучшим приближениям к собственным значениям матрицы A, которые можно получить, используя подпространство $\mathscr{S}$.

Метод одновременных итераций часто сочетается с обратными итерациями, т. е. вместо матрицы A используется $(A - \sigma I)^{-1}$, где σ — параметр сдвига. Указанным способом можно определить k собственных значений матрицы A, ближайших к σ, и соответствующие собственные векторы. Сходимость итераций может значительно ускориться по сравнению с исходным вариантом метода, особенно в том случае, когда требуемые собственные значения образуют кластер (т. е. относительно близки друг к другу). Скорость сходимости часто оказывается зависящей от размерности k итерируемого подпространства, и одним из недостатков метода является то, что значение k не известно априори, хотя оно и может быть легко изменено в процессе итераций либо путем добавления новых векторов к базису, либо при помощи исчерпывания из базиса тех собственных векторов, которые уже получены с достаточной точностью. Векторы Ритца не обязательно должны вычисляться на каждом шаге. Прежде чем вычислять очередной набор векторов Ритца, можно умножить матрицу A на векторы базиса несколько раз или даже использовать вместо матрицы A некоторую функцию от A, например, многочлен от матрицы. Для этого обычно применяются многочлены Чебышева,

---

[1]) Это утверждение автора неверно; см. § 14.2 монографии [Parlett, 1980]. В общем случае можно рассчитывать не более чем на приближенное равенство этих аппроксимаций. — *Прим. перев.*

[2]) Алгоритм (1) — (5) способен давать правильные результаты только в случае положительно определенной матрицы A. Если есть два собственных значения матрицы A, скажем, λ и μ, такие что $\lambda = -\mu$, то таким способом не удастся определить не только отрицательные собственные значения, но и собственные векторы для λ и μ, так как в этом случае соответствующие аппроксимации Рэлея—Ритца для A и A^{-2} не совпадают даже приближенно. Достаточно рассмотреть пример $n = 2$, $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $U_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Поэтому как прямые, так и обратные итерации подпространства лучше всего (с точки зрения надежности вычислений) выполнять по алгоритму 14.2.2 из [Parlett, 1980], хотя в нем требуется в два раза больше умножений матрицы на вектор. Если же принято решение использовать более экономичный на каждой итерации алгоритм (1) — (5), то следует предпринять определенные меры предосторожности, проверяя, в частности, невязки вычисляемых собственных пар. — *Прим. перев.*

которые обладают рядом свойств, позволяющих добиться улучшения сходимости; указанный подход лежит в основе методов *чебышевского ускорения* итераций. Здесь опять трудно определить все детали алгоритма заранее, и поэтому для выбора степени многочлена приходится применять эвристические правила. Рутисхаузер [Rutishauser, 1970] разработал алгоритм, включающий в себя большинство известных усовершенствований. Соответствующая программа упоминается в каталоге Хита [Heath, 1982, p. 83]; там же даются сведения о том, как получить доступ к этой программе.

Одновременные итерации можно использовать для решения обобщенной задачи на собственные значения $Ax = \lambda Bx$. В этом случае обычно применяется техника обратных итераций со сдвигом a. После введения сдвига задача на собственные значения принимает следующий вид:

$$(A - \sigma B)\, X = BX\Lambda. \qquad (6.57)$$

Тогда на m-й итерации

(1) Предполагается, что уже вычислена $n \times k$-матрица U_{m_1}. Матрица U_{m-1} не обязана быть ортонормальной или B-ортонормальной. Пусть также известна диагональная $k \times k$-матрица Λ_{m-1} образованная приближениями к собственным значениям.

(2) Вычисляется матрица $R = BU_{m-1}\Lambda_{m-1}$ и решается система уравнений

$$(A - \sigma B)\, C = R$$

относительно $n \times k$-матрицы C. Это отношение отвечает уравнению (6.57).

(3) Теперь переходим к вычислению аппроксимаций Рэлея—Ритца из подпространства span (C) в соответствии со способом, описанным в конце 6.9. Для этого находим две $k \times k$-матрицы Рэлея:

$$H_A \equiv C^{\mathrm{T}} (A - \sigma B)\, C = C^{\mathrm{T}} R,$$
$$H_B \equiv C^{\mathrm{T}} BC.$$

(4) Решаем полную обобщенную задачу на собственные значения для пучка матриц малого размера H_A, H_B:

$$(H_A - \sigma H_B)P = H_B P \Lambda_m.$$

В результате получаем новую диагональную матрицу Λ_m, образованную значениями Ритца, и H_B-ортогональную $k \times k$-матрицу P:

$$P^{\mathrm{T}} H_B P = I_k.$$

(5) Формируем новый базис

$$U_m = CP.$$

Столбцы матрицы U_m суть новые векторы Ритца. Затем выполняется проверка на сходимость итераций и в случае необходимости производится возврат к этапу (1).

Программа для решения обобщенной задачи на собственные значения методом одновременных итераций была опубликована в работе [Nikolai, 1979] и включена в каталог Хита [Heath, 1982, p. 87a].

6.11. АЛГОРИТМ ЛАНЦОША

Метод Ланцоша *[Lanczos,* 1950] получается в том случае, когда «почти» инвариантное подпространство $\mathscr{P}$, в котором строятся аппроксимации Рэлея—Ритца, выбирается в виде *подпространства Крылова.* Для произвольного ненулевого вектора b подпространство Крылова определяется следующим образом:

$$\mathscr{K}^m = \text{span}\,(b,\ Ab,\ A^2b,\ \ldots,\ A^{m-1}b). \qquad (6.58)$$

Такой выбор представляется весьма разумным, так как подпространство $\mathscr{K}^m$ действительно будет почти инвариантно относительно A при достаточно большом значении m. Чтобы показать это, рассмотрим любой вектор u из $\mathscr{K}^m$. Тогда u можно представить в виде линейной комбинации векторов $b,\ Ab,\ \ldots,\ A^{m-1}b$. В результате умножения матрицы A на вектор u получается линейная комбинация векторов $Ab,\ A^2b,\ \ldots,\ A^mb$, все слагаемые которой, за исключением последнего члена, пропорционального A^mb, принадлежат подпространству $\mathscr{K}^m$. Однако в § 6.8 было показано, что вектор $A^{m-1}b$ при достаточно больших значениях m близок к собственному вектору матрицы A [1]), так что вектор $A^mb = A\,(A^{m-1}b)$ приблизительно пропорционален вектору $A^{m-1}b$ и, таким образом, «почти» принадлежит $\mathscr{K}^m$. Таким образом, мы убедились в том, что для любого u из $\mathscr{K}^m$ вектор Au «в основном» принадлежит $\mathscr{K}^m$, а это и означает, что $\mathscr{K}^m$ почти инвариантно относительно A, причем близость $\mathscr{K}^m$ к инвариантному подпространству улучшается с увеличением значения m.

Выбор подпространства в виде (6.58) позволяет выгодно использовать следующие важные свойства:

(1) Матрица Релея $Q^{\mathrm{T}}AQ$ — трехдиагональная; ниже она обозначается через T. Поэтому, как уже указывалось в § 6.7, вычисление пар Ритца существенно упрощается.

(2) Справедливы трехчленные рекуррентные соотношения, связывающие последовательные столбцы ортонормального базиса Ланцоша Q, и эти соотношения можно использовать для вычисления векторов базиса Q.

[1]) Конечно, при условии, что доминирует только одно собственное значение. — *Прим.. перев.*

(3) Алгоритм Ланцоша обычно использует последовательность подпространств Крылова $\mathcal{K}^1$, $\mathcal{K}^2$, ..., $\mathcal{K}^m$ и вычисляет пары Ритца для матрицы A, отвечающие каждому из этих подпространств. Сходимость обычно оказывается очень быстрой; как правило, достаточно брать $m \approx 2n^{1/2}$.
(4) Матрица A используется только для умножения на векторы. Поэтому ее можно задавать подпрограммой, вычисляющей произведение Ax для заданного вектора x. Пример такой подпрограммы дается в § 7.13.

Однако метод Ланцоша не лишен изъяна:

(5) Ошибки округления разрушают ортогональность матрицы Q.

К счастью, известны способы устранения указанного недостатка, и если влияние ошибок округления преодолено, то

(6) Алгоритм оказывается весьма экономичным и достаточно «автоматическим», чтобы служить основой для разработки стандартных программ. Для задач с разреженными матрицами большого размера он считается наиболее предпочтительным.
(7) Существуют версии алгоритма, предназначенные для отыскания крайних собственных значений, некоторых внутренних собственных значений и даже всех собственных значений симметричных или эрмитовых матриц очень большого размера. Некоторые версии алгоритма позволяют также вычислять соответствующие собственные векторы. Другие версии не дают такой возможности, однако векторы, отвечающие уже известным собственным значениям, всегда можно легко определить, используя, например, метод обратных итераций.

Прежде чем перейти к изложению алгоритма, обсудим некоторые результаты. Здесь и далее ортонормированный базис подпространства $\mathcal{K}^m$ будем обозначать через Q_m вместо Q, явно указывая, таким образом, размерность подпространства $\mathcal{K}^m$ (которую мы будем предполагать равной m; таким образом, рассматривается случай, когда векторы b, ... $A^{m-1}b$ в (6.58) линейно независимы). Во избежание путаницы необходимо помнить, что Q_m — прямоугольная матрица размера $n \times m$ с ортонормальными *столбцами,* т. е. $Q_m^T Q_m = I_m$. Из этого свойства и из определения матрицы Релея $T_m = Q_m^T A Q_m$ вовсе не следует, что $AQ_m = Q_m T_m$, так как в общем случае $Q_m Q_m^T \neq I_n$. Правильное соотношение можно записать следующим образом:

$$AQ_m = Q_m T_m + R_m, \tag{6.59}$$

где R_m — *матрица невязки;* структура этой матрицы в случае $m = 4$ (см. уравнения (6.64)) наглядно показана на рис. 6.3.

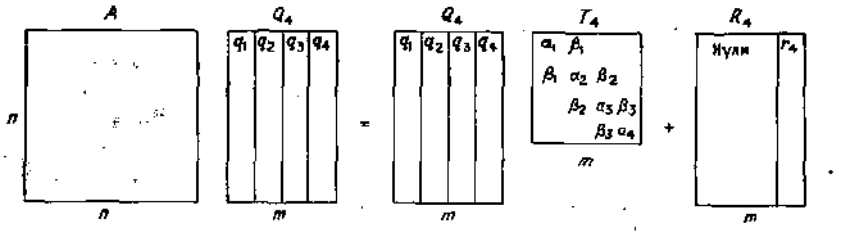


Рис. 6.3. Графическое представление алгоритма Ланцоша (6.59) в случае $m = 4$.

Конечно, если $m = n$, то матрица Q_n — ортогональная, и тогда $R_n = 0$ и $AQ_n = Q_nT_n$.

Для произвольного стартового вектора b подпространство Крылова определяется согласно (6.58), однако нам требуется построить именно ортонормированный базис $Q_m = (q_1, \ldots, q_m)$ этого подпространства. Такое построение можно осуществить непосредственно, не прибегая к предварительному вычислению последовательности Крылова. Требуемый базис строится шаг за шагом таким образом, что для *каждого* $j \leq m$ выполняется следующее соотношение:

$$\mathcal{K}^j \equiv \text{span}\,(b,\ Ab,\ \ldots.\ A^{j-1}b) = \text{span}\,(q_1, \ldots, q_j). \qquad (6.60)$$

Поскольку матрица T_m также будет строиться шаг за шагом, скоро станет ясно, что значение m. не нужно фиксировать априори. Величина m должна рассматриваться скорее как параметр, значение которого нужно увеличивать до тех пор, пока не будет достигнута сходимость. Это свойство позволяет трактовать алгоритм Ланцоша как итерационный метод, хотя, в сущности, он таковым *не является* (поскольку при $m = n$ матрица A *точно* приводится к трехдиагональной форме). Поэтому в соотношении (6.60) вместо индекса j мы будем использовать индекс m:

$$\mathcal{K}^m \equiv \text{span}\,(b,\ Ab,\ \ldots.\ A^{m-1}b) = \text{span}\,(q_1,\ q_2, \ldots q_m). \qquad (6.61)$$

Покажем теперь, как строятся последовательные ортонормированные базисы Q_m, $m = 1, 2, \ldots$, удовлетворяющие условию (6.61). Определим $q_1 = b/\|b\|$, так что (6.61) выполняется для $m = 1$. Предположим теперь, что (6.61) справедливо для некоторого m. Это означает, что вектор q_m можно выразить в виде линейной комбинации векторов $b, \ldots, A^{m-1}b$; следовательно, произведение Aq_m можно записать как линейную комбинацию векторов $Ab, \ldots, A^mb$ с теми же коэффициентами, и поэтому Aq_m принадлежит подпространству $\mathcal{K}^{m+1}$. Если, кроме того, Aq_m не является линейной комбинацией $q_1, \ldots. q_m$, то мы *определим*

q_{m+1} как векторную компоненту Aq_m, ортогональную к $q_1, \ldots, q_m$ и нормированную соответствующим образом. Тогда

$$\mathcal{K}^{m+1} = \text{span}\ (q_1, q_2, \ldots q_m, q_{m+1}), \qquad (6.62)$$

откуда следует, что условие (6.61) выполнено для каждого значения m, *если только* вектор q_{m+1} определен указанным выше образом. Заметим, что если для некоторого значения m окажется, что вектор Aq_m линейно зависит от $q_1, \ldots, q_m$, то это даже к лучшему, так как в этом случае Aq_m принадлежит $\mathcal{K}^m$, и подпространство $\mathcal{K}^m$ будет инвариантным, что дает возможность точно вычислить соответствующие m собственных пар матрицы A.

Каждый вектор q_i ортогонален ко всем остальным векторам q_j, где $j \neq i$. В частности, из того, что Aq_j принадлежит $\mathcal{K}^{j+1}$, следует

$$q_i^{\mathrm{T}} A q_j = 0, \quad j < i - 1. \qquad (6.63)$$

Это означает, что матрица Рэлея $T_m = Q_m^{\mathrm{T}} A Q_m$ имеет верхнюю хессенбергову форму; с другой стороны, матрица T_m симметрична, и поэтому T_m необходимо является трехдиагональной. Кроме того, если транспонировать соотношение (6.63), поменять в нем местами индексы i и j и учесть симметрию матрицы A, то получаем $q_i^{\mathrm{T}} A q_j = 0$ при $i < j - 1$, а это означает, что вектор Aq_j уже ортогонален всем q_i при $i<l-1$. Таким образом для того, чтобы получить вектор q_{j+1} из вектора Aq_j, достаточно ортогонализовать Aq_j по отношению к векторам q_{j-1} и q_j, а затем нормировать результат.

Перейдем теперь к подробному анализу соотношения (6.59). Его наиболее существенные детали в частном случае $m = 4$ демонстрирует рис. 6.3; при этом используются стандартные обозначения элементов матрицы T_m. Из рис. 6.3 непосредственно видно, что

$$\begin{aligned} Aq_1 &= \alpha_1 q_1 + \beta_1 q_2, \\ Aq_2 &= \beta_1 q_1 + \alpha_2 q_2 + \beta_2 q_3, \\ Aq_3 &= \beta_2 q_2 + \alpha_3 q_3 + \beta_3 q_4, \\ Aq_4 &= \beta_3 q_3 + \alpha_4 q_4 + r_4. \end{aligned} \qquad (6.64)$$

По определению матрицы $T_m = Q_m^{\mathrm{T}} A Q_m$ получаем, что $\alpha_4 = q_4^{\mathrm{T}} A q_4$ и $\beta_3 = q_3^{\mathrm{T}} A q_4$. Поэтому последнее из уравнений (6.64) можно переписать в виде $r_4 = Aq_4 - (q_3^{\mathrm{T}} A q_4)\, q_3 - (q_4^{\mathrm{T}} A q_4)\, q_4$, что вточности совпадает с выражением, которое получается в результате ортогонализации Aq_4 к q_3 и q_4 (из доказанного выше следует, в частности, что вектор Aq_4 уже ортогонален к q_1 и q_2) при вычислении вектора q_5. Таким образом, $q_5 = r_4/\beta_4$, где $\beta_4 = \| r_4 \|$, и, положив для удобства $q_0 = 0$, мы получаем соотношение

$$Aq_i = \beta_{i-1} q_{i-1} + \alpha_i q_i + \beta_i q_{i+1} \qquad (6.65)$$

справедливое при $i \leq m$; заметим, что в случае $i — m$ указанное уравнение определяет вектор невязки $r_m = \beta_m q_{m+1}$. К началу m-го шага алгоритма Ланцоша предполагается, что доступен вектор невязки r_{m-1} и его норма; тогда вычисляются векторы q_m и Aq_m. Затем, вместо вычисления $a_m = q_m^T Aq_m$ с использованием Aq_m с последующей ортогонализацией относительно q_{m_1} и q_m, оказывается более предпочтительным (с точки зрения численной устойчивости) сначала ортогонализовать Aq_m относительно вектора q_{m-1}, потом вычислить α_m, и, наконец, выполнить завершающий этап ортогонализации. Теперь можно явно сформулировать алгоритм. Выбирается стартовый вектор b с использованием либо априорной информации о собственных векторах, если она доступна, либо стандартного вектора типа $(1, 1, \ldots, 1)^T$, либо вектора со случайными компонентами. Полагаем $q_0 = 0$, $r_0 = b$ и $\beta_0 = \| b \|$. Затем для $m = 1, 2, \ldots$ повторяется следующая последовательность шагов:

Шаг 1 (Пополнение ортонормированного базиса). $q_m \leftarrow \leftarrow r_{m-1}/\beta_{m-1}$,

Шаг 2 (Вычисление «промежуточной» невязки). $r_m \leftarrow Aq_m — — \beta_{m-1} q_{m-1}$, а затем вектор q_{m-1} передается на периферийное запоминающее устройство.

Шаг 3 (Вычисление очередного диагонального элемента матрицы q_m^T). a_m - -

Шаг 4 (Завершение вычисления невязки). $r_m \leftarrow r_m — a_m q_m$.

Шаг 5. (Вычисление нормы невязки). $\beta_m \leftarrow \| r_m \|$.

Эта последовательность вычислений повторяется необходимое число раз. Пары Ритца (μ_i, x_i), аппроксимирующие собственные пары матрицы A, получаются, как и в предыдущем параграфе, посредством решения задачи на собственные значения для матрицы T_m:

$$T_m h_i = \mu_i h_i, \quad i = 1, 2, \ldots, k; \quad k \leq m, \tag{6.66}$$

с последующим вычислением [1)]

$$x_i = Q_m h_i. \tag{6.67}$$

Для контроля сходимости хорошим критерием служит Норма вектора невязки $Ax_i — \mu_i x_i$ который, в силу (6.67) и (6.66), равен $(AQ_m — Q_m T_m) h_i$ или, с учетом (6.59), $R_m h_i$. Учитывая структуру матрицы R_m, показанную на рис. 6.3, получаем

$$Ax_i - \mu_i x_i = r_m h_{im}, \tag{6.68}$$

[1)] Векторы h_i; предполагаются нормированными. — *Прим. перев.*

где h_{im} — m-я компонента вектора h_i; и, наконец, в силу того, что $\| r_m \| = \beta_m \geq 0$, приходим к равенству

$$\| Ax_i - \mu_i x_i \| = \beta_m \mid h_{im} \mid. \qquad (6.69)$$

Причиной того, что величина β_m вычисляется на шаге 5 алгоритма, а не в начале следующего цикла, является желание использовать эту величину в оценке (6.69). Видно, что если в результате решения задачи (6.66) найдены m-векторы h_i, то оценку невязки для каждой из пар Ритца можно получить косвенно, не вычисляя при этом приближений x_i к собственным векторам и избегая, таким образом, обращений к столбцам матрицы Q_m, которые хранятся на периферийном запоминающем устройстве. Отметим также, что есть способы, позволяющие вычислять компоненты h_{im} без вычисления векторов hi целиком. Теперь, в силу оценок (6.13) и равенства (6.69), можно заключить, что существует собственное значение λ матрицы A, удовлетворяющее неравенствам

$$\mu_i - \beta_m \mid h_{im} \mid \leq \lambda \leq \mu_i + \beta_m \mid h_{im} \mid. \qquad (6.70)$$

Алгоритм Ланцоша без каких-либо модификаций может применяться для решения обобщенной задачи на собственные значения, приведенной к стандартной форме (6.11) или (6.12). Так, если используется формулировка (6.11), то вместо A в произведениях с соответствующими векторами фигурирует матрица $U_B^{-T} A U_B^{-1}$. При этом достаточно хранить только матрицы A и U_B, а произведения вида $z = U_B^{-T} A U_B^{-1} u$ вычисляются следующим образом: сначала вычисляем вектор $v = U_B^{-1} u$, применяя при этом, например, технику, изложенную в параграфе 7.28, затем формируем $w = Av$, и, наконец, получаем искомый результат $z = U_B^{-T} w$, используя алгоритм, описанный в § 9.6. В остальном алгоритм Ланцоша остается без изменений.

Существует также способ использования стандартной формы вида $B^{-1} Ax = \lambda x$, не требующий явного вычисления матрицы B^{-1}. Соответствующая процедура требует лишь решения линейной системы с матрицей B на каждом шаге, что особенно удобно, если выполнение факторизации матрицы B оказывается нежелательным (например, по причине слишком большого размера матриц), и формулировка (6.11), таким образом, становится неприменимой. В случае отказа от факторизации B системы типа $Bz = u$ решаются итерационным методом. Основное рекуррентное соотношение алгоритма Ланцоша (6.65), записанное с матрицей $B^{-1}A$ вместо A, принимает следующий вид:

$$Aq_i = \beta_{i-1} Bq_{i-1} + \alpha_i Bq_i + \beta_i Bq_{i+1}. \qquad (6.71)$$

Алгоритм работает точно так же, как и раньше; единственным отличием является то, что столбцы матрицы Q_m теперь B-ортого-

нальны, т. е. $Q_m^T B Q_m = I_m$,. В соответствии с теми же уравнениями (6.66) и (6.67) определяются пары Ритца пучка *A, B,* отвечающие построенной в соответствии с (6.71) трехдиагональной матрице T_m. В случае когда требуется определить лежащие в заданном интервале собственные значения пучка *A, B* и отвечающие им собственные векторы, оказывается выгодным применять метод Ланцоша к последовательности «обращенных» и «сдвинутых» задач для пучка *B, A* — σB [Ericsson, Rune, 1980]. Соответствующая программа под названием STLM описана на с. 86 каталога Хита [Heath, 1982], и к ней можно получить доступ. В работе Скотта [Scott, 1981] содержится очень хорошее изложение алгоритма Ланцоша, а также обзор текущих исследований по этой теме.

6.12. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ АЛГОРИТМА ЛАНЦОША

На практике алгоритм Ланцоша ведет себя иначе, чем в теории. Ошибки округления нарушают ортогональность столбцов матрицы Q_m до такой степени, что они становятся линейно зависимыми. После того, как появились работы Пейджа [Paige, 1971, 1972, 1976] и Такахаши и Натори [Takahashi, Natori, 1972], стало известно, что это явление связано со сходимостью пары Ритца к собственной паре матрицы Л. Сходимость наступает в процессе выполнения итераций алгоритма, и. в то же время столбцы матрицы Q_m становятся почти линейно зависимыми. Если продолжать выполнение итераций, то может наблюдаться сходимость новых пар Ритца; однако, поскольку линейная независимость столбцов Q_m больше не имеет места, некоторые новые пары Ритца могут и, вообще говоря, будут приближать одни и те же собственные пары матрицы *A.* Может появиться значительное количество повторных копий уже полученных ранее собственных пар, и конечно же, число шагов алгоритма уже не будет ограничено величиной $m \leq n$.

Однако и из этой схемы удается извлечь полезную информацию. Предложение Пейджа заключалось в том, чтобы продолжать выполнение алгоритма до тех пор, пока не исчерпается объем памяти, а затем произвести вычисление пар Ритца и отбросить те из них, которые оказались повторными. Другой способ заключается в предотвращении появления повторных копий пар Ритца при помощи «принудительной» ортогонализации. В *алгоритме Ланцоша с переортогонализацией* это осуществляется путем выполнения непосредственной ортогонализации каждого вектора q_{m+1} ко всем q_i при $i = 1, \ldots, m$. Вычислительные затраты в этом случае велики, хотя они могут быть сокращены вдвое, если хранить матрицу Q_m в факторизованном виде [Golub. et al., 1972] с сомножителями, являющимися матрицами отражения (см. § 6.5). *Выборочная орпгогонализация* [Parlett, 1980, Parlett, Scott, 1979]

представляет собой более тонкий подход, основанный на том наблюдении, что вектор q_{m+1} отклоняется преимущественно в сторону тех векторов Ритца x_i которые достаточно хорошо аппроксимируют собственные векторы матрицы A. Для того, чтобы идентифицировать такие векторы без их непосредственного вычисления, а также отследить ситуации, когда переортогонализация действительно выгодна, используется тщательный анализ вычислительных погрешностей. Таким образом, оказывается возможным не выполнять переортогонализацию на каждом шаге, а если и выполнять, то с привлечением лишь некоторых *«пороговых»* векторов Ритца; преследуемая цель заключается в поддержании разумной степени линейной независимости и глобальной ортогональности, скажем, в соответствии с критерием $\|I - Q_m^T Q_m\| < 0.01$. Опыт показывает, что величина $\sqrt{\varepsilon}$ должна принимать значения порядка $\sqrt{\varepsilon}$, где ε — машинная точность. Основные отличительные черты указанного алгоритма заключаются в следующем:

(1) Предотвращается формирование повторных копий сошедшихся пар Ритца.
(2) Число шагов m остается минимальным и никогда не превосходит n. Процедура оканчивается в случае появления достаточно малого значения β_m.
(3) В случае, когда кратность собственного значения матрицы A, скажем λ_1, больше единицы, соответствующее множество ортонормированных собственных векторов удается определить полностью. Причина заключается в том, что выборочная ортогонализация сохраняет ортогональность каждого вектора невязки относительно уже полученных собственных векторов, отвечающих λ_1, но не предотвращает возникновения составляющих вычислительной погрешности, пропорциональных другим собственным векторам, которые отвечают собственному значению λ_1. Наличие этих составляющих со временем обеспечит получение требуемых приближений, и в конечном счете будет сформирован весь базис инвариантного подпространства, отвечающего собственному значению λ_1.
(4) Не требуется задания никаких параметров алгоритма, значения которых определяются пользователем. Алгоритм может работать в «автоматическом» режиме, не требуя в процессе своего использования обращений за консультацией к специалисту.
(5) Как только стало известно, что вектор Ритца дает достаточно хорошее приближение, он должен быть вычислен и сохранен, даже если соответствующий собственный вектор не требуется получить в качестве искомого результата. Это делает рассматриваемую процедуру подходящей для отыскания нескольких

крайних собственных пар. В процессе итераций метода Ланцоша сначала получаются приближения к собственным векторам, отвечающим крайним собственным значениям; поэтому в случае, когда требуется найти некоторые внутренние собственные пары, приходится вычислять и хранить какое-то количество собственных векторов помимо нужных.

Существует связь между алгоритмом Ланцоша и методом сопряженных градиентов для решения системы линейных уравнений $Au = v_1$ с начальным приближением $u_0 = 0$ [Cullum, Willoughby, 1977, 1978]. Такая интерпретация привела к лучшему пониманию поведения процедуры Ланцоша в условиях воздействия вычислительных ошибок при полном отказе от классического требования глобальной ортогональности векторов q_i. Если требовать, чтобы каждое собственное значение матрицы T_m аппроксимировало некоторое собственное значение матрицы A, то некоторая степень ортогональности векторов q_i должна обеспечиваться; если же ортогональность не поддерживается, то все же удается доказать, что некоторое подмножество собственных значений T_m приближает собственные значения A. Таким образом, если значение m превосходит л и достаточно велико, то оказывается возможным получить все собственные значения A, или, например, все собственные значения A, лежащие в заданном интервале. Такой метод получил название *алгоритма Ланцоша без переортогонализации,* и его отличительные особенности заключаются в следующем:

(1) Матрица T_m вычисляется и для $m > n$, невзирая на полную потерю глобальной ортогональности векторов q_i. Максимальное значение m зависит от свойств спектра матрицы A. Если собственные значения распределены достаточно равномерно, то для практических целей можно взять $m \approx 2n$. Если же собственные значения матрицы A образуют кластеры [1]), то при $m = 2n$ обычно получаются лишь приближения к каждому из кластеров, в *то* время как для того, чтобы различить отдельные собственные значения, входящие в кластер, может потребоваться примерно $m = 6n$ итераций. Известны случаи, когда требуется выполнить до $m = 10n$ итераций.

(2) Если n велико и $m > n$, то не представляется практически возможным хранить все вычисленные векторы q_i. Поэтому векторы q_i, не хранятся вообще и, таким образом, функции алгоритма ограничиваются лишь вычислением собственных значений. Такая реализация алгоритма требует весьма небольшого объема памяти. В случае необходимости собственные

[1]) В данном контексте кластером называют группу относительно близки» друг к другу собственных значений. — *Прим. перев.*

векторы можно вычислить, например, при помощи обратных итераций.

(3) Алгоритм, в сущности, становится итерационным. Возникает необходимость разработки критериев, позволяющих отбрасывать «ложные» приближения к собственным значениям и оценивать качество приемлемых приближений. Проверка, основанная на сравнении собственных значений матриц T_m и T_{m-1} [van Kats, van der Vorst, 1977] требует задания специфических параметров алгоритма, значения которых должны определяться пользователем. Этого недостатка лишен другой тест, использующий собственные значения матрицы $\check{T}_m$ порядка m—1, получаемой вычеркиванием первой строки и первого столбца матрицы T_m. Ключевым моментом является то, что если μ — простое собственное значение $\check{T}_m$, то существует собственное значение λ матрицы A такое, что верхняя оценка модуля разности $|\lambda—\mu|$ обратно пропорциональна величине $d = \mathrm{Det}(\mu I — \check{T}_m)$. В случае близости μ к некоторому собственному значению $\check{T}_m$ эта граница велика, и μ может не являться хорошим приближением к какому-либо собственному значению матрицы A. С другой стороны, если μ достаточно далеко отстоит от любого собственного значения $\check{T}_m$, то величина $1/d$ мала и должно найтись собственное значение A, очень близкое к μ; поэтому такое значение μ следует рассматривать как приближение к собственному значению матрицы A. В противном случае значение μ считается «ложным»; при этом в результате проверки каждого из собственных значений $\check{T}_m$, разрешается исключить из рассмотрения не более одного собственного значения матрицы T_m. Собственное значение T_m, имеющее кратность больше единицы, расценивается как приемлемое приближение. Теоретически матрица T_m не может иметь кратных собственных значений, однако при выполнении вычислений на ЭВМ указанная ситуация может встретиться.

(4) Недостаток описанного теста заключается в том, что он не дает возможности определить кратность собственных значений матрицы A. Действительно, учитывая лишь количество собственных значений, вычисленных на некотором шаге, не удается получить ответ на фундаментальный вопрос о том, найдены ли все собственные значения матрицы A. Эта трудность преодолевается путем сравнения числа приемлемых приближений к собственным значениям для двух различных значений m, скажем m_0 и $m_0 + \frac{n}{2}$. Если указанные два числа совпадают, то принимается решение об окончании итераций. В случае

когда требуется определить кратности, можно применить метод бисекции.

(5) Если матрица A — разреженная и для вычисления произведения Ax достаточно выполнить kn операций, где $k << n$, то все собственные значения матрицы A можно найти с затратой $O(mn)$ операций и $O(m)$ памяти. Для того чтобы вычислить все собственные значения матрицы T_m, достаточно $O(m^2)$ операций и $O(m)$ памяти. Если значение m лишь в несколько раз превосходит n, то для вычисления полного спектра матрицы A потребуется $O(n^2)$ операций и $O(n)$ памяти.

Была предложена также версия, получившая название алгоритма Ланцоша с *периодической переортогонализацией* [Grcar, 1981]. Несколько программ для решения стандартных и обобщенных задач на собственные значения, основанных на методе Ланцоша, включены в каталог Хита [Heath, 1982] и к ним можно получить доступ.

6.13. БЛОЧНЫЙ И ЛЕНТОЧНЫЙ АЛГОРИТМЫ ЛАНЦОША

Обычное подпространство Крылова строится исходя из стартового вектора b в соответствии с соотношением (6.60), где $q_1 = b/\| b \|$. Указанное определение можно обобщить, выбрав вместо одного стартового вектора множество любых k ортонормированных векторов, упорядоченных в виде $n \times k$-матрицы Q_1 и определив подпространство в виде

$$\widehat{\mathcal{K}}^{km} = \text{span} \qquad (Q_1, \qquad AQ_1, \ldots A^{m-1}Q_1). \qquad (6.72)$$

Соответствующим образом можно обобщить исходный алгоритм Ланцоша. Каждый вектор-столбец размера n заменяется на $n \times k$-матрицу, а каждый скаляр — на $k \times k$-матрицу. Вместо трехдиагональной матрицы T_m теперь появляется блочно-трехдиагональная симметричная матрица $\widehat{T}_m$ с $k \times k$-блоками на диагонали и обеих кодиагоналях. Блоки на верхней кодиагонали представляют собой нижние треугольные матрицы, а блоки на нижней кодиагонали — верхние треугольные матрицы, так что фактически матрица $\widehat{T}_m$ является ленточной матрицей с полушириной ленты $2k + 1$. Указанное обобщение было описано в работах [Cullum, Donath, 1974] и [Golub, Underwood, 1977].

Нам потребуется следующий результат. Пусть R — любая $n \times k$-матрица, отличная от нулевой, и $r = \text{rank}(R)$. Тогда существует единственное разложение вида $R = QB$, где Q — $n \times r$-матрица с ортонормальными столбцами: $Q^TQ = I_r$, а B — $r \times k$-матрица с нулевыми элементами в нижнем треугольнике: $b_{ij} = 0$

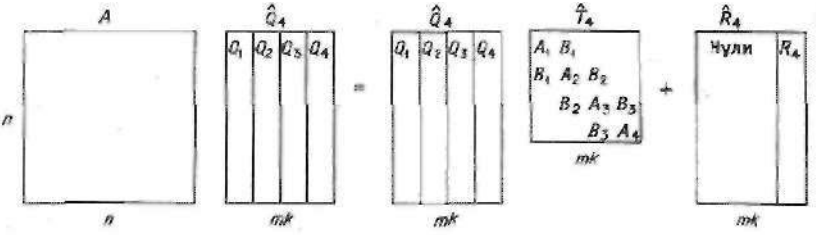


Рис. 6.4. Графическое представление блочного алгоритма Ланцоша в случае $m = 4$.

при $j < i$[1]). Заметим, что $r \leq k$. В частности, если $k < n$ и R — матрица полного столбцового ранга $r = k$, то B — квадратная верхняя треугольная $k \times k$-матрица. Этот результат есть не что иное, как матричная переформулировка процедуры ортогонализации Грама—Шмидта, примененной к столбцам матрицы R, взятым в естественном порядке.

Рис. 6.4 иллюстрирует основное соотношение $A\hat{Q}_m = \hat{Q}_m\hat{T}_m + \hat{R}_m$ в случае $m = 4$. Уравнения (6.64) заменяются на следующие:

$$\begin{aligned} AQ_1 &= Q_1A_1 + Q_2B_1, \\ AQ_2 &= Q_1B_1^T + Q_2A_2 + Q_3B_2, \\ AQ_3 &= Q_2B_2^T + Q_3A_3 + Q_4B_3, \\ AQ_4 &= Q_3B_3^T + Q_4A_4 + R_4. \end{aligned} \tag{6.73}$$

Для единообразия обозначений предположим, что обобщенное подпространство Крылова (6.72) определяется заданием множества k линейно независимых векторов размера n, упорядоченных в виде матрицы R_0. Тогда на первом шаге алгоритма вычисляется Q_i, в то время как получаемая также матрица B_0 нигде далее не используется. Для упрощения изложения будем предполагать, что матрица $A^{m-1}Q_1$ имеет полный ранг k, и, следовательно, тем же свойством обладают и матрицы B_i. Блочный алгоритм Ланцоша вполне аналогичен обычному алгоритму, описанному в § 6.11, за исключением того, что теперь первым шагом будет факторизация матрицы R_{m-1} с целью получения Q_m и B_{m-1}. Пусть $Q_0 = 0$; тогда для $m = 1, 2, \ldots$ повторяется следующая последовательность шагов:

Шаг 1 (Факторизуется матрица R_{m-1}, в результате чего получается ортонормальная матрица Q_m и верхняя треугольная B_{m-1}). $R_{m-1} = Q_mB_{m-1}$.

[1]) На самом деле единственности нет, да и нужна ли она автору? — *Прим. перев.*

Шаг 2 (Вычисляется «промежуточная» невязка). $R_m \leftarrow \leftarrow AQ_m - Q_{m-1}B_{m-1}^{\mathrm{T}}$, а затем матрица Q_{m-i}, передается на периферийное запоминающее устройство.

Шаг 3 (Вычисляется очередной диагональный блок матрицы $\widehat{T}_m$). $Q_m^{\mathrm{T}}R_m \quad A_m \quad \leftarrow$.

Шаг 4 (Завершается вычисление невязки). $R_m \leftarrow R_m - Q_m A_m$.

Требуемые пары Ритца вычисляются из соотношений $\widehat{T}_m h_i = \mu_i h_i$ и $x_i = \widehat{Q}_m h_i$, Качество полученных приближений проверяется теми же способами, что и раньше, и итерации выполняются до тех пор, пока не будет достигнута требуемая точность. Описанный алгоритм сталкивается с теми же вычислительными трудностями, что и простой алгоритм Ланцоша, и его можно комбинировать с полной переортогонализацией (когда каждая матрица Q_m *ортогонализуется* относительно всех предыдущих Q_i) или с выборочной переортогонализацией, или же вообще отказаться от переортогонализации. Несмотря на повышенную вычислительную сложность, блочный метод обладает заметными достоинствами:

(1) Кратные собственные значения матрицы A с кратностью, не превосходящей k, могут теперь определяться непосредственно, то есть не только за счет влияния компонент, возникших в результате накопления вычислительной погрешности, как это было в случае простого алгоритма Ланцоша с выборочной переортогонализацией.

(2) Оценки скорости сходимости становятся лучше, чем для простого алгоритма. Преимущества блочного алгоритма Ланцоша перед исходной версией во многих отношениях аналогичны преимуществам одновременных итераций перед простыми векторными итерациями. Этот результат, справедливый в предположении, что все вычисления выполняются точно, был получен Саадом [Saad, 1980].

Вычисление собственных значений матрицы $\widehat{T}_m$, лента которой к тому же является заполненной, требует довольно больших затрат. На практике это обстоятельство вынуждает брать значение k достаточно малым, обычно от 2 до 4, если матрица. A имеет большой размер. Дело в том, что сама матрица A может иметь лишь несколько ненулевых элементов в каждой строке, и если блоки матрицы $\widehat{T}_m$ не малы, то могут оказаться конкурентоспособными другие методы.

Другая формулировка блочного алгоритма Ланцоша была дана Руэ [Rune, 1979]. Исходная матрица A непосредственно приводится к ленточному виду при помощи рекуррентных соотно-

шений, записанных в терминах векторов размера n, а не $n \times k$-матриц. Этот алгоритм идентичен блочному алгоритму Ланцоша [1], но обладает тем преимуществом, что не требует факторизации матриц.

Программы для решения стандартных и обобщенных задач на собственные значения, реализующие блочный алгоритм Ланцоша, включены в каталог Хита [Heath, 1982].

Используя описанную процедуру, можно вычислять сингулярные значения несимметричных или прямоугольных матриц [Cullum et al., 1981]. Техника, использующая расширенную систему уравнений и предотвращающая потерю разреженности, была предложена в работе [Golub et al., 1981].

6.14. МЕТОД МИНИМИЗАЦИИ СЛЕДА

В этом методе одновременно вычисляются p наименьших или наибольших собственных значений и соответствующих собственных векторов обобщенной задачи на собственные значения, или в частном случае стандартной задачи на собственные значения. Этого удается достичь за счет минимизации следа $p \times p$-матрицы, подчиненной квадратичным ограничениям. Рассмотрим множество всех $n \times p$-матриц Y с B-ортонормированными столбцами:

$$Y^{\mathrm{T}} BY = I_p. \tag{6.74}$$

Для каждой такой матрицы Y рассмотрим величину trace $(Y^{\mathrm{T}}AY)$ [2]. Минимум этой величины достигается при $Y = X_p$, где p столбцов матрицы X_p суть собственные векторы, отвечающие наименьшим собственным значениям пучка A, B. Кроме того, указанное минимальное значение следа равно сумме этих наименьших собственных значений. Таким образом,

$$\min_{Y} (\operatorname{trace} (Y^{\mathrm{T}}AY) \mid Y^{\mathrm{T}}BY = I_p) = \operatorname{trace} (X_p^{\mathrm{T}}AX_p) = \sum_{i=1}^{p} \lambda_i. \tag{6.75}$$

Метод формулируется в виде задачи минимизации квадратичного функционала:

$$\min \operatorname{trace} (Y^{\mathrm{T}} AY) \tag{6.76}$$

с ограничениями, заданными уравнением (6.74). Эта формулировка используется для решения задачи на собственные значения следующим образом: строится последовательность итерационных

---

[1]) В смысле совпадения получаемых результатов при условна отсутствия ошибок округления. — *Прим. перев.*

[2]) В этом параграфе используется обозначение trace $(C) = \sum_{i=1}^{n} c_{ii}$ для следа матрицы C. — *Прим. перев.*

приближений $Y_{k+1} = f(Y_k)$, удовлетворяющих ограничениям (6.74), таких что

$$\text{trace}\,(Y_{k+1}^{T} A Y_{k+1}) < \text{trace}\,(Y_k A Y_k). \qquad (6.77)$$

Скорость сходимости приближений оказывается такой же, как и для одновременных обратных итераций, т. е. имеет место глобальная сходимость j-го столбца матрицы Y к собственному вектору x_j со скоростью геометрической прогрессии, знаменатель которой равен $|\lambda_j/\lambda_{p+1}|$. Представляется, что этот метод способен конкурировать с алгоритмом Ланцоша по крайней мере для матриц умеренного размера. Метод минимизации следа был предложен в работе [Wisniewski, 1981], и можно получить доступ к соответствующей программе [Heath, 1982, p. 87].

6.15. РЕШЕНИЕ ЗАДАЧ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ ДЛЯ ЭРМИТОВЫХ МАТРИЦ

Задачу на собственные значения $Ax = \lambda x$ с эрмитовой матрицей $A = A^{H}$ можно решать, оставаясь в поле вещественных чисел, если перейти к вещественной симметричной матрице A' удвоенного порядка. Пусть $A = P + iQ$, где P и Q — вещественные матрицы, причем $P^{T} = P$ и $Q^{T} = -Q$. Пусть также $x = u + iv$, где векторы u и v — вещественные. Все собственные значения матрицы A вещественны, и поэтому задача на собственные значения

$$(P + iQ)(u + iv) = X(u + iv) \qquad (6.78)$$

может быть переписана в виде

$$A'w = \lambda w, \qquad (6.79)$$

где

$$A' = \begin{bmatrix} P & -Q \\ Q & P \end{bmatrix}.$$

Каждой собственной паре (λ, x) исходной задачи отвечают два различных собственных вектора w:

$$w_1 = \begin{bmatrix} u \\ v \end{bmatrix}, \quad w_2 = \begin{bmatrix} -v \\ u \end{bmatrix}. \qquad (6.80)$$

Аналогично, обобщенной задаче на собственные значения $Ax = \lambda Bx$, где $A^{H} = A = P + iQ$, $B^{H} = B = R + iS$, ставится в соответствие задача

$$A'w = \lambda B'w, \qquad (6.81)$$

где

$$A' = \begin{bmatrix} P & -Q \\ Q & P \end{bmatrix}, \quad B' = \begin{bmatrix} R & -S \\ S & R \end{bmatrix};$$

и $x = u + iv$. Формулы (6.80) для собственных векторов сохраняют силу и в этом случае.

Таким образом, для решения задач на собственные значения с комплексными эрмитовыми матрицами можно использовать математическое обеспечение, предназначенное для решения задач с вещественными матрицами. Однако, если на данной ЭВМ комплексная арифметика реализована достаточно эффективно, предпочтительнее работать с исходными эрмитовыми матрицами [1].

6.16. ЗАДАЧИ НА СОБСТВЕННЫЕ ЗНАЧЕНИЯ ДЛЯ НЕСИММЕТРИЧНЫХ МАТРИЦ

Вещественная несимметричная матрица A порядка n имеет n собственных значений, которые, вообще говоря, могут быть комплексными и иметь любую кратность вплоть до n. Если все собственные значения различны, то каждому из них отвечает собственный вектор, и эти η собственных векторов линейно независимы. Они называются *правыми собственными векторами,* так как являются решениями уравнения $Ax = \lambda\chi$, и могут быть в общем случае комплексными. Транспонированная матрица A^T имеет те же собственные значения, что и Л, а ее собственные векторы, удовлетворяющие уравнению $A^T y = \lambda y$, или $y^T A = \lambda y^T$, являются *левыми собственными векторами* матрицы A. Для этих векторов справедливо $x_j^T y_i = 0$, если $\lambda_i \neq \lambda_3$, и $x_i^T y_i \neq 0$, т. е. эти два множества векторов образуют биортогональную систему. Однако в случае наличия кратных собственных значений система собственных векторов может быть неполной и тогда диагонализация матрицы окажется невозможной.

Несимметричные задачи на собственные значения встречаются на практике; важным примером является моделирование конструкций, находящихся под действием сил вязкости в жидкой среде, скажем, в случае анализа аэродинамического флаттера. Сталкиваясь с подобными задачами, удобно сначала проверить, используя приемы, обсуждавшиеся в § 5.9 и 5.10, не приводится ли данная матрица при помощи перестановок [2] к блочно-треугольной форме. Тогда множество собственных значений матрицы получается в результате объединения множеств собственных значений диаго-

[1]) В случае наличия комплексных версий соответствующих программ. — *Прим. перев.*

[2]) Имеются в виду симметричные перестановки строк и столбцов. — *Прим. перев.*

вальных блоков, а собственные векторы могут быть легко найдены, если известны собственные значения. Кроме того, может случиться, что некоторые блоки будут иметь порядок, равный единице, и тогда собственным значением будет являться единственный диагональный элемент этого блока. Комбинация указанной процедуры с балансированием, которое улучшает точность последующих вычислений, детально обсуждалась Парлеттом и Райншем [Parlett, Reinsch, 1969], а соответствующая подпрограмма включена в пакет EISPACK [Smith et al, 1976].

Задачи на собственные значения, отвечающие каждому диагональному блоку, можно решать с использованием последовательностей Штурма [Gupta, 1976] или одновременных итераций [Dong, 1977]. Последний метод, в частности, был опробован для больших разреженных несимметричных матриц [Clint, Jennings, 1971; Jennings, Stewart, 1975, Stewart, 1976 b].Алгоритм, позволяющий получать как правые, так и левые собственные векторы для стандартной несимметричной задачи на собственные значения, был опубликован Стюартом и Дженнингсом [Stewart, Jennings, 1981]; другой подход был предложен Саадом [Saad, 1981]. Метод Ланцоша также может быть использован для решения несимметричных задач на собственные значения [Paige, 1974; Brannstrom, 1973]. Саад также рассмотрел *биортогональный алгоритм Ланцоша* и другие методы для несимметричных матриц большого размера, опубликовав как теоретические результаты, так и данные, относящиеся к экспериментальным расчетам [Saad, 1982]. При решении задач с очень разреженными несимметричными матрицами необходимо также иметь в виду методы приведения матрицы к специальным формам. Льюис предложил алгоритм, способный определять несколько внутренних собственных значений, который основан на сведении несимметричной задачи на собственные значения к серии симметричных задач с использованием обобщения итераций с отношением Релея на случай сингулярных векторов [Lewis, 1977].

Глава 7

Алгебра разреженных матриц

7.1. ВВЕДЕНИЕ

В этой главе мы обсудим элементарные операции алгебры разреженных матриц: транспонирование, перестановки строк и столбцов, упорядочение разреженного представления, сложение и умножение двух разреженных матриц, умножение разреженной матрицы на вектор. Сложение и умножение [1)] разреженных векторов рассматривалось в гл. 1; предполагается, что читатель знаком с этим материалом.

Мы рассмотрим также треугольное разложение симметричной положительно определенной разреженной матрицы (для которой выбор главных элементов на главной диагонали численно устойчив), прямой ход и обратную подстановку для разреженного случая. Нашей целью является исследование алгоритмических аспектов исключения в ситуации, когда нижний треугольник матрицы, содержащий подлежащие исключению ненулевые элементы, не хранится машиной в явном виде, а строчный формат затрудняет эффективное программирование столбцовых просмотров.

В технологии разреженных матриц разреженная матричная алгебра играет важную роль. Имеются многочисленные примеры алгоритмов, использующих операции с разреженными матрицами: гиперматричные и сверхразреженные методы (§ 1.11), сборка конечноэлементных уравнений (гл. 8), упорядочение строчного представления перед гауссовым исключением, и многие другие. Все эти операции тривиальны для плотных и даже для ленточных матриц, но для матриц, хранимых в строчном формате, они не таковы. Здесь мы сосредоточимся на алгоритмах, использующих разреженный строчный формат (см. § 1.8, 1.9). Чтобы получить хорошие результаты, программирование алгоритмов не должно быть дилетантским. Основное соображение— конструировать алгоритмы таким образом, чтобы общая память и число арифметических операций, необходимых для выполнения определенного матричного действия, зависели линейно или были ограничены некоторой медленно растущей функцией от числа ненулевых элементов матрицы, но не от общего числа ее элементов.

В литературе алгебре разреженных матриц уделено мало внимания. Систематическое изучение этого предмета предпринято Густавсоном (исследовательский центр имени Уотсона фирмы IBM). В периоде 1972 по 1976 г. им опубликованы несколько отчетов и статей. Большая часть материала по разреженной матричной алгебре, который мы излагаем ниже, основана на его результатах, хотя с этой областью были связаны и внесли в нее полезный вклад еще несколько авторов.

Фортран является сейчас универсально признанным языком. Алгоритмы этой главы записываются сразу на Фортране; чтобы

[1)] Скалярное. — *Прим. перев.*

избежать ненужного дублирования символов, мы вводим символы Фортрана и в текст (всюду, где это возможно). Алгоритмы не оформляются как подпрограммы, чтобы их можно было использовать и в качестве подпрограмм, и как фрагменты других программ. Преобразование их в подпрограммы требует только оператора SUBROUTINE, который без труда допишет пользователь, поскольку в каждом случае ясно, где входные параметры и где результаты. Разумеется, нужны еще операторы RETURN и END. Все комментарии делаются отдельно, но при желании пользователь может набить текст на картах. Алгоритмы использовались и тестировались в течение нескольких лет. Мы подробно разъясняем их и ожидаем, что они будут эффективны и совместимы с большинством трансляторов.

Следует сделать несколько замечаний. Рассмотрим нижеследующий цикл, в котором IA — это массив строчных указателей для строчного представления разреженной матрицы:

```
      DO 10K = IA(I),IA(I+1)-1
10    CONTINUE
```

Смысл оператора DO ясен: цикл должен быть выполнен для значений k от IA (I) до IA (I + 1) — 1 с шагом 1. Если IA (I + + 1) — I < IA (I), что соответствует случаю, когда I-я строка матрицы пуста [1]), то цикл выполнять не нужно. Однако на некоторых машинах такой оператор DO будет воспринят как ошибочный, на других даже при IA (I + 1) — 1 < IA (I) тело цикла один раз будет выполнено. Мы избежим обеих этих неприятностей и получим правильно воспринимаемую программу, записав цикл следующим образом:

```
      IAA=IA(I)
      IAB=IA(I + 1)-1
      IF(IAB.LT.IAA) GO TO 20
      DO 10 K=IAA,IAB
10    CONTINUE
20
```

Некоторые трансляторы оптимизируют циклы, вынося вовне всякое вычисление, не зависящее от их счетчиков. Программист может записать внутрь цикла полные выражения. Полученная программа будет напоминать математические формулы и легко читаться, но она не будет эффективной без оптимизации циклов. Ей не достает переносимости. В отличие от этого, наши программы

[1]) Точнее, не содержит ненулевых элементов. — *Прим. перев.*

уже оптимизированы. Они несколько труднее для понимания, но должны эффективно работать на любой машине.

Для эффективной реализации матричных операций типа умножения или LU разложения можно использовать современные конвейерные машины или систолические массивы процессоров [Kung, Leiserson, 1979].

7.2. ТРАНСПОНИРОВАНИЕ РАЗРЕЖЕННОЙ МАТРИЦЫ

Имеется простой алгоритм, преобразующий строчное представление матрицы в столбцовое представление той же матрицы, или наоборот [Gustavson, 1976a, 1978]. Поскольку столбцовое представление матрицы — это строчное представление транспонированной к ней, то алгоритм в сущности транспонирует матрицу. Еще одно свойство алгоритма: результирующее представление упорядочено в том смысле, что столбцовые индексы элементов каждой строки находятся в естественном порядке возрастания. Поэтому если применить алгоритм дважды к транспонированию матрицы, первоначально заданной в неупорядоченном представлении, то будет получено упорядоченное представление той же матрицы. Если матрица симметрична, а представление — полное [1)], то для упорядочения представления достаточно одного транспонирования, хотя для симметричных матриц обычно используется верхнее [2)], а не полное представление. С небольшими изменениями тот же алгоритм можно применить для перестановки строк и/или столбцов разреженной матрицы.

Интересная и важная возможность появляется, когда некоторая матричная операция выполняется посредством алгоритма, состоящего из символического и численного этапов, и при этом желательно, чтобы результат был упорядочен. Поскольку портрет матрицы, определяемый символическим этапом алгоритма, не меняется численным этапом, то удобно упорядочить только портрет *до* проведения собственно вычислений. Последовательность процедур будет следующей:

(1) Определить портрет конечной матрицы, пользуясь символическим этапом данного алгоритма.

(2) Упорядочить портрет, применяя дважды алгоритм транспонирования.

(3) Вычислить значения элементов посредством численного этапа данного алгоритма.

Эта последовательность действий особенно разумна, когда нужно вычислить много матриц с одинаковым портретом, но различными численными значениями. Другой важный случай —

[1)] А не треугольником. — *Прим. перев.*

[2)] Т. е. верхним треугольником. — *Прим. перев.*

гауссово исключение, которое может быть выполнено численно только после того, как портрет результирующей матрицы упорядочен.

Шилд и Вышнепольский модифицировали в 1982 г. алгоритм Густавсона. Их версия в некоторых случаях работает быстрее, особенно если необязательно упорядочивать индексы конечной матрицы; ее можно использовать для выделения подматрицы в разреженной матрице. Эти авторы разработали также алгоритм для транспонирования с возможными перестановками в ситуации, когда в основной памяти находится только исходная матрица, а результирующая матрица записывается на внешнюю память.

Пусть IA, JA, AN — некоторое разреженное представление NxM матрицы. Мы хотим получить IAT, JAT, ANT — разреженное представление транспонированной матрицы. Если задаться вопросом «какие элементы матрицы, хранимой массивами IA, JA, AN, принадлежат данному столбцу», то это приведет к очень неэффективному просмотру IA и JA. Вместо того, мы введем M целых списков, поначалу пустых, и указатели первой свободной позиции каждого; вначале это будут первые позиции. Вводим также M вещественных списков. Затем мы обходим все ненулевые элементы матрицы. Для каждого ненулевого элемента I, J мы добавляем I к J-му целому списку, а численное значение этого элемента — к J-му вещественному списку, после чего увеличиваем значение соответствующего указателя. Проиллюстрируем эту процедуру примером.

Рассмотрим следующую разреженную матрицу (для простоты численные значения ненулевых элементов не указываются):

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 0 & X & 0 & X & X \\ X & 0 & 0 & X & 0 & 0 \\ 0 & 0 & X & X & 0 & 0 \\ X & 0 & X & X & 0 & 0 \\ 0 & X & 0 & 0 & X & X \end{bmatrix} \end{array}. \tag{7.1}$$

Схема RR (C)U [1] для нее такова:

```
IA = 1 4 6 8 11 14
JA = 5 6 3,4  1, 3 4,4 3 1, 2 6 5
строка =  1     2 3    4     5
```

В этом примере массив AN опущен. Так как в матрице 6 столбцов, мы заводим шесть списков. Затем мы обходим первую строку,

[1]) См. § 1.9. —*Прим. перев.*

которая содержит столбцовые индексы 5, 6 и 3; поэтому в списки 5, 6 и 3 вставляется 1:

```
1:
2:
3: 1
4:
5: 1
6: 1
```

Далее мы вставляем 2 в списки 4 и 1:

```
1: 2
2:
3: 1
4: 2
5: 1
6: 1
```

После того, как все пять строк обработаны, списки выглядят так:

```
1: 2 4
2: 5
3: 1 3 4
4: 2 3 4
5: 1 5
6: 1 5
```

или

```
JAT = 2 4, 5, 1 3 4, 2 3 4, 1    5, 1    5
столбец = 1     2     3       4       5       6
```

С точностью до массивов IAT и ANT — это схема CR (C) O заданной матрицы, или схема RR (C) O транспонированной матрицы. Массив IAT получается путем простого подсчета элементов в каждом списке; массив ANT, если требуется, можно получить, вставляя численное значение каждого ненулевого элемента, хранимое в AN, в вещественный список сразу после того, как соответствующее целое число внесено в целый список.

На практике списки организуются непосредственно в массивах JAT и ANT, а указатели первой свободной позиции каждого списка — в массиве IAT. Таким образом, дополнительной памяти не требуется, и перемещение данных в памяти машины сведено к минимуму. Полностью алгоритм будет представлен и подробно разъяснен в следующем параграфе.

7.3. АЛГОРИТМ ТРАНСПОНИРОВАНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ ОБЩЕГО ВИДА

Вход: IA, JA, AN — заданная матрица в форме RR (C) U
N — число строк матрицы
M — число столбцов матрицы
Выход IAT, JAT, KNT — транспортированная матрица в форме RR (C) O

```
1.              MH = M + 1
2.              NH=N + 1
3.              DO 10  I=2,MH
4.    10        IAT(I)=0
5.              IAB = IA(NH)-1
6.              DO 20 I = 1,IAB
7.              J = JA(I) + 2
8.              IF(J.LE.MH)IAT(J)=IAT(J)+l
9.    20        CONTINUE
10.             IAT(I)=1
11.             IAT(2)=1
12.       IF(M.EQ.I)GOTO        40
13.       DO          30 I=3,MH
14.    30         IAT(I) = IAT(I)+IAT(I-1)
15.    40         DO 60 I = l,N
16.             IAA = IA(I)
17.    IAB  =    IA(I    +    l)-1
18.             lF(IAB.LT.IAA)GOTO 60
19.             DO 50 JP = IAA,IAB
20.             J=JA(JP)+1
21.             K = IAT(J)
22.             JAT(K)=I
2J.             ANT(K)=AN(JP)
24.    50         IAT(J)=K + 1
25.    60         CONTINUE
```

θτοτ алгоритм транспонирует разреженную NxM матрицу. В каждом из массивов JAT и ANT организуется M списков. Указатели первой позиция каждого списка размещаются в массиве IAT следующим образом. В строках 6—9 программы определяется число элементов в списках: когда строка 9 выполнена в последний раз, IAT (J) содержит число элементов в (J-2)-м столбце матрицы, для значений J от 3 до M + 1. Сами указатели конструируются в строках 13 и 14: после заключительного выполнения строки 14 IAT (l) указывает позицию начала (I—l)-го списка в массивах JAT и ANT, для значений I от 2 до M + 1. Таким образом, когда в (I—1)-й список внесены все элементы и соответственно увеличено значение указателя IAT (I), то IAT (I) автоматически превращается в указатель первой позиции I-го списка, следующего в массивах JAT и ANT сразу за (I—1)-м списком. Этот результат, согласующийся с определением указателей, данным в гл. 1, получен здесь с минимумом вычислительных затрат.

В цикле DO 60 просматривается каждая строка матрицы, и ее элементы вставляются в списки, организованные в массивах JAT и ANT. В строке 20 ненулевой элемент находится в строке I и

столбце J—1. В строке 21 K указывает позицию массивов JAT и ANT, отвечающую первой свободной позиции (J—l)-го списка. В строках 22 и 23 строчный индекс I и значение AN (JP) ненулевого элемента вставляются соответственно в списки JAT и ANT; в строке 24 увеличивается значение указателя IAT (J).

7.4. УПОРЯДОЧЕНИЕ РАЗРЕЖЕННОГО ПРЕДСТАВЛЕНИЯ

Если алгоритм из § 7.3 применяется дважды, то будет получено упорядоченное представление заданной матрицы [Gustavson, 1976a, 1978]. Упорядоченные представления нужны в некоторых приложениях, например, при гауссовом исключении или для печати матрицы на выходе. Заметим, что упорядочение списка из n чисел обычно требует $n(n-1)/2$ сравнений [1]), поскольку каждое число нужно сравнить со всеми, следующими за ним в списке. Здесь же упорядочение достигается с затратой только aN + bM + + cZ элементарных операций, где a, b, c — малые числа, Z — общее число ненулевых элементов. Таким образом, асимптотическая сложность разреженного алгоритма — линейная.

В некоторых случаях требуется транспонировать или упорядочить только портрет матрицы IA, JA.Для этой цели можно использовать алгоритм из § 7.3, опустив в нем оператор строки 23, после чего он превращается в алгоритм символического транспонирования разреженной матрицы общего вида. Два важных примера [2]): связность конечно-элементной сетки и гауссово исключение (или треугольное разложение). Эти примеры подробно обсуждаются соответственно в гл. 8 и данной главе. Здесь мы упомянем только, что матрица инцидентности — это булева матрица, т. е. матрица с элементами 0 или 1. В разреженном формате значения ненулевых элементов [3]) не хранятся и, как следствие, матрица задается только своим портретом. Для гауссова исключения представление должно быть упорядоченным. Однако, так как исключение вначале выполняется символически, а затем численно и так как для символического этапа упорядоченность представления не нужна, то обычной процедурой будет та, что описана в § 7.2: проделать символическое гауссово исключение, дважды транспонировать полученный неупорядоченный портрет, посредством уже упорядоченного портрета провести численный этап гауссова исключения. Транспонирование, требуемое этой процедурой, является только символическим.

Алгоритм упорядочения (под названием *одновременной сортировки* [4]) был независимо найден в работе [Alvarado, 1979]. Другой

---

[1]) Если не использовать быстрые алгоритмы сортировки. — *Прим. перев.*
[2]) Приложений этого алгоритма. — *Прим. перев.*
[3]) Матрицы инцидентности. — *Прим. перев.*
[4]) В оригинале — simultaneous radix sort. — *Прим. перев.*

алгоритм, в некоторых случаях более быстрый, разработали Шилд и Вышнепольский [Szyld, Vishnepolsky, 1982].

7.5. ПЕРЕСТАНОВКА СТРОК ИЛИ СТОЛБЦОВ РАЗРЕЖЕННОЙ МАТРИЦЫ: ПЕРВАЯ ПРОЦЕДУРА

Изложенные выше идеи можно использовать для перестановки строк и/или столбцов матрицы, заданной в разреженном формате [Gustavson, 1976a]. Пусть

$$K = (k_1, k_2, \ldots, k_N) \qquad (7.2)$$

— некоторая перестановка чисел (1, 2, ..., N), описывающая требуемую перестановку N строк NxM матрицы A. Можно определить соответствующую NxN матрицу перестановки P:

$$p_{i,k_i} = 1, \quad p_{ij} = 0, \quad \text{во всех остальных случаях.}$$

Каждая строка и каждый столбец P содержат только по одной единице, и P — ортогональная ($P^{\mathrm{T}} = P^{-1}$). Если A умножить слева на P, то строка A с номером k_i станет строкой i полученной матрицы PA. Левое умножение A на P можно осуществить посредством алгоритма из § 7.3, лишь немного изменив его: на этапе I нужно рассматривать вместо строки I матрицы A строку k_{I}. Замены требуют только строки 16, 17 и 22. Если для хранения перестановки используется массив KK с N позициями, то новые строки имеют вид:

```
16.     IAA = IA(KK(I))
17.     IAB=IA(KK(I)+1)-1
22.     JAT(K) = KK(I)
```

На выходе алгоритма будет получена транспонированная для переупорядоченной матрицы A, а именно $(PA)^T$. Алгоритм из § 7.3 можно применить дважды, чтобы переставить в A не только строки, но и столбцы. Задача при этом ставится следующим образом: заданы NxN матрица перестановки P для строк и MxM матрица перестановки Q для столбцов, нужно вычислить матрицу PAQ^{T}. Имеем:

$$PAQ^{\mathrm{T}} = (Q\,(PA)^{\mathrm{T}})^{\mathrm{T}}. \qquad (7.4)$$

Первое применение алгоритма к A с P как матрицей перестановки дает $(PA)^{\mathrm{T}}$. Второе применение алгоритма к $(PA)^{\mathrm{T}}$ с Q как матрицей перестановки приводит к желаемому результату. Эта процедура удобна, если необходимо переставить в A и строки, и столбцы. Если же нужно переставить только строки или только столбцы, то более выгодной может оказаться нижеследующая процедура.

7.6. ПЕРЕСТАНОВКА СТРОК ИЛИ СТОЛБЦОВ РАЗРЕЖЕННОЙ МАТРИЦЫ: ВТОРАЯ ПРОЦЕДУРА

Рассмотрим вначале перестановку столбцов *A*. Пусть перестановка определяется массивом J, указывающим, что столбец *A* с номером J станет столбцом J (I) новой матрицы. Заметим, что это определение отличается от определения массива KK в предыдущем параграфе. Теперь мы просто проходим по массиву столбцовых индексов JA и заменяем каждый элемент JA (I) на J (JA(I)). Численные значения ненулевых элементов, хранимые в AN, остаются прежними. Читатель легко сможет записать соответствующий алгоритм.

Теперь рассмотрим перестановку строк матрицы *A,* и пусть J — массив, определяющий перестановку, как сказано абзацем выше. Столбцовые индексы и численные значения элементов I-й строки *A* хранятся в массивах JA и AN в позициях от IA (I) до IA (I + 1)-1. Эти числа нужно переместить на новые позиции, соответствующие строке J (I), в новых массивах JB и BN. Для этого следует построить новый массив указателей IB, используя информацию из IA и J. Затем заполняются новые массивы столбцовых индексов и численных значений JB и BN путем простого перемещения чисел из позиций IA (I), ..., IA (I + 1) — 1 массивов JA, AN в позиции JB (J (I)),..., JB (J (I) + 1)—1 массивов JB, BN.

7.7. УПОРЯДОЧЕНИЕ ВЕРХНЕГО ПРЕДСТАВЛЕНИЯ РАЗРЕЖЕННОЙ СИММЕТРИЧНОЙ МАТРИЦЫ

Имеется еще один возникающий на практике случай: упорядочение верхнего представления разреженной симметричной матрицы *A*. Эту задачу можно свести к случаю матрицы общего вида, рассматривая заданное представление как полное представление другой матрицы *B,* верхний треугольник которой совпадает с верхним треугольником *A,* а нижний треугольник и главная диагональ заполнены нулями. Тогда *B* упорядочивается двумя применениями алгоритма из § 7.3, и результатом будет упорядоченное верхнее представление *A*.

Заметим, что представление B^T, полученное после первого применения алгоритма транспонирования, есть представление A^T типа RR (L) O (или представление *A* типа CR (U) O), в то время как мы ищем представление *A* типа RR (U) O. Другими словами, для упорядочения симметричной матрицы, заданной в верхнем представлении, нужны два транспонирования, но каждое из них затрагивает только половину матрицы. При полном представлении достаточно одного транспонирования целой матрицы.

7.8. СЛОЖЕНИЕ РАЗРЕЖЕННЫХ МАТРИЦ

Сложение матриц — это одна из операций матричной алгебры. Поскольку вычитание матрицы эквивалентно сложению с матрицей, у которой обращены знаки ненулевых элементов, то мы будем называть сложением как собственно сложение, так и вычитание. Сложение разреженных матриц выполняется двумя алгоритмами: символическим алгоритмом, определяющим портрет окончательной матрицы, и численным алгоритмом, находящим значения ненулевых элементов и использующим с выгодой априорное знание их позиций. Можно организовать сложение численно только в один этап, но при таком подходе мало что можно выиграть. Сдругой стороны, разбиение процедуры сложения на два этапа вносит в программу дополнительную степень свободы, что может быть очень удобно, например, при итерационных вычислениях, где портрет фиксирован, а численные значения меняются, или когда требуется упорядоченное представление, которое может быть достигнуто упорядочением только портрета до получения численных значений. Упомянем здесь, что результатом символического алгоритма будет неупорядоченный портрет, даже если портреты всех складываемых матриц упорядочены. В то же время численный алгоритм требует задания портрета на своем входе и не меняет его.

Символический алгоритм сложения использует метод переменного переключателя, рассмотренный в § 1.14. Сложение выполняется строка за строкой. Поскольку всякая строка однозначно характеризуется своим индексом и строчные индексы берутся в порядке возрастания, то каждый может быть взят за значение переключателя. Численный алгоритм также выполняется строка за строкой, и для накапливания ненулевых элементов используется расширенный массив, как это объяснено в § 1.15.

В этой главе представлены процедуры сложения двух или большего числа разреженных матриц общего вида. Те же процедуры можно применить для сложения двух или большего числа симметричных матриц, заданных верхним представлением. Необходимость складывать симметричные матрицы с матрицами общего вида возникает редко и не будет здесь обсуждаться. Наши процедуры легко распространить на случай, когда складываемые матрицы имеют различное число строк и столбцов. Алгоритмические аспекты сложения разреженных матриц обсуждаются в следующем параграфе с помощью примера. Основные идеи почерпнуты из работ [Gustavson, 1972, 1976c; Eisenstat et al., 1975].

7.9. ПРИМЕР СЛОЖЕНИЯ ДВУХ РАЗРЕЖЕННЫХ МАТРИЦ

Рассмотрим сложение следующих двух матриц A и B, результатом которого будет матрица C:

$$\begin{bmatrix} 0 & 0 & 2 & 0 & -1 & 0 \\ 4 & 0 & 3 & 3 & 7 & 0 \\ -2 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 4 & 6 & 0 & 2 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 & 0 & -1 & 5 \\ 4 & 0 & 3 & 3 & 5 & 0 \\ 2 & 6 & 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Пусть матрицы A и B заданы в формате *RR* (*C*) *U:*

JA = 5 3, 4 3 1 5, 1 6, 4 2
AN = —1 2, 3 3 4 7, —2 —1, 1 1
строка = 1 2 3 4

JB = 1 6 3, 5, 4 2 1, 2 3
BN = 1 5 — 1, —2, 2 6 4, —1 1
строка = 1 2 3 4

Здесь для простоты массивы указателей IA и IB опущены, а индексы строк выписаны в явном виде. Так как в результирующей матрице C 6 столбцов, то для метода переменного переключателя нужен целый вектор IX с 6 позициями. Массиву IX задается нулевое начальное состояние:

IX = 0 0 0 0 0 0

Чтобы получить матрицу C, мы вначале построим JC, сливая строка за строкой массивы JA и JB. Для первой строки это сводится к слиянию списков 5 3 и 1 6 3. Эти числа последовательно вставляются в список JC. Когда в JC зставляется число 5, в позицию IX (5) записывается строчный индекс 1. Затем в JC вставляется 3, а в IX (3) записывается 1. Чтобы избежать повторений в JC, перед вставкой каждого элемента проверяют IX на наличие 1. Например, прежде чем вставлять в JC последнее число 3, мы проверяем значение IX (3) и обнаруживаем, что оно равно 1 (переключатель «включен»); это значит, что 3 уже было включено в JC. Когда строка 1 обработана, мы имеем следующую ситуацию:

JC = 5 3 1 6
IX = 1 0 1 0 1 1

IC легко строится с помощью указателя, отмечающего первую свободную позицию JC. Например, для данного случая IC (l) = i, IC (2) = 5.

Далее мы должны слить вторые строки 4 3 1 5 и 5. Поскольку строчный индекс сейчас равен 2, то в IX мы записываем 2 и про-

веряем IX на наличие 2. Смысл техники переменного переключателя теперь ясен: избегается восстановление нулевого состояния IX после обработки каждой строки; действительно, перед началом обработки *i*-й строки в IX хранятся только числа, меньшие *i*, что не препятствует использованию *i* как значения переключателя для строки *i*. Например, после обработки строки 2 имеем:

JC = 5 3 1 6, 4 3 1 5
IX = 2 0 2 2 2 1

Массив IX готов к обработке строки 3. Заметим здесь, что после обработки каждой строки можно было бы экономно возвращать IX в нулевое состояние. «Экономно» означает, что число операций будет пропорционально числу ненулевых элементов, а не всех элементов строки. Это достигается использованием информации, хранимой в JC. Например, если JC = 5 3 1 6, то мы переопределяем только IX (5), IX (3), IX (I) и IX (6). Эта процедура позволяет экономить машинную память, если IX может быть упакован или храним меньшим числом машинных слов путем описания его как массива типа LOGICAL * 1, т. е. попросту битового массива: каждый бит может хранить только 0 либо 1 и, следовательно, IX должен быть возвращен в нулевое состояние после обработки каждой строки.

Теперь мы должны построить CN—массив, содержащий численные значения ненулевых элементов матрицы *C*. Это численный этап алгоритма. Для накапливания ненулевых элементов каждой строки, хранимых в AN и BN, мы используем расширенный массив X. Начиная со строки 1, мы преждевсего, исходя из JC, засылаем нули в позиции X с номерами 5, 3, 1, 6. Затем, поскольку JA = 5 3 …, то мы пересылаем числа — 1 и 2 из AN в позиции X соответственно 5-ю и 3-ю. Далее, используя JB и BN, мы прибавляем числа 1, 5 и —1 к содержимому позиций X с номерами 1, 6 и 3 соответственно. Наконец, снова привлекая JC, мы выбираем из 5-й, 3-й, 1-й и 6-й позиций X окончательные значения для пересылки их в CN. Остальные строки обрабатываются точно так же. Интересно отметить, что $c_{42} = 0$. Однако, поскольку символический этап алгоритма не имеет никаких сведений о численных значениях элементов, элемент c_{42} будет рассматриваться как ненулевой, и его значение 0 будет включено в CN. Точное взаимное уничтожение происходит редко, и присутствие в NC небольшого числа нулей не вызывает проблем. В некоторых случаях, однако, взаимные сокращения могут быть частыми, например, для матриц, все ненулевые элементы которых равны 1 либо —1. В таких случаях, возможно, было бы разумно заполнять JC и CN одновременно, строка за строкой, немедленно перестраивая их для исключения появляющихся нулей.

Ясно, что описанные идеи весьма просто перенести на случай, когда складываются более, чем две матрицы. Можно также складывать матрицы с различным числом строк нли столбцов. Мы перейдем теперь к описанию символического и численного алгоритмов.

7.I0. АЛГОРИТМ СИМВОЛИЧЕСКОГО СЛОЖЕНИЯ ДВУХ РАЗРЕЖЕННЫХ МАТРИЦ С РАЗМЕРАМИ NxM

Вход: IA, JA портрет первой матрицы в форме RR (C) U
IB, JB портрет второй матрицы в форме RR (C) U
N число строк каждой матрицы
M число столбцов каждой матрицы
Выход: IC, JC портрет суммы матриц в форме RR (C) U
Рабочее пространство: IX массив длины M для переменных переключателей.

```
1.              IP= 1
2.              DO 10 I = l,M
3.      10      IX(I)=0
4.              DO 50 I = l,N
5.              IC(I)=IP
6.              IAA = IA(I)
7.          IAB=IA(I      +1)-1
8.          IF(IAB.LT.IAA)GOTO      30
9.              DO 20 JP=IAA,IAB
10.             J=JA(JP)
11.             JC(IP)=J
12.             IP=IP+1
13.     20  IX(J)  =  1
14.     30      IBA=IB(I)
15.             IBB = IB(I + 1) - 1
16.             lF(IBB.LT.IBA)GOTO 50
17.             DO 40 JP = IBA,IBB
18.              J=JB(JP)
19.             lF(IX(J).EQ.I)GOTO 40
20.             JC(IP)=J
21.             IP = IP+1
22      40      CONTINUE
23.     50      CONTINUE
24.             IC(N+1)=IP
```

В этом алгоритме IP используется как указатель для JC. В строке 1 IP присзаивается начальное значение 1; в строках 12 и 21 значение IP увеличивается всякий раз, как к списку JC добавляется очередной элемент. В строках 5 и 24 IP привлекается также для построения массива указателей IC. Массиву переменных переключателей IX в строках 2 и 3 придается нулевое начальное состояние. Значение I, определяемое в строке 4, указывает номер строки. В цикле DO 20 просматривается строка I первой из заданных матриц: ее столбцовые индексы, если она не-

пуста, записываются в JC (строка 11), а строчный индекс — в IX (строка 13), «включая» тем самым соответствующий переключатель. В цикле DO 40 совершается обход строки I второй матрицы. Для каждого столбцового индекса J, определяемого строкой 18, в строке 19 проверяется значение переключателя: если значение IX (J) равно 1, то переключатель включен. Это значит, что J уже вставлен в список JC и не должен вставляться повторно. В противном случае J добавляется к JC (строка 20). Читатель может подумать, что между строками 21 и 22 должен был бы стоять оператор lX(J) = I, чтобы зарегистрировать факт вставки столбцового индекса J в список JC. Однаков такой записи теперь нет необходимости, потому что при обработке строки 1 это значение J больше не встретится: в представлении строки в массиве JB нет повторений столбцовых индексов.

7.11. АЛГОРИТМ ЧИСЛЕННОГО СЛОЖЕНИЯ ДВУХ РАЗРЕЖЕННЫХ МАТРИЦ С N СТРОКАМИ

Вход:	lA, JA, AN	первая матрица, заданная в форме RR (C) U
	IB, JB, BN	вторая матрица, заданная в форме RR (C) U
	IC, JC	портрет суммы матриц в форме RR (C) U
	N	число строк каждой матрицы
Выход:	CN	численные значения ненулевых элементов суммы матрац
Рабочее пространство:	X	расширенный массив для накапливания ненулевых элементов; длина X равна M, числу столбцов в матрицах

```
 1.            DO 70 I=1,N
 2.            IH = I + 1
 3.            ICA = IC(I)
 4.      ICB         = IC(IH)-1
 5.            IF(ICB.LT.ICA)GOTO 70
 6.            DO 10 IP = ICA,ICB
 7.   10       X(JC(IP))=0.
 8.            IAA = IA(I)
 9       IAB         = IA(IH)-1
10.            IF(IAB.LT.IAA)GOTO 30
11.            DO 20 IP = IAA,IAB
12.   20       X(JA(IP))=AN(lP)
13.   30       IBA = IB(I)
14.      IBB         = IB(IH)
15.            lF(IBB.LT.IBA)GOTO 50
16.            DO 40 IP=IBA,IBB
17.            J = JB(IP)
18.   40       X(J)=X(J) + BN(IP)
19    50       DO 60 IP = ICA,ICB
20.   60       CN(IP)=X(JC(IP))
21.   70       CONTINUE
```

Этот алгоритм использует расширенный массив X для накапливания ненулевых элементов каждой строки заданных матриц, как объяснялось в § 7.9. В строках 6—7 программы засылаются нули в позиции X, соответствующие ненулевым элементам строки I матрицы C. Затем ненулевые элементы строки I первой из складываемых матриц (если таковые имеются) загружаются в CN (строки 11—12). В строке 18 в X добавляются ненулевые элементы строки I второй складываемой матрицы. Наконец, полученные ненулевые элементы загружаются в CN (строка 20). Заметим, что алгоритм использует априорное знание позиций ненулевых элементов (указываемых массивами IC, JC) и оперирует только с такими элементами.

Отметим также, что алгоритм сохраняет заданный порядок в JC и заполняет CN в том же порядке. Поэтому если IC, JC находятся в форме RR (C) O в результате того, что вслед за алгоритмом символического сложения из § 7.10 был дважды применен алгоритм символического транспонирования из § 7.2, то результат численного сложения также будет иметь форму RR (C)O.

7.12. ПРОИЗВЕДЕНИЕ РАЗРЕЖЕННОЙ МАТРИЦЫ ОБЩЕГО ВИДА И ВЕКТОРА-СТОЛБЦА

Теперь мы исследуем операции умножения разреженной матрицы на вектор-столбец и вектора-строки на разреженную матрицу. Результатом этих операций будет заполненный вектор, а не разреженная матрица. Поэтому в алгоритмах умножения вычисления выполняются прямо, без символического этапа.

В зависимости от формы представления заданной матрицы используются различные алгоритмы. Простейший случай — это когда матрица, прямоугольная или квадратная, хранится в форме RR (C) U. Заметим, что даже симметричную матрицу можно задать представлением этого типа и что упорядоченное представление есть частный случай неупорядоченного: алгоритм будет эффективен для матрицы любого типа при условии, что она находится в форме RR (C) U или RR (C) O. Однако для симметричной матрицы более подходят формы RR (DU) U или RR (U) U. В схеме RR (DU) U хранятся посредством массивов IA, JA, AN только ненулевые элементы главной диагонали и верхнего треугольника матрицы; в схеме RR (U) U диагональные элементы хранятся отдельно в массиве AD. Поскольку нижний треугольник матрицы не хранится, значительно сокращаются требования к памяти.

Важным приложением этих алгоритмов является вычисление векторов Ланцоша, необходимое при итерационном решении линейных уравнений методом сопряженных градиентов, а также при вычислении собственных значений и собственных векторов матрицы (гл. 6). Достоинство этих процедур, с вычислительной

точки зрения, состоит в том, что единственная требуемая матричная операция, — это повторное умножение матрицы на последовательность заполненных векторов; сама матрица не меняется. То же свойство имеет метод Гаусса—Зейделя для решения линейных систем (см. гл. 9).

В этом параграфе мы рассмотрим умножение разреженной матрицы общего вида, хранимой в форме RR (C) U посредством массивов IA, JA, AN, на заполненный вектор-столбец *b,* хранимый в массиве B [1] Результатом будет новый заполненный вектор *c,* размещаемый в массиве C:

$$c = Ab. \tag{7.6}$$

Пусть N — число строк матрицы. Если *b* — заполненный, то к его элементам, находящимся в B, доступ может быть произвольным. Если же *b* разреженный и хранится в B в упакованном виде, то нужно вначале сформировать массив указателей IP; тогда, как объясняется в § 1.17, за элементом b_i мы обращаемся к B (IP (i)). Теперь Порядок, в котором накапливаются скалярные произведения, определяется (произвольным) порядком хранения элементов матрицы. Для каждой ее строки I мы находим с помощью IA значения первой IAA и последней IAB позиций, занимаемых элементами строки I в массивах JA и AN. Затем, чтобы вычислить скалярное произведение строки I и вектора B, мы просто просматриваем JA и AN на отрезке от IAA до IAB: каждое значение, хранимое в JA, есть столбцовый индекс и используется-для извлечения из массива B элемента, который должен бытьумножен на соответствующее число из AN. Результат каждого умножения прибавляется к C (I). Нижеследующий алгоритм выполняет ту же работу для случая заполненного вектора *b.*

7.13. АЛГОРИТМ УМНОЖЕНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ ОБЩЕГО ВИДА НА ЗАПОЛНЕННЫЙ ВЕКТОР-СТОЛБЕЦ

Вход:	IA, JA, AN	заданная матрица в форме RR (C) U
	B	заданный заполненный вектор
	N	число строк матрицы
Выход:	C	вектор-произведение размерности N

```
1          DO 20 I=1,N
2.         U = 0.
3.    IAA  =  IA(I)
4.   IAB  =  IA(I  +  1)-1
5.         IF(IAB.LT.IAA)GOTO 20
6.         DO 10 K = IAA,IAB
7.    10   U = U + AN(K)*B(JA(K))
8.    20   C(I) = U
```

[1]) Как видно из дальнейшего, в данном параграфе рассматривается умножение на разреженный вектор. Алгоритм умножения на заполненный вектор приведен в § 7.13. — *Прим. перед.*

В цикле DO 20 по очереди обрабатываются N строк матрицы. В строке 2 переменной U присваивается нулевое начальное значение. U заменяет C (I) при накапливании скалярных произведений (строка 7), потому что трансляторы с Фортрана генерируют более эффективные машинные коды, если используются переменные без индексов. Кроме того, чтобы повысить точность накапливания в строке 7, можно описать U как переменную двойной точности; тогда окончательный результат будет округлен в строке 8 [Gustavson, 1972, p. 51; Wilkinson, 1965, p. 228]. Заметим, что C (I) = 0, если строка I пуста. Смысл оператора в строке 5 в том, чтобы выявлять пустые строки матрицы. В строке б K — указатель позиции в массивах JA и AN, а JA (K) в строке 7 — столбцовый индекс, соответствующий элементу AN (K).

7.I4. ПРОИЗВЕДЕНИЕ ВЕКТОРА-СТРОКИ И РАЗРЕЖЕННОЙ МАТРИЦЫ ОБЩЕГО ВИДА

Пусть вектор-строка b^T хранится в массиве B, а разреженная матрица A общего вида — в массивах IA, JA, AN в форме RR (C) U. Требуется умножить матрицу слева на вектор-строку; результатом будет заполненная строка c^T, записываемая в массиве C:

$$c^T = b^T A. \qquad (7.7)$$

Одна из возможностей — записать это выражение в виде $c = A^T b$ (символ T указывает транспонированную матрицу) и применить алгоритм из § 7.13. Транспонирование разреженных матриц обсуждалось в § 7.2; векторы в транспонировании не нуждаются, поскольку b и b^T, или c и c^T хранятся в памяти машины совершенно одинаковым образом. Однако сложность алгоритма транспонирования матрицы сравнима со сложностью алгоритма умножения; вдобавок, требуется дополнительная память для A^T. Поэтому лучше вычислять $c^T = b^T A$ непосредственно. Пример из § 7.15 показываег, как это можно сделать; соответствующий алгоритм приведен в § 7.16.

7.J5. ПРИМЕР УМНОЖЕНИЯ ЗАПОЛНЕННОЙ СТРОКИ НА РАЗРЕЖЕННУЮ МАТРИЦУ ОБЩЕГО ВИДА

Рассмотрим следующее произведение:

$$(c_1 c_2 c_3) = (b_1 b_2) \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}. \qquad (7.8)$$

Имеем

$$\begin{aligned} c_1 &= b_1 a_{11} + b_2 a_{21}, \\ c_2 &= b_1 a_{12} + b_2 a_{22}, \\ c_3 &= b_1 a_{13} + b_2 a_{23}. \end{aligned} \qquad (7.9)$$

Прямое использование этих равенств затруднительно, так как требует доступа к столбцам матрицы A, элементы которой хранятся по строкам. Поэтому перепишем равенства таким образом:

$$\begin{aligned} c_1 &\leftarrow b_1 a_{11}, \\ c_2 &\leftarrow b_1 a_{12}, \\ c_3 &\leftarrow b_1 a_{13}, \\ c_1 &\leftarrow c_1 + b_2 a_{21}, \\ c_2 &\leftarrow c_2 + b_2 a_{22}, \\ c_3 &\leftarrow c_3 + b_2 a_{23}. \end{aligned} \qquad (7.10)$$

Знак ← указывает фортранный оператор присваивания. Теперь обращение к элементам A происходит в той последовательности, в какой они хранятся в строчном представлении, и алгоритм становится эффективным.

7.16. АЛГОРИТМ УМНОЖЕНИЯ ЗАПОЛНЕННОЙ СТРОКИ НА РАЗРЕЖЕННУЮ МАТРИЦУ ОБЩЕГО ВИДА

Вход:	IA, JA, AN	заданная матрица в форме RR (C) U
	B	заданный заполненный вектор
	N	число строк матрицы
	M	число столбцов матрицы
Выход:	C	вектор-произведение

```
 1.          DO 10 I=1,M
 2.    10    C(I)=0.
 3.          DO 30 I=1,N
 4.          IAA = IA(I)
 5.    IAB = IA(I + 1) - 1
 6.          lF(IAB.LT.IAA)GOTO 30
 7.          Z=B(I)
 8.          DO 20 K = IAA,IAB
 9.          J = JA(K)
10.    20    C(J) = C(J) + AN(K)*Z
11.    30    CONTINUE
```

Этот алгоритм выполняет вычисления, описанные в предыдущем параграфе. В цикле DO 10 элементам массива C присваиваются нулевые начальные значения. В цикле DO 30 по очереди обрабатываются N строк матрицы. Пустые строки выявляются и пропускаются в строке 6 программы. В цикле DO 20 просматриваются все элементы каждой строки I. В строке 9 J — столбцовый индекс такого элемента, AN (K) в строке 10 — его численное значение. Значение множителя B (I), общее для всех элементов строки I, присваивается в строке 7 переменной Z и используется для вычисления произведений в строке 10. Результаты частичных скалярных умножений накапливаются в C (J). Заметим, что значения J следуют друг за другом, вообще говоря, в произвольном порядке, поскольку, по нашему предположению, представление

неупорядочено; и обращения к элементам C также будут происходить в случайном порядке, В данном случае в отличие от алгоритма из § 7.13 нельзя накапливать скалярные произведения посредством безындексной переменной.

7.!7. ПРОИЗВЕДЕНИЕ СИММЕТРИЧНОЙ РАЗРЕЖЕННОЙ МАТРИЦЫ И ВЕКТОРА-СТОЛБЦА

Здесь мы обсудим умножение симметричной разреженной матрицы, хранимой посредством массивов IA, JA, AN, AD в форме RR (U) U, и заполненного вектора, находящегося в массиве B. Алгоритм умножения весьма сходен с описанным в § 7.13; отличие состоит в том, что теперь каждое число в AN характеризует два симметричных элемента матрицы, которые должны быть умножены на *различные* элементы массива B и (после умножения) прибавлены к *различным* элементам массива C. Как следствие, накапливания должны производиться непосредственно в массиве C. Алгоритм приведен в следующем параграфе.

7.18. АЛГОРИТМ УМНОЖЕНИЯ СИММЕТРИЧНОЙ РАЗРЕЖЕННОЙ МАТРИЦЫ НА ЗАПОЛНЕННЫЙ ВЕКТОР-СТОЛБЕЦ

Вход: IA, JA, AN, AD заданная матрица в форме RR (U) U
B заданный заполненный вектор-столбец
N порядок матрицы
Выход; C вектор-произведение

```
 1          DO  10  I=1,N
 2     10   C(I) = AD(I)*B(I)
 3.         DO 30 I = 1,N
 4.         IAA=IA(J)
 5.         IAB = IA(I + 1) - 1
 6.         IF(IAB.LT.IAA)GOTO 30
 7.         U = C(I)
 8.         Z = B(I)
 9.         DO 20 K = IAA,IAB
10.         J=JA(K)
11.         U = U + AN(K)*B(J)
12.    20   C(J)=C(J) + AN(K)*Z
13.         C(I) = U
14.    30   CONTINUE
```

В цикле DO 10 элементам массива C присваиваются начальные значения (строка 2). В цикле DO 30 по очереди обрабатываются N строк матрицы. В строке 6 выявляются и обходятся пустые строки. Индекс K в строке 9 пробегает позиции всех ненулевых элементов непустой строки I. Каждое число AN (K) играет двойную роль: элемента I, J матрицы и элемента J, I; J определено в строке 10. Поэтому для каждого K нужно пересчитывать два скалярных произведения (строки 11 и 12). Для повы-

шения эффективности в цикле DO 20 вместо C (I) и B (I) используются U и Z.

Имеются еще несколько вариантов умножения разреженной матрицы на вектор. Опираясь на предыдущее обсуждение, читатель легко сможет записать соответствующие алгоритмы. Однако такие случаи редко возникают на практике.

Стоит отметить, что в случае симметричной матрицы произведения типа «строка на матрицу» простым транспонированием могут быть сведены к типу «матрица на столбец». В действительности, представления в памяти вектора-строки и вектора-столбца одинаковы, а симметричная матрица совпадает со своей транспонированной. Следовательно, алгоритмом данного параграфа можно пользоваться без каких-либо изменений.

7.19. УМНОЖЕНИЕ РАЗРЕЖЕННЫХ МАТРИЦ

В этом параграфе мы изучим алгоритмы, используемые для вычисления произведения двух разреженных матриц. Пусть даны $p \times q$ матрица A и $q \times r$ матрица B. Матрица-произведение C имеет размеры $p \times r$ и элементы

$$c_{ik} = \sum_{j=1}^{q} a_{ij} b_{jk}, \quad i = 1, \ldots, p, \quad k = 1, \ldots, r. \tag{7.11}$$

Эта формула выражает элемент c_{ik} как скалярное произведение i-й строки A и k-го столбца B. Однако, если B задана строчным форматом, то к ее столбцам нет прямого доступа. Одно возможное решение этой проблемы состоит в транспонировании B посредством алгоритма из § 7.3. Тогда уравнение (7.11) примет вид

$$c_{ik} = \sum_{j=1}^{q} a_{ij} (B^{\mathrm{T}})_{kj}, \tag{7.12}$$

использующий только строки A и B^{T}. Сейчас будет описан более эффективный и элегантный метод, не требующий транспонирования B и позволяющий задание обеих матриц A и β посредством строчного формата [Gustavson, 1976c] [1]. Основная идея метода — изменить порядок вычисления попарных произведений в равенстве (7.11): при фиксированных i и j элемент a_{ij} умножается на все элементы b_{jk} j-й строки B, и эти произведения прибавляются к соответствующим позициям расширенного вещественного накопителя X. Когда таким образом обработаны все элементы i-й строки A, массив X содержит полную i-ю строку C. Простую иллюстрацию этого приема дает пример в § 7.20. Симво-

[1]) См. также работу Jennings A. A sparse matrix scheme for the computer analysis of structures, Int. J, Computer Mathematics, 1968, 2, 1—21.

лический алгоритм приведен в § 7.21, численный алгоритм — в § 7.22. Символическая и численная обработка могут выполняться и одновременно. Символический алгоритм опирается на метод переменного переключателя (объясняемый в § 1.14), причем IX служит массивом переключателей. Об использовании расширенного накопителя X в численном алгоритме речь шла в § 1.15. Другая возможность — использовать в качестве накопителей разреженные строки матрицы-произведения, привлекая расширенный целый массив указателей (см. § 1.16). Результирующая матрица C получается в неупорядоченном представлении, даже если представления A и B были упорядочены. Чтобы упорядочить представление C, можно дважды применить алгоритм численного транспонирования. Если символическое и численное умножение производятся раздельно, то лучше упорядочить только портрет C двойным применением алгоритма символического транспонирования. Так как при численной обработке порядок сохраняется, то конечный результат будет упорядоченным.

Можно взглянуть на умножение матриц и по-другому. Представим $p \times q$ матрицу A как набор из q p-мерных векторов-столбцов, а $q \times r$ матрицу B как набор из q r-мерных векторов-строк:

$$A = (a_1, a_2, \ldots, a_q),$$

$$B = \begin{bmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_q^T \end{bmatrix}. \qquad (7.13)$$

Произведение вида $a_i b_i^T$ представляет собой $p \times r$ матрицу ранга 1. Матрицы ранга 1 рассматривались в § 2.2. Согласно (7.11), произведение $C = AB$ можно записать в виде суммы q матриц ранга 1:

$$AB = \sum_{i=1}^{q} a_i b_i^T. \qquad (7.14)$$

Одно из приложений уравнения (7.14) — подсчет числа умножений, необходимых для вычисления C при разреженных A и B. Пусть $n(v)$ — число ненулевых элементов вектора v. Ясно, что для построения матрицы $a_i b_i^T$ *ранга* 1 требуется $n(a_i)\, n(b_i)$ умножений, а для полного произведения AB нужны

$$n(AB) = \sum_{i=1}^{q} n(a_i)\, n(b_i) \qquad (7.15)$$

умножений. Число сложений будет таким же, если засчитывать сложения с нулями.

7.20. ПРИМЕР УМНОЖЕНИЯ ДВУХ МАТРИЦ, ХРАНИМЫХ ПО СТРОКАМ

Рассмотрим следующее матричное произведение:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}. \tag{7.16}$$

Имеет

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21}, \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22}, \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21}, \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22}. \end{aligned} \tag{7.17}$$

Вместо вычисления произведений этим способом будем придерживаться следующего порядка:

$$\begin{aligned} x_1 &\leftarrow a_{11}b_{11}, \\ x_2 &\leftarrow a_{11}b_{12}, \\ x_1 &\leftarrow x_1 + a_{12}b_{21}, \\ x_2 &\leftarrow x_2 + a_{12}b_{22}, \\ c_{11} &\leftarrow x_1, \\ c_{12} &\leftarrow x_2, \\ x_1 &\leftarrow a_{21}b_{11}, \\ x_2 &\leftarrow a_{21}b_{12}, \\ x_1 &\leftarrow x_1 + a_{22}b_{21}, \\ x_2 &\leftarrow x_2 + a_{22}b_{22}, \\ c_{21} &\leftarrow x_1, \\ c_{22} &\leftarrow x_2. \end{aligned}$$

Знак ← указывает фортранный оператор присваивания. Заметим, что каждый элемент первой матрицы последовательно умножается на все элементы строки второй матрицы, легко доступные, если вторая матрица задана в строчном формате.

7.21. АЛГОРИТМ СИМВОЛИЧЕСКОГО УМНОЖЕНИЯ ДВУХ РАЗРЕЖЕННЫХ МАТРИЦ, ЗАДАННЫХ В СТРОЧНОМ ФОРМАТЕ

Вход:	IA, JA	портрет первой матрицы в форме RR (C) U
	IB, JB	портрет второй матрицы в форме RR (C) U
	NP	число строк первой матрицы
	NQ	число столбцов первой матрицы и строк второй
	NR	число столбцов второй матрицы
Выход;	IC, JC	портрет матрицы C в форме RR (C) U
Рабочее пространство:	IX	массив длины NP для переменных переключателей

```
1.              IP=1
2.              DO 10 I = l,NR
3.    10        IX(I)=0
4.    DO   40   I   =   l,NP
5.              IC(I) = IP
6.     IAA      =      IA(I)
7.    IAB   =   IA(I   +   1 ) - 1
8.              IF(IAB. LT. IAA)GO TO 40
9.              DO 30 JP = IAA,IAB
10.             J = JA(IP)
11.             IBA = IB(J)
12.             IBB = IB(J + 1 ) - 1
13.             IF(IBB.LT.IBA)GOTO30
14.             DO 20 KP = IBA,IBB
15.             K=JB(KP)
16.             IF(IX(K).EQ.I)GOTO 20
17.             JC(IP)=K
18.     IP      =      IP+1
19.       IX(K)        = I
20.     20      CONTINUE
21.     30      CONTINUE
22.     40      CONTINUE
23.             IC(NP + 1)=IP
```

В этом алгоритме IP — указатель первой свободной позиции в JC. В строке 1 IP получает начальное значение и затем увеличивается всякий раз, как в список JC вставляется новый элемент (строка 18). IP используется также для заполнения массива IC (строки 5 и 23). Массиву переменных переключателей IX в строках 2 и 3 придается нулевое начальное состояние. В цикле DO 40 по очереди обрабатываются NP строк первой заданной матрицы. В цикле DO 30 просматриваются ненулевые элементы строки I (строка 9 алгоритма); их столбцовые индексы J определяются в строке 10. Для каждого J цикл DO 20 просматривает ненулевые элементы строки J второй заданной матрицы (строка 14); соответствующие столбцовые индексы K определяются в строке 15. Строка 16 — это проверка массива *переключателей* IX с целью выяснения, не было ли число K уже добавлено к описанию строки I в списке JC. Если нет, то K добавляется к JC (строка 17); этот факт регистрируется в IX (строка 19).

7.22. АЛГОРИТМ ЧИСЛЕННОГО УМНОЖЕНИЯ ДВУХ РАЗРЕЖЕННЫХ МАТРИЦ, ЗАДАННЫХ В СТРОЧНОМ ФОРМАТЕ

Вход:	IA, JA, AN	первая заданная матрица в форме RR (C) U
	IB, JB, BN	вторая заданная матрица в форме RR (C) U
	IC, JC	портрет матрицы C в форме RR (C) U
	NP	число строк матриц A и C
Выход:	CN	*численные* значения ненулевых элементов матрицы C

Рабочее пространство: расширенный накопитель длины, равной X числу столбцов матрицы C

```
 1.            DO 50 I = 1,NP
 2.            ICA=IC(I)
 3.            ICB = IC(I + 1 ) - 1
 4.            IF(ICB. LT. ICA)GO TO 50
 5:            DO 10 J = ICA.ICB
 6.    10      X(JC(J))=0.
 7.            IAA = IA(I)
 8.      IAB=IA(I       + 1 ) - 1
 9.            DO 30 JP = IAA,IAB
10.            J=JA(JP)
11.            A = AN(JP)
12.            IBA =IB(J)
13             IBB = IB(J+1) - 1
14.            IF(IBB.LT.IBA)GOTO 30
15.            DO 20 KP = IBA,IBB
16.            K=JB(KP)
17.    20  X(K) = X(K) + A*BN(KP)
18.    30      CONTINUE
19.            DO 40 J=ICA,ICB
20.    40      CN(J)=X(JC(J))
21.    50      CONTINUE
```

В этом алгоритме I указывает номер строки первой заданной матрицы и результирующей матрицы. Для каждой строки I засылаются нули в те позиции расширенного накопителя X, которые соответствуют ненулевым элементам матрицы *C* (строки 5 и 6). Заметим, что программа достигает строки 9, только если строка I матрицы C непуста; это в свою очередь означает, что непуста строка I матрицы *A*. Поэтому IAB $\geq$ IAA, и оператор IF(IAB.LT.IAA) GO TO 40 перед строкой 9 (см. § 7.1) в данном случае не нужен. В цикле DO 30 просматриваются ненулевые элементы строки I первой матрицы. Значение элемента I, J присваивается безындексной переменной A (строка 11). Затем в цикле DO 20 сканируются ненулевые элементы строки J второй матрицы, для каждого элемента J, K вычисляется частичное произведение (как это описано в § 7.20) и добавляется к X (K). Когда цикл DO 30 завершен, массив X содержит в себе строку I результирующей матрицы, которая теперь загружается в CN (строки 19 и 20). Эта процедура обсуждалась в § 1.15, Иначе накопление можно было бы выполнять прямо в CN, используя расширенный целый массив указателей (см. § 1.16). Запись соответствующего алгоритма предоставляется читателю в качестве упражнения.

Огметим еще, что описанный алгоритм не меняет упорядочения в JC. Поэтому, если IC, JC заданы в форме RR (C) O (явившейся результатом двойного применения алгоритма символического транспонирования из § 7.3 после того, как проработал алгоритм символического умножения из § 7.21), то и результат численного умножения будет иметь форму RR (C) O: ведь порядок сохраняется.

7.23. ТРЕУГОЛЬНОЕ РАЗЛОЖЕНИЕ РАЗРЕЖЕННОЙ СИММЕТРИЧНОЙ МАТРИЦЫ, ЗАДАННОЙ В СТРОЧНОМ ФОРМАТЕ

В этом параграфе мы будем предполагать, что разреженная симметричная матрица A задана в верхнем строчном разреженном формате и нужно провести для A гауссово исключение, выбирая главные элементы на главной диагонали в правильном порядке. Это — нетривиальная задача, поскольку подлежащие исключению ненулевые элементы находятся в нижнем треугольнике A, который явным образом не хранится. Кроме того, лоскольку представление матрицы —строчное, доступ кстолбцамзатруднен. Схемагауссоваисключения по строкамлучшеприспособлена для матрицы, заданной строчным форматом.чем более распространенная схема исключения по столбцам. В строчной схеме из каждой строки I матрицы A поэлементно вычитаются кратные всех предыдущих строк с тем, чтобы у новой строки I слева от диагонали были только нули. Остальные элементы строки делятся на диагональный, что порождает строку верхнего треугольного множителя U; диагональные элементы хранятся отдельно, составляя диагональную матрицу D.

Треугольное разложение разреженной матрицы выполняется вначале символически, а затем численно. Для символического разложения нужен портрет IA, JA матрицы A в неупорядоченном представлении; будет получен портрет IU, JU матрицы U, также неупорядоченный. Однако для численного разложения портрет IU, JU требуется упорядочить, хотя IA, JA, AN могут оставаться в неупорядоченном представлении. Отсюда следует, что после символического разложения (и перед численным) нужно дважды применить алгоритм символического транспонирования из § 7.3, чтобы упорядочить IU, JU.

Алгоритмические трудности, возникающие при разложении разреженной симметричной матрицы, заданной верхним представлением, лучше всего разъяснить с помощью примера.

$$A = \begin{array}{c|ccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \hline 1 & x & 0 & 0 & 0 & 0 & x & 0 & 0 & 0 & x & 0 \\ 2 & 0 & x & 0 & x & 0 & 0 & 0 & 0 & x & 0 & 0 \\ 3 & 0 & 0 & x & 0 & x & 0 & x & 0 & 0 & 0 & x \\ 4 & 0 & x & 0 & x & 0 & 0 & x & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & x & 0 & x & 0 & 0 & x & 0 & 0 & 0 \\ 6 & x & 0 & 0 & 0 & 0 & x & 0 & 0 & x & 0 & 0 \\ 7 & 0 & 0 & x & x & 0 & 0 & x & 0 & 0 & 0 & x \\ 8 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & 0 & 0 & x \\ 9 & 0 & x & 0 & 0 & 0 & x & 0 & 0 & x & 0 & x \\ 10 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x \\ 11 & 0 & 0 & x & 0 & 0 & 0 & x & x & x & x & x \end{array} \qquad (7.18)$$

В этой симметричной матрице для простоты не указаны численные значения ненулевых элементов.

Треугольное разложение этой матрицы начнется с четвертой строки, поскольку в предыдущих строках все элементы слева от диагонали равны нулю. Из строки 4 поэлементно вычитаем строку 2, умноженную на константу, подобранную так, чтобы аннулировать элемент a_{42}. При этом в строке 4 появляется новый ненулевой элемент, а именно элемент 4, 9. Далее из строки 5 вычитаем третью строку, чтобы исключить a_{53}; в позициях 5, 7 и 5, 11 возникнут новые ненулевые элементы. Затем исключаем a_{61}, что приводит к появлению элемента в позиции б, 10. Предполагая, что разложение доведено до этого места, мы подробно рассмотрим обработку строки 7 при исключении элементов a_{73} и a_{74}.

Первая трудность заключается в том, что в машине представлены только верхний треугольник и диагональ *A*. У нас нет информации о том, что элементы a_{73} и a_{74} не равны нулю и должны быть исключены. Нет также информации о том, что при вычитании строки 3 появится новый ненулевой элемент в позиции 7, 5 и его тоже придется исключать. Зато мы знаем, что не равны нулю симметричные элементы a_{37}, a_{47} и что при обработке строки 5 в позиции 5, 7, симметричной к 7, 5, возник новый ненулевой элемент. Другими словами, чтобы найти подлежащие исключению ненулевые элементы строки 7, лежащие левее диагонали, мы должны исследовать ненулевые наддиагональные элементы *столбца 7*.

Далее, мы знаем, что после обработки строки 7 все ее элементы слева от диагонали будут равны нулю. Поэтому вычислять эти элементы нет нужды. Следует вычислять только элементы справа от диагонали и диагональный элемент, а для этого требуются лишь элементы предыдущих строк, расположенные в столбце 7 или справа от него. В общем случае придется выполнять операцию: «для заданных *i, j*, где $j > i$, найти элементы строки *i,* у которых столбцовые индексы $\geq j$». Для эффективного ее выполнения нужно, чтобы представления строк *U* были упорядочены. Как мы увидим ниже, это требование относится только к численному этапу гауссова исключения, но не к символическому.

Предположим теперь, что все шаги, необходимые для локализации нужных ненулевых элементов, уже проведены, и мы имеем следующие списки столбцовых индексов для элементов, находящихся в столбце 7 или правее:

строка 3: 7 11
строка 4: 7 9
строка 5: 7 8 11
строка 7: 7 11

Чтобы получить портрет новой строки 7, эти списки столбцовых индексов нужно слить. Если для *этого* использовать метод переменного переключателя из § 1.14, то результатом будет неупорядоченное представление

новая строка 7: 7 11 9 8

Однако здесь мы можем извлечь выгоду из следующего свойства гауссова исключения по строкам [Rose et al., 1976]: необходимо и достаточно учитывать, кроме самой строки 7, только те строки, для которых ненулевой элемент в столбце 7 является *первым* ненулевым элементом данной строки справа от диагонали. В данном случае можно не учитывать строку 3, потому что первый ее ненулевой элемент справа от диагонали стоит в позиции 3, 5, т. е. левее столбца 7. В действительности из $a_{35} \neq 0$ по симметрии следует $a_{53} \neq 0$. Это значит, что элемент 5, 3 аннулирован при обработке строки 5 путем вычитания строки 3; но тогда все ненулевые элементы строки 3 были слиты с элементами строки 5. Поэтому, учитывая ненулевые элементы строки 5, мы в то же время учитываем ненулевые элементы, стоящие в строке 3 справа от столбца 7. Итак, достаточно рассматривать только следующие портреты:

строка 4: 7 9
строка 5: 7 8 11
строка 7: 7 11

При их слиянии получаем правильный новый портрет:

новая строка 7: 7 9 8 11

Заметим, что эти соображения, относящиеся к символическому этапу гауссова исключения, не требуют, чтобы представления строк были упорядоченными. В самом деле, мы должны лишь регистрировать, у каких строк *первый* ненулевой элемент справа от диагонали находится в данном столбце. Это означает, что остальные ненулевые элементы таких строк расположены *правее* рассматриваемого столбца; следовательно, чтобы сформировать портрет новой строки (используя метод переменного переключателя), нам нужны *все* эти элементы, каков бы ни был их порядок. Таким образом, символический этап гауссова исключения не требует упорядоченности представления. Это выгодно для нас, поскольку, как видно из примера, символический алгоритм порождает неупорядоченные портреты.

Для регистрации строк, имеющих первый ненулевой элемент в данном столбце, мы должны сопоставить каждому столбцу свое множество строк. Например, строки 4 и 5 принадлежат множеству, ассоциированному со столбцом 7. На практике наименьший столбцовый индекс каждой строки определяется одновременно с формированием ее портрета, после чего к множеству.

ассоциированному с минимальным столбцом, добавляется номер данной строки. Например, при сборке портрета строки 7 выясняется, что справа от диагонали наименьший столбцовый индекс равен 8, и в множество, ассоциированное со столбцом 8, немедленно вставляется номер 7. При обработке строки 8 в соответствующем множестве будет обнаружен строчный индекс 7 и будет использован портрет строки 7.

Заметим еще, что множества строк, ассоциированные с разными столбцами, не пересекаются, т. е. каждая строка принадлежит одному и только одному столбцу. Поэтому все множества строчных индексов могут храниться одним массивом IP длины N (N — порядок матрицы) в форме кольцевых цепей (см. § 1.3). На самом деле при обработке строки I для хранения множеств используются только I — 1 начальных позиций IP, потому что к этому времени в множества вставлены только номера строк 1, ..., (I — 1). Остальные позиции 1Pхранят столбцовые указатели входа для всех цепей: в данный момент интерес представляют только столбцы с номерами от I до N. Дальнейшие подробности можно найти в § 7.25, где будет представлен и разъяснен алгоритм символического гауссова исключения.

Массив IU также можно использовать двояко. При обработке строки L в позициях IU от L до I — 1 находятся указатели расположения одноименных строк в массиве JU. Остальные позиции свободны. Так как в список JU теперь будут вставляться лишь столбцовыеиндексы,неменьшие I, то позиции IU от I до N используются как расширенный массив переменных переключателей.

7.24. ЧИСЛЕННОЕ ТРЕУГОЛЬНОЕ РАЗЛОЖЕНИЕ РАЗРЕЖЕННОЙ СИММЕТРИЧНОЙ МАТРИЦЫ, ЗАДАННОЙ В СТРОЧНОМ ФОРМАТЕ

В этом параграфе мы будем рассматривать ту же матрицу, что и в § 7.23. Предположим, что символическое разложение и упорядочение портрета уже проделаны и IU, JU находятся в упорядоченном строчном верхнем формате. Теперь нас будет интересовать численный этап гауссова исключения. Предположим еще, что исключение проведено вплоть до строки 6, и исследуем процесс обработки строки 7. Мы знаем, что ненулевые элементы строки 7 расположены в позициях 7, 8, 9 и 11. Чтобы вычислить их значения, нужно просмотреть столбец 7, установить, что строки 3, 4 и 5 имеют ненулевые элементы в этом столбце, найти в этих строках ненулевые элементы, столбцовые индексы которых не меньше 7, наконец, умножить эти элементы на подходящие константы и вычесть из элементов строки 7. В описанной процедуре несколько шагов, и три из них мы изучим детально.

(1) Найти элементы данной строки (скажем, строки 3), у которых столбцовые индексы больше или равны номеру обрабатываемой строки (скажем, строки 7). Для описания строки 3 в масси-

вах JU и UN начальной и конечной позициями являются IU (3) и IU (4) — 1. Строка 3 упорядочена, и ее ненулевые элементы стоят в столбцах:

строка 3: 5 7 11

Однако нам нужна только та часть строки 3, которая начинается со столбца 7. Поэтому требуется другой указатель начала (назовем его IUP (3)); в то же время IU (4)—1 по-прежнему можно использовать как указатель конца. В момент обработки строки 7 IUP (3) указывает позицию ненулевого элемента строки 3, стоящего в столбце 7. После того как строка 3 использована для обработки строки 7, значение IUP (3) увеличивается на единицу. Так как строка 3 упорядочена, IUP (3) будет теперь указывать позицию *следующего* ненулевого элемента строки 3, т. е. элемента из столбца 11; это значение *сохранится* до обработки строки 11. Дело в том, что при обработке строк 8, 9, 10 строка 3 не будет использована, поскольку не имеет ненулевых элементов s столбцах 8, 9 и 10, а значит и значение IUP (3) не будет изменено.

(2) Выяснить, какие строки имеют ненулевые элементы в данном столбце, для определенности в столбце 7. В отличие от ситуации, рассматривавшейся в § 7.23, где для символического процесса нужны были только строки, у которых в столбце 7 находится первый ненулевой элемент, теперь требуются все строки, имеющие ненулевой элемент в этом столбце. Здесь также в множества, ассоциированные со столбцами, приписываются строки, но если прежде использованная строка в дальнейшем просто игнорировалась, то теперь она приписывается к множеству, соответствующему *следующему* ненулевому элементу этой строки. За исключением этой детали все множества хранятся в первых I—1 позициях массива IP размерности N, а остальные позиции P используются для столбцовых указателей входа в множества.

(3) Вычесть кратные отрезков назначенных строк из обрабатываемой строки, скажем строки 7. Элементы очередной строки, скажем строки 3, нужно умножить на отношение элементов a_{37} (который вследствие симметрии равен a_{73}) и a_{33}. Так как позицию a_{37} указывает IUP (3), а. a_{33} находится в DI (3) (в действительности BDI (3) записано число a_{33}^{-1}, потому что в DI удобно хранить не сами элементы диагональной матрицы D, а обратные к ним числа), то упомянутое отношение легко может быть вычислено. Для сложения разреженных вещественных векторов требуется еще расширенный накопитель. В таком качестве используются позиции I, ... N массива DI, которые при обработке строки I свободны. Этим путем достигается очень эффективное использование имеющейся памяти.

В § 7.25 и 7.26 будут представлены алгоритмы символического и численного треугольного разложения.

7.25. АЛГОРИТМ СИМВОЛИЧЕСКОГО ТРЕУГОЛЬНОГО РАЗЛОЖЕНИЯ СИММЕТРИЧНОЙ РАЗРЕЖЕННОЙ МАТРИЦЫ A

Вход: IA, JA — портрет заданной матрицы A в форме RR (U) U
N — порядок матриц A и U
Выход: IU, JU — портрет матрицы U в форме RR (U) U
Рабочее пространство: IP — массив длины N для ценных списков строк, ассоциированных со столбцами
Массив IU используется также в качестве массива переменных переключателей.

```
      NM = N - 1
      NH = N + 1
      DO 10 I = 1, N
      IU(I) = 0
10    IP(I) = 0
      JP = I
      DO 90 I = 1, NM
      JPI = JP
      JPP = N + JP - I
      MIN = NH
      IAA = IA(I)
      IAB = IA(I + 1) - 1
      IF(IAB.LT.IAA) GOTO 30
      DO 20 J = IAA, IAB
      JJ = JA(J)
      JU(JP) = JJ
      JP = JP + 1
      IF(JJ.LT.MIN) MIN = JJ
20    IU(JJ) = 1
30    LAST = IP(I)
      IF(LAST.EQ.0) GOTO 60
      L = LAST
40    L = IP(L)
      LH = L + I
      IUA = IU(L)
      IUB = IU(LH) - 1
      IF(LH.EQ.I) IUB = JPI - 1
      IU(I) = I
      DO 50 J = IUA, IUB
      JJ = JU(J)
      IF(IU(JJ).EQ.I) GOTO 50
      JU(JP) = JJ
      JP = JP + 1
      IU(JJ) = I
      IF(JJ.LT.MIN) MIN = JJ
50    CONTINUE
      IF(JP.EQ.JPP) GO TO 70
      IF(L.NE.LAST) GOTO 40
60    IF(MIN.EQ.NH) GOTO 90
70    L = IP(MIN)
      IF(L.EQ.0) GOTO 80
      IP(I) = IP(L)
      IP(L) = 1
      GO TO 90
80    IP(MIN) = I
      IP(I) = I
90    IU(I) = IPI
      IU(N) = JP
      IU(NH) = JP
```

Принципы, на которых основан алгоритм, описаны в § 7.23. Здесь мы ограничимся некоторыми указаниями для лучшего понимания программы. В строках 4 и 5 устанавливается нулевое начальное состояние в массиве переменных переключателей IU и массиве цепных списков IP. Переменная JP указывает первую свободную позицию в списке JU; она получает начальное значение 1 в строке 6 и увеличивается на единицу всякий раз, как в JU вставляется новый столбцовый индекс (строки 17 и 33).

В цикле DO 90 по очереди обрабатываются строки матрицы. Для каждой строки I текущее значение JP временно присваивается переменной JPI (строка 8), а впоследствии, когда позиция I массива IU уже не используется для переключателя, это значение записывается в IU (I) для постоянного хранения (строка 47). Назначение цикла DO 20 — перенести столбцовые индексы JJ, соответствующие строке I матрицы A, из JA в JU. Как обычно, в массив переключателей IU при этом записывается текущий переключатель I (строка 19). Переменная MIN, начальное значение которой присвоено в строке 10, выделяет наименьший столбцовый индекс.

Значение указателя входа в кольцевую цепь, перечисляющую строки, ассоциированные со столбцом I, выбирается из IP (I) в строке 20 и присваивается переменной LAST. Фактически LAST — это номер одной из ассоциированных строк, а именно «последней» строки. Строкой 21 предусмотрен случай, когда со столбцом I не ассоциирована никакая строка. В строке 23 L — «следующий» элемент цепи. Необходимость в строке 27 объясняется тем обстоятельством, что позиция IU (I) пока не определена как указатель. В строке 28 в IU (I) записывается переключатель I, чтобы воспрепятствовать вставке I в JU в качестве столбцового индекса. В цикле DO 50 просматривается строка L матрицы *U*. В строке 31 проверяется массив переключателей Ш; при необходимости столбцовый индекс JJ вставляется в JU (строка 32); новая вставка JJ в JU предупреждается записью I в IU (JJ) в строке 34. В следующей строке вычисляется минимальный столбцовый индекс.

Переменная JPP, определяемая в строке 9, — это значение, которого бы достигла JP, если бы в строке I *все* элементы справа от диагонали были не равны нулю. Эта возможность проверяется в строке 37, и если тест дает положительный результат, то для экономии времени счета дальнейшие вычисления пропускаются. Такая ситуация может возникать часто, особенно для последних строк матрицы.

Если L — не «последняя» строка цепи, ассоциированной со столбцом I, то управление передается из строки 38 в строку 23; здесь выбирается следующая строка цепи. При исчерпании цепи

будет выполняться строка 39. Равенство MIN = NH означало бы, что справа от диагонали строка I пуста; в этом случае управление передается в строку 47. В протизоположном случае в строке 1 имеются один или несколько ненулевых элементов, и следует вставить номер I в цепь, ассоциированную со столбцом MIN. В строке 40 выделяется указатель L входа в эту цепь, в строке 42 цепь разбивается и в строке 43 вставляется номер I, Если цепь пока пуста, то управление передается из строки 41 в строку 45, где определяется указатель входа в цепь и создается сама цепь из единственного элемента I (строка 46).

7.26. АЛГОРИТМ ЧИСЛЕННОГО ТРЕУГОЛЬНОГО РАЗЛОЖЕНИЯ СИММЕТРИЧНОЙ ПОЛОЖИТЕЛЬНО ОПРЕДЕЛЕННОЙ РАЗРЕЖЕННОЙ МАТРИЦЫ A

Вход:	IA, JA, AN, AD	матрица A, заданная схемой RR (U) U
	IU, JU	портрет матрицы U в форме RR (U) O
	N	порядок матриц A и U
Выход:	UN	численные значения ненулевых элементов матрицы U в форме RR (U) O
	DI	обратная к диагональной матрице D
Рабочее пространство:	IP	массив длины N для цепных списков строк, ассоциированных со столбцами
	IUP	массив длины N для вспомогательных указателей к отрезкам строк
	DI	используется как расширенный накопитель

```
      DO l0 J=1,N
10    IP(J)=0
      DO 130 I=1,N
      IH=I + 1
      IUA=IU(I)
      IUB=IU(IH)-1
      IF(IUB.LT. IUA)GO TO 40
      DO 20 J=IUA,IUB
20    DI(JU(J))=0.
      IAA=IA(I)
      IAB=IA(IH)-1
   IF(IAB.LT.IAA)GOTO        40
      DO 30 J=IAA,IAB
30    DI(JA(J))=AN(J)
40    DI(I)=AD(I)
      LAST = IP(I)
      lF(LAST .EQ.0)GO TO 90
      LN = lP(LAST)
50    L = LN
      LN=IP(L)
      IUC=IUP(L)
      IUD = IU(L + 1) I
      UM = UN(IUC)*DI(L)
      DO 60 J = IUC,IUD
      JJ=JU(J)
60    DI(JJ) = DI(JJ)-UN(J)*UM
      UN(IUC)=UM
      IUP(L)=IUC+1
      IF(IUC. EQ. IUD)G0 TO 80
```

```
30.                J = JU(IUC + 1)
31.                JJ=IP(J)
32.                IF(JJ.EQ.0)GO TO 70
33.                IP(L)=IP(JJ)
34.                IP(JJ)=L
35.                GO TO 80
36.     70         IP(J)=L
37.                IP(L)=L
38.     80         IF(L.NE.LAST)GOTO 50
39.     90         DI(I)=I./DI(I)
40.                IF(IUB.LT.IUA)GOTO 120
41.                DO 100 J=IUA,IUB
42.    100         UN(J)=DI(JU(J))
43.                J = JU(IUA)
44.                JJ = IP(J)
45.                IF(JJ.EQ 0)GOTO 110
46.                IP(I) = IP(JJ)
47.          IP(JJ)        = I
48           GOTO          120
49.     110        IP(J)=I
50.                IP(I) = I
51.     120        IUP(I)=IUA
52.    130         CONTINUE
```

В строках 1 и 2 этого алгоритма массиву IP придается нулевое начальное состояние, означающее, что все цепные списки строк пока пусты. В цикле DO 130 обрабатываются строки матриц *A* и *U,* определяемые индексом I. Если строка I матрицы *U* не содержит ненулевых внедиагональных элементов, то управление передается из строки 7 в строку 15, где обрабатывается диагональный элемент, по предположению, всегда положительный. В противном случае засылаются нули в позиции расширенного массива DI, соответствующие ненулевым элементам (строка 9), а затем (строка 14) в DI загружаются ненулевые элементы строки I матрицы *A* (если таковые имеются). Диагональный элемент строки I загружается в строке 15.

Значение указателя входа в кольцевую цепь, перечисляющую строки, ассоциированные со столбцом I, выбирается из IP (I) в строке 16 и присваивается переменной LAST. Если цепь пуста, то значение LAST равно нулю и управление будет передано из строки 17 в строку 39. В противном случае LAST есть номер одной изстрок, имеющих ненулевой элемент в столбце I, а именно «последней» строки цепи; L в строке 19 — это номер «следующей» строки, «первой» строки списка. В строке 21 переменная IUC = IUP (L) указывает ненулевой элемент строки L, стоящий в столбце I, а IUD в строке 22 — конец строки L. Коэффициент, на который нужно умножить ненулевые элементы строки L перед вычитанием, вычисляется в строке 23 и помещается в UM. Самовычитание элементов строки L, умноженных на UM, выполняет цикл DO 60.

Мы уже отмечали, что элементы каждой строки следует нормировать делением на диагональный элемент, чтобы получить

строку верхнего треугольного множителя U. В этом смысл строки 27, где UM определена в строке 23, а DI (L) в действительности — число, обратное к диагональному элементу (нужно рассмотреть строку 39 и учесть, что L < I). Важно понимать, что каждая строка I поначалу генерируется и хранится в ненормированном виде. Всякий ненулевой элемент любой строки нормируется позже, в момент, когда он используется как множитель для строки L (строки 23 и 27). При таком способе обработки экономится много умножений [Tinney, Walker, 1967, p. 1805]. Поскольку строки 23 и 27 выполняются для каждого ненулевого элемента, причем ровно один раз, то имеется гарантия правильной нормировки всех элементов.

Вспомогательный указатель IUP, отмечающий позицию ненулевого элемента строки L, стоящего в столбце I, увеличивается на.единицу (строка 28) и указывает теперь следующий ненулевой элемент строки L, если такой имеется. Если в строке L больше нет ненулевых элементов, то управление передается из строки 29 в строку 38, где возобновляется просмотр цепи. В противном случае, если J — столбцовый индекс следующего ненулевого элемента строки L, то (строки 30—37) строка L вставляется в цепь, ассоциированную со столбцом J. Заметим, что не так было в символическом алгоритме из § 7.25, где строка L использовалась только однажды, а затем не принималась в учет.

Строка 39 выполняется, если все строки, ассоциированные со столбцом I, уже обработаны. Если полученная строка I непуста, то ненулевые элементы, пока ненормированные, выбираются из расширенного накопителя DI и помещаются в UN (цикл DO 100). Затем новая строка I вставляется в цепь, ассоциированную со столбцовым индексом J своего первого ненулевого элемента. Эта операция производится в строках 43—50. Наконец, прежде чем перейти к обработке следующей строки, определяется вспомогательный указатель IUP (1) первого ненулевого элемента строки I (строка 51).

7.27. ПРИМЕР ПРЯМОГО ХОДА И ОБРАТНОЙ ПОДСТАНОВКИ

В этом параграфе будет разобран простой пример прямого хода и обратной подстановки, который послужит подготовительным материалом для алгоритма, излагаемого з § 7.28. Пусть $Ax = b$ — линейная система, которую нужно решить, $A = U^T DU$ — разложение симметричной матрицы A. Общий случай был рассмотрен в § 2.13[1]). Симметричный случай вполне

[1]) Имеется в виду решение несимметричной системы с факторизованной матрицей. — *Прим. перев.*

аналогичен. Лннейная система принимает вид $U^{\mathrm{T}}DUx = b$. Полагая $z = DUx$, $w = Ux$, можем написать

$$\begin{aligned} U^{\mathrm{T}}z &= b, \\ Dw &= z, \\ Ux &= w. \end{aligned} \tag{7.19}$$

Из систем (7.19) находим вначале z, затем w и, наконец, x. Поскольку U и U^{T} — треугольные матрицы, а D — диагональная, то решение систем не вызывает затруднений. Рассмотрим следующий пример:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ e & f & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} 1 & d & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.20}$$

Первая из систем (7.19) имеет вид

$$\begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ e & f & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}. \tag{7.21}$$

Последовательной подстановкой получаем

$$\begin{aligned} z_1 &= b_1, \\ z_2 &= b_2 - dz_1, \\ z_3 &= b_3 - ez_1 - fz_2. \end{aligned} \tag{7.22}$$

Вторая из систем (7.19):

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}. \tag{7.23}$$

Находим

$$\begin{aligned} w_1 &= z_1/a, \\ w_2 &= z_2/b, \\ w_3 &= z_3/c. \end{aligned} \tag{7.24}$$

Так как в памяти обычно хранится не сама D, а обратная к ней, то формулы (7.24) требуют только умножений, которые на большинстве машин производятся быстрее делений. Наконец, последняя система (7.19):

$$\begin{bmatrix} 1 & d & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}. \tag{7.25}$$

Окончательное решение легко вычисляется обратной подстановкой:

$$\begin{aligned} x_3 &= w_3, \\ x_2 &= w_2 - fx_3, \\ x_1 &= w_1 - dx_2 - ex_3. \end{aligned} \tag{7.26}$$

7.2.8 АЛГОРИТМ РЕШЕНИЯ СИСТЕМЫ $U^TDUx = b$

Вход: IU, JU, UN — верхняя треугольная матрица U с единичной диагональю, заданная схемой RR (U) U
DI — обратная к диагональной матрице D
B — вектор правых частей b
N — порядок системы (N > 1)
Выход: X — вектор неизвестных x.

```
1.            NM = N-1
2.            DO 10   I=l,N
3.    10      X(I)=B(I)
4             DO 40 K = l,NM
5.            IUA=IU(K)
6.            IUB = IU(K + 1)-l
7.            XX = X(K)
8.            IF(IUB.LT.IUA)GOTO 30
9.            DO 20 I=IUA,IUB
10.   20    X(JU(I))    = X(JU(I))-UN(I)*XX
11.   30      X(K)=XX*DI(K)
12.   40      CONTINUE
13.           X(N)=X(N)*DI(N)
14.           K=NM
15.   50      lUA = IU(K)
16.           IUB=IU(K + 1) - 1
17.       IF(IUB.LT.IUA)GOTO        70
18.           XX = X(K)
19.           DO 60 I=IUA,IUB
20.   60      XX = XX-UN(I)*X(JU(I))
21.           X(K) = XX
22.   70      K = K-1
23.       IF(K.GT.0)GOTO        50
```

Работу этого алгоритма легче понять, если мы будем ссылаться на пример из § 7.27. Вначале в массиве X вычисляется вектор z (см. уравнения (7.22)). Начальное состояние массиву присваивается в строках 2—3. Цикл DO 40 последовательно обрабатывает строки U. Вычисления, указываемые формулами (7.22), производятся по столбцам, поскольку в этом случае доступ будет осуществляться к строкам матрицы U. Как только закончена обработка очередной строки U, компонента X (K) умножается на число, обратное к K-му элементу D, которое хранится в DI (K) (см. строку 11). Эта операция требуется формулами (7.24). После того как выполнена строка 13, в массиве X находится вектор w.

Решение системы (7.25) начинается со строки 14. В массиве X сейчас находятся элементы вектора w. В цикле DO 60 совершаются операции, соответствующие формулам (7.26), на этот раз строка за строкой, поскольку тогда доступ к U снова будет происходить по строкам· Окончательное решение x будет получено в массиве X.

Глава 8

Инцидентность и сборка по узлам

8.1. ВВЕДЕНИЕ

В этой главе мы продолжим описание некоторых полезных алгоритмов, сконструированных специально для разреженных матриц, хранимых в строчном формате. Нас будут интересовать сеточные задачи. Типичные сеточные задачи возникают при применении *метода конечных элементов* [Zienkiewicz, 1977] к дискретизации и вычислению приближенного решения краевой задачи для уравнения с частными производными. Сеточные задачи получаются и в том случае, если краевая задача решается *методом граничных элементов* [Brebbia, 1978]. В этом методе дискретизуется только граница области. Если область представляет собой трехмерное тело, то нужно дискретизировать граничную поверхность; результатомбудет двумерная сетка, аналогичная двумерной сетке метода конечных элементов.

Конкретнее, нас будут здесь интересовать алгоритмические аспектыраспределения памятии управления данными при построении ассоциированной с сеткой разреженной *матрицы жесткости* [1]. Предположим, что исходная область аппроксимируется сеткой из m элементов, не имеющих общих внутренних точек; однако граница двух соседних элементов считается принадлежа-

[1]) В оригинале говорится о матрице узлового ансамбля (nodal assembly matrix). Мы предпочли здесь и далее заменить это название более употребительным — матрица жесткости. — *Прим. перев.*

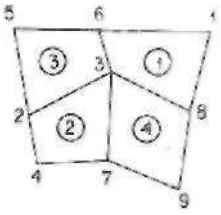


Рис. 8.1. Простая конечноэлементная сетка.

щей им обоим. В углах элементов, а также, возможно, в других точках границы или даже внутри элементов размещаются узлы. Следовательно, можно сказать, что узел, лежащий на границе двух элементов, принадлежит им обоим, а узел в углу или на ребре [1] принадлежит всем элементам, включающим в себя этот угол или это ребро. Пусть n — число узлов. Элементы нумеруются в произвольном порядке последовательными числами от 1 до m. Нумеруются и узлы числами от 1 до n. Пример двумерной сетки с пронумерованными узлами и элементами показан на рис. 8.1. *Матрица инцидентности* [2] E — это булева матрица, определяемая следующим образом: E — прямоугольная матрица с размерами $m \times n$, причем $e_{ij} = 1$, если узел j принадлежит элементу i, и $e_{ij} = 0$ в противоположном случае.

Таким образом, строка матрица E соответствует элементу, а единицы в этой строке — номерам, присвоенным узлам этого элемента. Часто используется и транспонированная матрица E^{T} с размерами $n \times m$. Строка E^{T} соответствует узлу, а единицы в этой строке — элементам, которым принадлежит данный узел. При хранении разреженным строчным форматом обе матрицы E и E^{T}, будучи булевыми, полностью определяются своими портретами. Для матрицы E будем использовать массивы IE, JE; для матрицы E^{T} — массивы IET, JET. Поскольку массив JE содержит столбцовые индексы ненулевых элементов каждой строки E, а столбцовые индексы — это в точности номера узлов элемента, отвечающего данной строке, то ясно, что JE интерпретировать как последовательность списков номеров узлов всех элементов. Обычно узлы любого элемента перечисляются в JE в некотором условленном порядке, описывающем какое-то топологическое свойство дискретизации. Например, для двумерного элемента, узлы которого находятся на границе и в углах, принято упорядочивать узлы в направлении против часовой стрелки, отправляясь от произвольного угла. Если имеются внутренние узлы, то они идут после граничных. С точки зрения разреженного строчного формата этот способ перечисления соответствует неупорядоченному представлению. Интересно, что при хранении E полным форматом это соглашение о внутриэлементном упорядочении невозможно.

---

[1]) По-видимому, автор имеет в виду ребро трехмерной сетки. — *Прим. перев.*

[2]) В оригинале — the connectivity matrix, т. е. буквально матрица связности. Мы предпочли название «матрица инцидентности» исходя из аналогии с понятием, употребляемым в теории графов. Это название и используется в дальнейшем. — *Прим. перев.*

Аналогичные замечания справедливы в отношении E^{T}. Массив JET — это последовательность списков элементов, которым принадлежит каждый узел. Здесь, однако, обычно нет необходимости придерживаться какого-либо соглашения при перечислении элементов. В контексте наших задач, это — благоприятное обстоятельство: представление IET, JET матрицы E получается путем транспонирования представления IE, JE матрицы E посредством алгоритма из § 7.3. Последний строит JET таким образом, что для каждого узла элементы упорядочиваются по возрастанию присвоенных им номеров.

Задача построения матрицы инцидентности E относится к более широкой категории задач, связанных с *генерированием сеток* (см., например, [Pissanetzky, 1981]).

Функциональное подпространство, в котором разыскиваются приближения к решению дифференциального уравнения, определяется указанием множества *функций формы; обычно* это простые функции, между которыми, с одной стороны, и узлами сетки — с другой, установлено соответствие. Описание сетки будет полным, если заданы — по отношению к какой-либо координатной системе — позиции узлов для всех элементов и все функции формы. Однако для наших целей этой информации не потребуется.

Как объясняется в § 4.14, обычные реализации метода конечных элементов предполагают, что матрица A собрана, и предстоит решить систему линейных уравнений

$$Ax = b \tag{8.1}$$

или задачу на собственные значения

$$Ax = \lambda x. \tag{8.2}$$

Для скалярных задач с каждым узлом связано одно неизвестное, поэтому порядок матрицы A равен n. A называется матрицей *узлового ансамбля,* а часто также матрицей *жесткости* — термин, происходящий из строительной механики. A вычисляется как сумма элементных матриц $A^{(e)}$ (см. уравнение (4.28)), и коэффициент α_{ij} не равен нулю, только если найдется элемент, которому принадлежат оба узла i и j; в этом случае будем говорить, что узлы i и j *связаны.* Так как всякий узел связан лишь с несколькими другими узлами сетки, то A имеет всего несколько ненулевых элементов в каждой строке; что еще более важно, число ненулевых элементов в строке не зависит от размера сетки. Следовательно, A — разреженная матрица, и общее число ее ненулевых элементов пропорционально числу узлов сетки. Портрет A симметричен, а часто и сама матрица A симметрична и положительно определена.

Элементные матрицы $A^{(e)}$ обычно строятся в цикле DO, который вычисляет матрицу каждого элемента и немедленно прибавляет или *включает* [1] ее в матрицу A. При необходимости сформировать систему (8.1) вычисляется и включается в b также и элементный узловой вектор. Конкретный способ вычисления элементных матриц и векторов зависит от дифференциального уравнения, аппроксимирующего подпространства и выбранного численного метода. Для нас этот способ не представляет интереса, так как мы будем иметь дело с деталями процесса сборки, но не с собираемыми численными значениями. Поскольку $a_{ij} \neq 0$, только если i и j — узлы элемента e, то $A^{(e)}$ обычно хранится и обрабатывается как полная подматрица порядка, равного числу узлов элемента e.

8.2. ГРАНИЧНЫЕ УСЛОВИЯ ДЛЯ СКАЛЯРНЫХ ЗАДАЧ

В ряде случаевпри сборке линейной системы (8.1) могут быть заранее предписаны значения неизвестных, ассоциированных с некоторыми узлами сетки. Так обстоит дело с узлами, попадающими на границу или части границы, где заданы граничные условия Дирихле. Эти узлы мы будем называть *узлами Дирихле.* С узлом Дирихле связано, таким образом, единственное очевидное уравнение

$$x_i = \bar{x}_i, \tag{8.3}$$

где $\bar{x}_i$ — заданное значение неизвестного x_i. Это очевидное уравнение обычно включается в систему, порождая строку матрицы A, где 1 на диагонали является единственным ненулевым элементом, и правую часть, которой служит заданное значение. Основание для сохранения узлов Дирихле в формулировке системы — то, что существуют другие узлы, связанные с узлами Дирихле, и другие уравнения, где присутствуют члены, содержащие неизвестные Дирихле. Поскольку «неизвестные» Дирихле в действительности известны, такие члены следует исключить из уравнений, вычитая их из соответствующих правых частей. Эту операцию проще выполнять, когда узлы Дирихле сохранены в формулировке. Описанная методика особенно удобна при использовании разреженных форматов благодаря легкости, с какой можно учитывать, не входя в дополнительные расходы, пустые строки A.

Распространенная практика состоит в том, чтобы собирать линейную систему (8.1), как обычно, из элементных матриц, включая уравнение i. Затем i-я строка A заполняется нулями, а на диагональ помещается 1. После этого во внедиагональные позиции столбца i записываются нули, а элементы, хранившиеся

[1]) В оригинале — assemble, — *Прим. перев.*

в этих позициях прежде, умножаются на x_i и вычитаются из b. Наконец, b_i полагается равным 1. Очевидно, что если A — симметричная матрица, то эти операции не нарушат симметрию. Если A вдобавок положительно определенная, то как легко показать, модифицированная матрица также положительно определена. Действительно, без потери общности можно рассмотреть случай $i = n$. Так как A симметрична и положительно определена, то (см. свойство 6 из § 2.2) главные миноры A порядков 1, 2, ..., n — 1 положительны. Когда в строку и столбец с номером n записываются нули, а в позицию (n, n) — единица, то ясно, что определитель полученной матрицы снова будет положителен. Согласно свойству 6 из § 2.2, выполнено необходимое и достаточное условие, обеспечивающее положительную определенность модифицированной матрицы.

Матрицу A часто хранят и обрабатывают как ленточную матрицу. Так как с ростом числа узлов в сетке ширина ленты увеличивается, то при большой сетке приходится хранить и обрабатывать большое количество нулей. Поэтому в случае систем высокого порядка лучше подходят методы для разреженных матриц. Достоинством последних является то, что на нумерацию узлов и элементов не накладывается априорных ограничений, поскольку здесь не приходится учитывать ширину ленты. У ленточных матричных процедур есть еще тот недостаток, что узлы Дирихле могут вызывать увеличение ширины ленты.

8.3. ГРАНИЧНЫЕ УСЛОВИЯ ДЛЯ ВЕКТОРНЫХ ЗАДАЧ

Метод конечных элементов часто используется для решения задач, где неизвестной функцией является векторное поле. В таких случаях каждая компонента поля рассматривается отдельно, как если бы она была скалярной функцией, и представляется на сетке посредством *своего набора узловых неизвестных. Следовательно,* с каждым узлом сетки теперь связывается несколько неизвестных. Например, в трехмерных задачах теории упругости, где моделью является упругое тело, неизвестные, ассоциированные с каждым узлом i — это три его перемещения x_{i1}, x_{i2}, x_{i3} вдоль координатных направлений. В гидродинамике модель — это жидкость, а неизвестные — компоненты вектора скорости в каждом узле. В электродинамике за неизвестные берут компоненты электрического или магнитного поля.

В векторных задачах имеются различные типы граничных условий. В некоторых узлах векторы могут быть заранее фиксированы, т. е. все компоненты должны иметь предписанные значения. Или же от вектора в некотором узле требуется, чтобы он сохранял заданное направление — в двумерной или трехмерной задаче, — либо был параллелен заданной плоскости для трех-

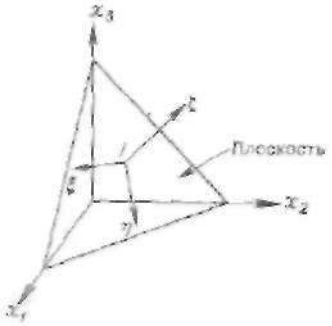


Рис. 8.2. Локальная координатная система в случае заданной плоскости.

мерной задачи. Например, для точки упругого тела перемещение может быть задано или ограничено заданной прямой или плоскостью. В случае жидкости, если часть границы области, где решается задача, является осью или плоскостью симметрии, то скорость в соответствующих точках должна принадлежать этой оси или плоскости; в других же точках границы скорость может быть предписана. Если для вектора задано значение или заданная прямая параллельна координатному направлению, или заданная плоскость перпендикулярна координатному направлению, то получаются условия типа уравнения (8.3), которые легко обрабатываются описанными выше стандартными процедурами. Мы рассмотрим теперь более трудную ситуацию, когда линия не параллельна или плоскость не перпендикулярна никакому координатному направлению.

Для определенности мы ограничимся анализом трехмерной задачи. Если вектор компонент (x_{i1}, x_{i2}, x_{i3}) узла i должен лежать в заданной плоскости, то

$$v_1x_{i1} + v_2x_{i2} + v_3x_{i3} = 0, \tag{8.4}$$

где (v_1, v_2, v_3) — нормальный вектор этой плоскости. Если вектор неизвестных (x_{i1}, x_{i2}, x_{i3}) должен быть параллелен прямой, то

$$v_1x_{i1} + v_2x_{i2} + v_3x_{i3} = 0,$$
$$w_1x_{i1} + w_2x_{i2} + w_3x_{i3} = 0. \tag{8.5}$$

где (v_1, v_2, v_3) и (w_1, w_2, w_3) — два заданных неколлинеарных вектора, перпендикулярных этой прямой. Уравнения типа (8.4) или (8.5) можно присоединить к системе линейных уравнений, как мы сделали это с уравнением (8.3). Однако если A симметрична, то симметрия будет утрачена. Мы опишем сейчас другую процедуру, основанную на замене переменных и сохраняющую как симметрию, так и положительную определенность A.

Возьмем для узла i локальную координатную систему ξ, η, ζ, где в случае заданной плоскости ось ζ проводится перпендикулярно этой плоскости (что показано на рис. 8.2), а в случае заданной прямой ось ξ параллельна этой прямой (см. рис. 8.3). Вектор (x_{i1}, x_{i2}, x_{i3}) узла i можно представить следующим образом:

$$\begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \end{bmatrix} = \begin{bmatrix} \lambda_1 & \mu_1 & \nu_1 \\ \lambda_2 & \mu_2 & \nu_2 \\ \lambda_3 & \mu_3 & \nu_3 \end{bmatrix} \begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix}, \tag{8.6}$$

Здесь $(\lambda_1, \lambda_2, \lambda_3)$, (μ_1, μ_2, μ_3), (ν_1, ν_2, ν_3) — направляющие косинусы соответственно осей ξ, η и ζ, а (ξ, η, ζ) — компоненты вектора в локальной системе. Матрица в уравнении (8.6) — ортогональная. Вектор параллелен заданной плоскости, если $\zeta = 0$, и параллелен заданной прямой, если $\eta = \zeta = 0$. Таким образом, если в качестве координат в узле i взять ξ, η, ζ, то нужные условия принимают очень простой вид. Вместо обычного вектора x узловых неизвестных:

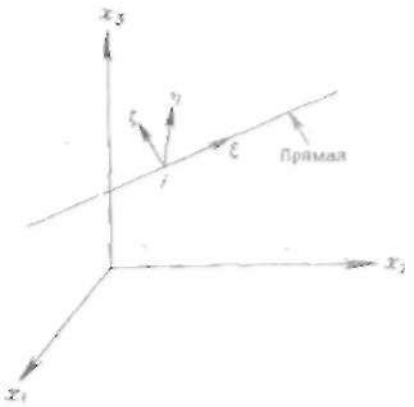


Рис. 8.3. Локальная координатная система в случае заданной прямой.

$$x = (x_{11}, x_{12}, x_{13}, \ldots, x_{i1}, x_{i2}, x_{i3}, \ldots, x_{n1}, x_{n2}, x_{n3})^{\mathrm{T}} \quad (8.7)$$

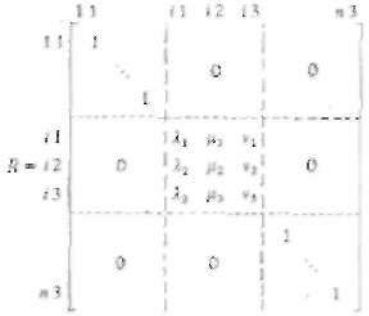
используется следующий вектор x':

$$x' = (x_{11}, x_{12}, x_{13}, \ldots, \xi, \eta, \zeta, \ldots, x_{n1}, x_{n2}, x_{n3})^{\mathrm{T}}. \quad (8.8)$$

Из уравнения (8.6) имеем

$$x = Rx', \quad (8.9)$$

где:



Матрица R также ортогональная, поскольку $RR^{\mathrm{T}} = I$, или $R^{\mathrm{T}} = R^{-1}$. В общем случае может присутствовать группа узлов, где заданы плоскости, и другая группа, где заданы прямые. У матрицы R тогда будет группа блоков с размерами 3x3 в различных местах главной диагонали; остальные диагональные позиции заняты единицами. Вектор x' будет содержать группу троек ло-

кальных координат. R по-прежнему будет ортогональна. Система линейных уравнений

$$Ax=b \tag{8.11}$$

собирается из элементных матриц обычным способом. Умножая уравнение (8.11) слева на R^{T} и используя уравнение(8.9), получим

$$R^{\mathrm{T}}ARx' = R^{\mathrm{T}}b, \tag{8.12}$$

или, полагая $B = R^{T}AR$, $b' = R^{\mathrm{T}}b$,

$$Bx' = b'. \tag{8.13}$$

В уравнении (8.13) матрица B симметричная и положительно определенная (свойство 8 из § 2.2). Условия $\zeta = 0$ или $\eta = \zeta = 0$ вводятся в систему прежним образом: во все внедиагоналыше позиции соответствующих строк и столбцов записываются нули, в диагональные позиции — единицы, в одноименные позиции b' — нули. Как показано выше, это сохраняет и симметрию, и положительную определенность B. После этого решается система (8.13) и вычисляется вектор x'. Наконец, с помощью уравнения (8.9) определяется решение x относительно глобальной системы координат.

Легко видеть, что вследствие специального вида условий $\zeta = 0$ или $\eta = \zeta = 0$ мы получим тот же результат при меньших вычислительных затратах, если положим

$$R = \begin{array}{c|ccccc} & 1\,1 & i\,1 & i\,2 & i\,3 & n\,3 \\ \hline 1\,1 & 1 & & 0 & & 0 \\ i\,1 & & \lambda_1 & 0 & 0 & \\ i\,2 & 0 & \lambda_2 & 0 & 0 & 0 \\ i\,3 & & \lambda_3 & 0 & 0 & \\ & & & & 1 & \\ n\,3 & 0 & & 0 & & 1 \end{array} \tag{8.14}$$

для плоскости, заданной в узле i, и

$$R = \begin{array}{c} \\ 11 \\ \\ i1 \\ i2 \\ i3 \\ \\ n3 \end{array} \begin{array}{c} \begin{array}{cccc} 11 & & i1\ i2\ i3 & & n3 \end{array} \\ \left[\begin{array}{ccc|ccc|c} 1 & & & & & & \\ & \ddots & & & 0 & & 0 \\ & & 1 & & & & \\ \hline & & & \lambda_1 & \mu_1 & 0 & \\ & 0 & & \lambda_2 & \mu_2 & 0 & 0 \\ & & & \lambda_3 & \mu_3 & 0 & \\ \hline & & & & & 1 & \\ & 0 & & & 0 & & \ddots \\ & & & & & & 1 \end{array}\right] \end{array} \tag{8.15}$$

если в узле i задана прямая. Получится матрица B, у которой соответствующие строки и столбцы уже нулевые; остается поместить единицу в одноименные диагональные позиции и нуль — в одноименные позиции b' [1]). B по-прежнему симметрична и положительно определена, и уравнение (8.9) сохраняет силу для упрощенной матрицы R. Поскольку блоки 3×3 в матрице A обычно заполнены, то эта процедура не только сохраняет разреженность но и может *увеличивать* ее. По существу описанный прием аналогичен хорошо известному методу экономизации для спектральных задач [Irons, 1963]. Другим способом обработки векторных граничных условий, также использующим преобразование координат, является *конденсация* (см., например, [Gallagher, 1975]).

8.4. ПРИМЕР МАТРИЦЫ ИНЦИДЕНТНОСТИ

Рассмотрим двумерную сетку, показанную на рис. 8.1. В ней N = 9 узлов и M = 4 элементов, пронумерованных произвольным образом. Номера элементов обведены кружками. Матрица инцидентности E для этой сетки такова:

$$E = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left[\begin{array}{ccccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array}.$$

Каждая строка этой матрицы ассоциируется с элементом, каждый столбец — с узлом. Строка 1 имеет ненулевые элементы в позициях 1, 3. 6 и 8, чем выражается тот факт.что узлы 1, 3, 6 и 8

[1]) Впрочем, согласно определению $b' = R^{\mathrm{T}}b$, нули в нужных позициях b' получаются автоматически. — *Прим. перев.*

принадлежат элементу 1. Ненулевые элементы столбца 7 находятся в позициях 2 и 4, потому что узел 7 принадлежит и элементу 2, и элементу 4. В этой полной записи номера узлов для каждого элемента перечисляются в порядке возрастания. Если же E представлена разреженным строчным форматом, то эти номера могут задаваться в условленном порядке, принятом для элементов. Например, при обычном для двумерных элементов соглашении об упорядочении в направлении против часовой стрелки мы могли бы получить такую форму RR (C) U:

```
IE = 1  5  9  13  17
JE = 3  8  1  6,  7  3  2  4,  5  2  3  6,  7  9  8  3.
```

Приведен только портрет E, так как все ненулевые элементы равны единице. Согласно определению разреженного формата, это представление неупорядоченное; однако в известном смысле оно все же упорядочено: ведь для каждого элемента узлы указаны в условленной последовательности; произвольным остается только выбор ее первого узла.

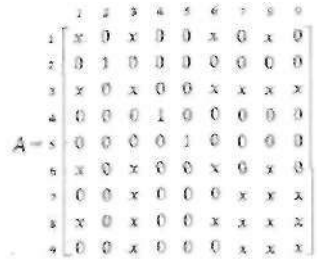
Транспонированную матрицу E^{T} также можно представить в разреженном формате, например посредством схемы RR (C) O:

```
IET = 1  2  4   8  9  10    12   14  16  17
JET = 1, 2  3,  1  2  3   4,  2,  3,  1   3,  2  4,  1  4,  4
```

Это представление легко получить из массивов IE, JE с помощью алгоритма транспонирования из § 7.3. В JET для каждого узла перечисляются номера ассоциированных с ним элементов; например, 2, 3 в строке 2 — это указание, что узел 2 принадлежит только элементам 2 и 3. В некоторых вычислениях одновременно используются оба представления: и IE, JE, и IET, JET.

8.5. ПРИМЕР МАТРИЦЫ ЖЕСТКОСТИ

Для сетки на рис. 8.1 матрица жесткости имеет следующий вид:



Здесь показан лишь портрет, поскольку численные значения можноопределить только путем вычисления и сборки матриц всех

элементов и они зависят от решаемой задачи. Во многих практических случаях матрица A симметрична, а ее портрет симметричен всегда. Предполагая, что в данном примере A симметрична, мы можем представить ее портрет посредством схемы *RR* (U) O:

IA = 1 4 9 15 16 17 18 20 21 21

JA = 3 6 8, 3 4 5 6 7, 4 5 6 7 8 9, 7, 6, 8, 8 9, 9

Такое представление (хотя и неупорядоченное) можно получить из массивов IE, JE, IET, JET, пользуясь алгоритмом из §8.6. Заметим, что внедиагональных ненулевых элементов только 20. Если бы A рассматривалась как ленточная матрица, то минимальная полуширина ленты в этом примере была равна 4, и число внедиагональных элементов [1]) составило 26. Это показывает преимущество разреженного представления даже для такой маленькой сетки.

Как указано в § 8.2, имеет смысл сохранять в системе уравнения, ассоциированные с узлами Дирихле; им будут соответствовать строки *A,* где единственный ненулевой элемент равен 1 и стоит на диагонали. Там же, в § 8.2, объяснялось, что члены, отвечающие связям других узлов с узлами Дирихле, нужно удалить из соответствующих уравнений, вычитая эти члены из правых частей. Предположим, что на рис. 8.1 узлами Дирихле будут узлы 2, 4 и 5. Для этого случая матрица жесткости принимает вид

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{bmatrix} x & 0 & x & 0 & 0 & x & 0 & x & 0 \\ 0 & x & x & x & x & x & x & 0 & 0 \\ x & x & x & x & x & x & x & x & x \\ 0 & x & x & x & 0 & 0 & x & 0 & 0 \\ 0 & x & x & 0 & x & x & 0 & 0 & 0 \\ x & x & x & 0 & x & x & 0 & x & 0 \\ 0 & x & x & x & 0 & 0 & x & x & x \\ x & 0 & x & 0 & 0 & x & x & x & x \\ 0 & 0 & x & 0 & 0 & 0 & x & x & x \end{bmatrix} \end{array}$$

В диагональных позициях 2, 4 и 5 теперь стоят единицы. Новая форма RR (U) O:

IA = 1 4 4 8 8 8 9 11 12 12

JA = 3 6 8, 6 7 8 9, 8, 8 9, 9

Здесь только 11 ненулевых внедиагональных элементов; в то же время ленточное представление по-прежнему имело бы 26 внедиагональных элементов.

[1]) Подлежащих хранению. — *Прим. перев.*

8.6. АЛГОРИТМ СИМВОЛИЧЕСКОЙ СБОРКИ СИММЕТРИЧНОЙ МАТРИЦЫ ЖЕСТКОСТИ

Вход: IE, JE матрица инцидентности сетки в форме RR (C) U
IET, JET транспонированная матрица E^{T} в форме RR (C) U
N число узлов сетки
IA массив длины N, содержащий N в позициях, соответствующих узлам Дирихле, и 0 в прочих позициях
Выход: IA, JA портрет матрицы жесткости в форме RR (U) U. Матрица имеет порядок N.

```
1.            JP=1
2             NM=N-1
3.            DO 30 I = 1,NM
4.            JPI=JP
5.            lF(IA(I).EQ.N)GOTO 30
6.            IETA = IET(I)
7-            IETB=IET(I + 1)-1
8.            DO 20 IP = IETA,IETB
9             J = IET(IP)
10.  IEA =    IE(J)
11.           IEB = IE(J + 1)-1
12.           DO 10 KP = IEA,IEB
13.           K=JE(KP)
14.   IF(K,    LE,I)GOTO    10
15.           IF(IA(K).GE.I)GOTO 10
16.           JA(JP)=K
17.           JP = JP+1
18.           IA(K)=I
19    10      CONTINUE
20.   20      CONTINUE
21    30 IA(I) = JPI
22.   IA(N)   =   JP
23.           IA(N+1) = JP
```

Этот алгоритм по существу тождествен символическому умножению (см. § 7.21), где сомножителями являются E^{T} и E, а результатом — портрет A [Gustavson, 1976c]. Массив IA играет три разных роли: на входе он указывает узлы Дирихле (если таковые имеются); в процессе работы алгоритма IA рассматривается как массив переменных переключателей (см. § 1.4); на выходе это массив указателей для JA. При обработке строки I в позициях 1, ... I — 1 массива IA хранятся указатели первых I — 1 строк A, а позиции I, ..., N используются как переключатели. Это смешанное использование не требует никаких дополнительных усилий, потому что вследствие выбора для матрицы верхнего представления для строки I интерес представляют лишь столбцовые индексы, большие чем I.

Переменная JP является указателем для списка JA; начальное значение она получает в строке 1, а увеличивается — в строке 17.

[1]) В верхнем представлении. — ***Прим. перев.***

В цикле DO 30 обрабатываются только N — I строк; так как в строке N нет элементов справа от диагонали, то она обязательно пустая [1]). Строка 5 рассчитана на уравнения, ассоциированные с узлами Дирихле: соответствующие, строки оставляются пустыми. Цикл DO 20 (строка 8) идет ло элементам, которым принадлежит узел I; заметим, что IETB ≥ IETA, поскольку каждый узел принадлежит хотя бы одному элементу. В строке 9 J — это номер одного из элементов, содержащих узел I. Цикл DO 10 (строка 12) идет по узлам элемента J; заметим, что обязательно выполняется неравенство IEB > IEA, поскольку каждый элемент имеет хотя бы два узла. В строке 13 К — это номер одного из узлов элемента J. В строке 14 отбрасываются все узлы, не находящиеся правее диагонали. В строке 15 проверяется переключатель; обратим внимание, однако, что IA (К) = I означало бы, что узел К уже включен в список JA и не должен вноситься туда вторично, в то время как IA (К) = N свидетельствует, что К — узел Дирихле, и его вообще не следует вставлять в JA. Остальные узлы записываются в JA (строка 16), и вслед за этим блокируется переключатель (строка 18). Переменная JP в строках 21, 22 и 23 используется для вычисления значений строчных указателей, помещаемых в IA.

8.7. АЛГОРИТМ ЧИСЛЕННОГО ВКЛЮЧЕНИЯ ЭЛЕМЕНТНОЙ МАТРИЦЫ ЖЕСТКОСТИ И ЭЛЕМЕНТНОГО УЗЛОВОГО ВЕКТОРА В ГЛОБАЛЬНУЮ МАТРИЦУ ЖЕСТКОСТИ *A* И ВЕКТОР ПРАВЫХ ЧАСТЕЙ *B* СИММЕТРИЧНЫЙ СЛУЧАЙ

Вход:	IA, JA	портрет матрицы А в форме RR (U) U. Порядок матрицы А равен N
	IDIR	массив, указывающий узлы Дирихле; он содержит 1 для узла Дирихле и 0 для любого другого узла
	AE, BE	элементная матрица жесткости и элементный узловой вектор, включаемые в массивы AN, AD и B
	JEP	глобальные номера узлов, ассоциированных со строками и столбцами AE
	NN	порядок матрицы AE; размерность вектора BE
Выход:	AN, AD	численные значения ненулевых элементов матрицы А в форме RR (U) U
	B	вектор правых частей системы линейных уравнений
Рабочее пространство:	IP	массив длины N, играющий роль расширенного целого масснва указателей. Начальное состоянне — нулевое. После использования массива оно восстанавливается
Замечание:		заданные значення неизвестных Дирихле должны быть записаны в соответствующие позиции В. Другими словами, если i — узел Дирихле с граничным условием C_i, то до использования данного алгоритма нужно положить IDIR (i) = 1, B (i) = C_i

```
 1.              DO 40 L=1,NN
 2.              I=JEP(L)
 3.              IF(IDIR(I).NE.0)GOTO 40
 4.              K=L-NN
 5.              AD(I) = AD(I) + AE(K + L*NN)
 6.    B(I)  =   B(I)   +   BE(L)
 7.              KK=0
 8.              DO 20 LL = 1,NN
 9.              K = K + NN
10.              IF(LL.EQ.L)GOTO 20
11               J = JEP(LL)
12.              IF(IDIR(J).NE.0)GOTO 10
13.              IF(J.LT.I)GOTO 20
14.     IP(J)    =      K
15.              KK=I
16.              GOTO20
17      10       B(I) = B(I)-B(J)*AE(K)
18      20       CONTINUE
19.              IF(KK.EQ.0)GOTO 40
20.              IAA = IA(I)
21.              IAB = IA(I + 1)-1
22.              DO 30 J = IAA,IAB
23.              K = IP(JA(J))
24.              IF(K.EQ.0)GOTO 30
25.              AN(J) = AN(J) + AE(K)
26.              IP(JA(J)) = 0
27.     30       CONTINUE
28.     40       CONTINUE
```

Этот алгоритм включает элементную матрицу AE порядка NN в соответствующие позиции глобальной матрицы жесткости порядка N; узловой вектор BE размерности NN включается в соответствующие позиции вектора правых частей B размерности N. Уравнения, ассоциированные с узлами Дирихле, в сборке не участвуют (строка 3); члены, соответствующие связям прочих узлов с узлами Дирихле, вычитаются из B непосредственно (строка 17). По существу данный алгоритм — это алгоритм сложения, и здесь применяется техника, обсуждавшаяся в § 7.8. Расширенный целый массив IP используется для хранения указателей на элементы AE, участвующие в сборке. Поскольку матрица AE хранится по столбцам [1]), то ее элемент (L, LL) находится в позиции K = L + + (LL — 1) * NN [2]). Начальное значение параметру *K,* присваивается в строке 4, внутри цикла DO 40, который по очереди обрабатывает строки AE. В строке 2 I — это глобальный строчный индекс, ассоциированный со строкой L матрицы AE. Диагональный элемент этой строки пересчитывается в строке 5 программы, а правая часть — в строке 6.

Цикл DO 20 просматривает остальные элементы строки L матрицы AE; в строке 10 отбрасывается диагональный элемент. В строке 9 значение K переопределяется таким образом, чтобы оно стало равно L + (LL — 1) * NN. Глобальный столбцовый

[1]) Элементная матрица жесткости хранится линейным массивом AE длины NN^2. — *Прим. перев.*

[2]) Одноименного массива. — *Прим. перев.*

индекс J вычисляется в строке 11. В строке 13 отбрасываются элементы, находящиеся в нижнем треугольнике *A*. К моменту, когда программа достигает строки 14, уже принято решение, что элемент (L, LL) матрицы AE, который находится в позиции K массива AE, должен быть добавлен к элементу (I, J) глобальной матрицы *A*. Этот факт регистрируется записью K в IP (J).

Переменная KK служит для выявления ситуаций, когда ни один элемент строки L матрицы AE не участвует в сборке. В симметричном случае такие ситуации возникают хотя бы раз для каждой элементной матрицы: в сборке не участвует строка I, для которой наибольший столбцовый индекс равен I. Начальное значениеKK получает в строке 7, а переопределяется в строке 15,если в строке I найден элемент, стоящий правее диагонали. Если затем в строке 19 выясняется, что KK = 0, то цикл DO 30 не выполняется.

В цикле DO 30 просматривается строка I матрицы *A*. В строке 23 из IP выбираются значения K, а в строке 25 соответствующий элемент из AE добавляется к нужной позиции массива AN. Строка 26 восстанавливает нули в IP.

Заметим, что элементы AE не могут добавляться к элементам AN непосредственно, так как списки столбцовых индексов JEP и JA не упорядочены. Алгоритм просматривает вначале JEP, устанавливая указатели в IP, и только потом JA, пользуясь указателями, чтобы найти требуемые элементы в AE. Этим путем для каждого элемента *A* производится фиксированное число операций, а следовательно, общее число операций будет линейно зависеть от общего числа ненулевых элементов.

Нужно отметить еще, что работа операторов в строках 23 и 24 не вполне производительна. Приведем значения величин NN — 1 (числа ненулевых внедиагональных элементов в строке матрицы AE) и N2 (максимального числа ненулевых элементов справа от диагонали в строке матрицы *A*) для некоторых типичных конечно-элементных сеток:

(a) Плоская сетка, составленная из треугольников: NN — 1 = = 2, NZ = 8.

(b) Плоская сетка, составленная из четырехугольников: NN — 1 = 3, NZ = 8.

(c) Трехмерная сетка, составленная из шестигранников: NN — — 1 = 7, NZ = 26.

В последнем случае, например, строки 23 и 24 будут выполняться 26 раз, но только 7 элементов будут пересчитаны в строке 25. Из этого положения, по-видимому, нет экономичного выхода. В качестве альтернативы можно было бы рассмотреть использование вместо IP расширенного вещественного массива X в соответствии с § 1.15. Мы просматриваем JA, загружаем ненулевые элементы из AN в соответствующие позиции X, затем добавляем к X ненулевые элементы строки L матрицы AE и, наконец, снова

просматриваем JA, загружая соответствующие элементы обратно в AN. Некоторые элементы при этом будут без пользы засылаться сначала в X, а потом опять в AN, но эта впустую затраченная работа может быть сравнима с ненужным выполнением операторов 23 и 24 такое же число раз.

8.8. АЛГОРИТМ ЧИСЛЕННОГО ВКЛЮЧЕНИЯ ЭЛЕМЕНТНОЙ МАТРИЦЫ ЖЕСТКОСТИ И ЭЛЕМЕНТНОГО УЗЛОВОГО ВЕКТОРА В ГЛОБАЛЬНУЮ МАТРИЦУ ЖЕСТКОСТИ *A* И ВЕКТОР ПРАВЫХ ЧАСТЕЙ *b*. ОБЩИЙ СЛУЧАЙ

Вход:	IA, JA	портрет матрицы *A* в форме RR (C) U. Порядок матрицы *A* равен N
	IDIR	массив, указывающий узлы Дирихле; он содержит 1 для узла Дирихле и 0 для любого другого узла
	AE, BE	элементная матрица жесткости и элементный узловой вектор, включаемые соответственно в массивы AN и B
	JEP	глобальные номера узлов, ассоциированных со строками и столбцами AE
	NN	порядок матрицы AE; размерность вектора BE
Выход:	AN	численные значения ненулевых элементов матрицы *A* в форме RR (C) U
	B	вектор правых частей системы линейных уравнений
Рабочее пространство:	IP	массив длины N, играющий роль расширенного целого массива указателей. Начальное состояние — нулевое. После использования массива оно восстанавливается
Замечание:		заданные значения неизвестных Дирихле должны быть записаны в соответствующие позиции B. Другими словами, если i — узел Дирихле с заданным значением C_i, тодо использования данного алгоритма нужно положить IDIR (i) = 1, B (i) = C_i

```
1              DO 40 L=l,NW
2.             I=JEP(L)
3              IK(IDIR(J).NE.0)GOTO 40
4.             K=L-NN
5.             B(I)=B(l)+BE(L)
Ь.             DO 20 LL=l,NN
7.             K = K + NN
S.             J = JEP(LL)
9.             IF(IDIR(J).NE.0)GOTO 10
10.            IP(J)=K
11.        GOTO        20
12     10      B(l)=B(I)-B(J)*AE(K)
15.    20      CONTINUE
14.            IAA = IA(I)
15.            IAB = IA(I + 1)-1
16             DO 30 J = IAA,IAB
17.            K = lP(JA(J))
IX.        IF(K.EQ.0)GOTO        30
19.            AN(J) = AN(J) + AE(K)
20             IP(JA(J))=0
21.    30      CONTINUE
22.    40      CONTINUE
```

Этот алгоритм по существу тождествен алгоритму из § 8.7. Все же некоторые важные различия заслуживают упоминания. Алгоритм данного параграфа предназначен для несимметричной матрицы *A,* хранимой полным представлением; поэтому массив AD отсутствует. Теперь к AN нужно добавлятьвсеэлементы каждой строки [1]) матрицы AE, исключая те элементы, которые соответствуют неизвестным Дирихле. Последние включаются в вектор правых частей B строкой 12. Так как никакой конечный элемент не может содержать только узлы Дирихле, то хотя бы один элемент каждой строки [1]) AE должен добавляться к AN. Цикл DO 30 должен выполняться во всех случаях [1]), следовательно, введение параметра KK. (строки 7, 15 и 19 в алгоритме из § 8.7) в данном случае излишне.

[1]) Кроме строк, ассоциированных с узлами Дирихле (см. строку 3 программы). — *Прим. перев.*

Глава 9

Алгоритмы общего назначения

9.1. ВВЕДЕНИЕ

В этой главе мы собрали отчасти алгоритмы для различных задач, связанных с технологией разреженных матриц, отчасти свои соображения по поводу составления подобных алгоритмов. Это собрание не организовано вокруг какой-нибудь конкретной тематики; скорее оно подсказано нашим опытом работы с программами, ориентированными на разреженные матрицы.

Разреженные треугольные матрицы уже рассматривались в гл. 7 в связи с решением систем линейных уравнений. Однако треугольные матрицы заслуживают специального изучения вследствие интересных свойств, которые они имеют. Например, если задана нижняя треугольная матрица [1]), то можно вычислить про-

[1]) Нет смысла оговаривать, что заданная треугольная матрица — нижняя, потому что все сказанное дальше справедливо и для верхних треугольных матриц. — *Прим. перев.*

изведение ее обратной на вектор или другую матрицу, не вычисляя в явном виде эту обратную матрицу. § 9.2—9.6 посвящены поэтому обсуждению алгоритмов, выполняющих наиболее употребительные операции с разреженными треугольными матрицами.

Итерационное решение линейных систем является темой многих добротных публикаций. Наиболее популярны методы, использующие векторы Ланцоша. Хороший общий обзор итерационных методов дан Аксельсоном [Axelsson, 1977]. Другой, также очень хороший и более свежий обзор принадлежит Джекобсу [Jacobs, 19811. Здесь мы опишем простой алгоритм Гаусса—Зейделя; обсуждение его — в § 9.7, а программа — в § 9.8.

В § 9.9 изложены некоторые соображения относительно составления алгоритма, помогающего находить ошибки в новой программе и проверяющего входные данные или промежуточные результаты.

Специалисты-аналитики часто испытывают затруднения при работе с разреженными матрицами: ведь разреженные схемы хранения сконструированы для машин, а не для людей. Очень нелегко составить себе представление о матрице, хранимой разреженным форматом. В § 9.10 *мы* обсуждаем один возможный способ преодоления этой трудности.

Наконец, в § 9.11—9.13 мы представим три очень простых, но полезных алгоритма. Читателю, желающему работать с разреженными матрицами, мы рекомендуем приспособить алгоритмы данной главы к своим целям.

9.2. УМНОЖЕНИЕ ОБРАТНОЙ ДЛЯ НИЖНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ НА МАТРИЦЫ ОБЩЕГО ВИДА

В этом параграфе мы рассмотрим вычисление произведений типа

$$X = U^{-T}B, \tag{9.1}$$

где B — разреженная матрица общего вида, U — верхняя треугольная разреженная матрица с единичной диагональю, U^T — транспонированная к ней, $U^{-T} \equiv (U^T)^{-1}$. Задача сводится к прямому ходу (см. § 7.27—7.28, где B и X — векторы-столбцы).

U^T — разреженная нижняя треугольная матрица с единичной диагональю. Матрица X может быть найдена без вычисления обратной для матрицы U^T (которая была бы плотной матрицей). Чтобы увидеть это, перепишем (9.1) следующим образом:

$$U^T X = B. \tag{9.2}$$

Получена система линейных уравнений, неизвестными которой являются элементы X. Система (9.2) решается обычным способом. Имеются символический этап и численный этап. В них исполь-

зуются метод переменного переключателя и расширенный вещественный накопитель (см. § 1.14 и 1.15). Алгоритмы приведены в § 9.3—9.4, а сейчас мы рассмотрим простой иллюстративный пример.

Пусть дана система

$$\begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \tag{9.3}$$

Для первой строки имеем

$$(x_{11}x_{12}) = (b_{11}b_{12}). \tag{9.4}$$

что означает

$$x_{11} = b_{11},\ x_{12} = b_{12}.$$

Для второй строки:

$$a\ (x_{11}x_{12}) + (x_{21}x_{22}) = (b_{21}b_{22}).$$

Следовательно,

$$(x_{21}x_{22}) = (b_{21}b_{22}) - a\ (x_{11}x_{12}). \tag{9.5}$$

Аналогично выводим для третьей строки:

$$(x_{31}x_{32}) = (b_{31}b_{32}) - b\ (x_{11}x_{12}) - c\ (x_{21}x_{22}). \tag{9.6}$$

Уравнения (9.4), (9.5) и (9.6) разрешаются последовательно, и в результате получается матрица X. Этот алгоритм приспособлен для разреженных матриц, заданных в строчном формате, поскольку доступ к LT и B осуществляется только по строкам.

9.3. АЛГОРИТМ СИМВОЛИЧЕСКОГО УМНОЖЕНИЯ ОБРАТНОЙ ДЛЯ НИЖНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ U^T НА МАТРИЦУ B ОБЩЕГО ВИДА

Этот алгоритм вычисляет портрет матрицы $X = U^{-T}B$.

Вход:	IUT, JUT	портрет U^T в форме RR (L) U, где L означает, что представлен только нижний треугольник U^T
	IB, JB	портрет матрицы B в форме RR (C) U
	N	порядок U и число строк B и X
	NC	число столбцов B и X
Выход:	IX, JX	портрет матрицы X в форме RR (C) U
Рабочее пространство:	IP	массив переменных переключателей длины NC

```
 1                DO 10 I=1,NC
 2     10         IP(I)=0
 3.               JP=1
 4.               IX(I)=l
 5.   DO    70    I    =    l,N
 6                IBA = IB(I)
 7.   IBB   =     IB(I    +    l)-1
 S        IF(IBB.LT.IBA)GOTO          30
 9.               DO 20 JJ = lBA,IBB
10.               J=JB(JJ)
II     IP(J)      =        1
12.    JX(JP)       =       J
13     20         JP=JP+1
14.    30         IUA=IUT(I)
15        IUB           = IUT(I+1)-1
16                IF(IUB.LTIUA)GOTO 60
17.               DO 50 JJ = IVA,IUB
18                J = JUT(JJ)
19.               IXA = IX(J)
20                IXB =  IX(J  +1)-1
21.       IF(IXB.LT.IXA)GOTO          50
22.               DO 40 KP = IXA,lXB
23.               K=JX(KP)
24.               IF(IP(Kl.EQ.I)GOTO 40
25                 IP(K)=I
26.               JX(JP)=K
27.               JP = JP+1
28.    40         CONTINUE
29.    50         CONTINUE
30.    60         IX(I + 1) = JP
31.    70         CONTINUE
```

В строках 1—2 этого алгоритма придается нулевое начальное состояние массиву переменных переключателей IP. Указатель JP для массива JX получает начальное значение в строке 3 и затем увеличивается всякий раз, как в список JX добавляется новый элемент (строки 13 и 27). Этот же указатель используется для заполнения массива IX (строка 30). В цикле DO 70 по очереди обрабатываются N матричных строк. Цикл DO 20 загружает в JX строку I матрицы B (строка 12); при этом в позицию J массива переключателей IP засылается I, чтобы воспрепятствовать повторной записи столбцового индекса J в описание строки I в массиве JX.

Теперь начинается алгоритм, соответствующий уравнениям (9.4), (9.5) и (9.6) предыдущего параграфа. 3 цикле DO 50 просматривается строка I матрицы U^{π}; для каждого найденного столбцового индекса J цикл DO 40 просматривает строку J матрицы X и приписывает к JX любой столбцовый индекс К, не записанный туда ранее. Заметим, что позиция К. массива переключателей проверяется в строке 24 и в случае необходимости блокируется в строке 25.

9.4. АЛГОРИТМ ЧИСЛЕННОГО УМНОЖЕНИЯ ОБРАТНОЙ ДЛЯ НИЖНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ U^T НА МАТРИЦУ B ОБЩЕГО ВИДА

Вход:	IUT, JUT, UNT	заданная треугольная матрица U^T в форме RR (L) U
	IB, JB, BN	заданная матрица B общего вида в форме RR (C) U
	IX, JX	портрет матрицы X в форме RR (C) U
	N	порядок U и число строк B и X
Выход:	XN	массив численных значений ненулевых элементов матрицы X в форме RR (C) U
Рабочее пространство:	X	расширенный накопитель длины, равной числу столбцов B и X

```
1.              DO 80 I=1,N
2.              IH = I+1
3.              IXA = IX(I)
4.              IXB = IX(IH)-1
5.              IF(IXB.LT.IXA)GOTO 80
6.              DO 10 IP = IXA,IXB
7.     10       X(JX(IP)) = 0.
8.              IBA = IB(I)
9.              IBB=IB(IH)-1
10.             IF(IBB.LT.IBA)GOTO30
11.             DO 20 IP = IBA,IBB
12.    20       X(JB(IP)) = BN(IP)
13.    30       IUA = IUT(I)
14.       IUB        = IUT(IH)-1
15.       IF(IUB.LT.IUA)GOTO          60
16.             DO 50 JP = IUA,IUB
17.             J=JUT(JP)
18.             IXC=IX(J)
19.    IXD   =    IX(J    +    1)-1
20.             IF(IXD.LT. IXC)GO TO 50
21.             A = UNT(JP)
22              DO 40 KP = IXC,IXD
23.             K=JX(KP)
24     40       X(K) = X(K)-A*XN(KP)
25.    50       CONTINUE
26.    60       CONTINUE
27.             DO 70 IP = IXA,IXB
28.    70       XN(IP) = X(JX(IP))
29     80       CONTINUE
```

Для заданных нижней треугольной N×N матрицы U^T с единичной диагональю и матрицы B общего вида, имеющей N строк, приведенный алгоритм вычисляет в массиве XN значения ненулевых элементов матрицы $X = U^{-T}B$. В цикле DO 80 по очереди обрабатываются матричные строки. Для каждого I засылаются нули в позиции расширенного накопителя X, отвечающие ненулевым элементам строки I матрицы X (строки 6—7 программы). Ненулевые элементы строки I матрицы B выбираются из массива BN, где они хранятся компактно, и помещаются в распакованном виде в массив X; это делает цикл DO 20. В цикле DO 50 просматриваются ненулевые элементы строки I матрицы U^T и выполняются операции, иллюстрируемые уравнениями (9.4)—(9.6) из § 9.2. Для каждого ненулевого элемента из строки I матрицы U^T, столб-

цовый индекс которого равен J, а численное значение *A,* в цикле DO 40 просматривается строка J матрицы *X* (к этому моменту уже полностью вычисленная, так как J < I); ее ненулевые элементы умножаются на A и вычитаются из соответствующих позиций X. Наконец, в цикле DO 70 окончательные значения выбираются из X и помещаются в XN; при выборке учитываются столбцовые индексы элементов строки I матрицы *X,* находящиеся в JX.

Вместо накопителя X в этом алгоритме может'йспользоваться методика § 1.16.

9.5. АЛГОРИТМ УМНОЖЕНИЯ ОБРАТНОЙ ДЛЯ ВЕРХНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ *U* С ЕДИНИЧНОЙ ДИАГОНАЛЬЮ НА ЗАПОЛНЕННЫЙ ВЕКТОР *x*

Этот алгоритм вычисляет вектор $w = U^{-1}x$.

Вход: IU, JU, UN матрица U в форме RR (U) U
X заданный вектор
N порядок матрицы U, N > 1
Выход: W вектор-произведение

```
 1.          DO 10 I = 1,N
 2.   10     W(I) = X(I)
 3.          I = N - 1
 4.   20   IUA = IU(I)
 5.      IUB=IU(I      + 1 ) - 1
 6.          IF(IUB. LT.IUA)G0 TO 40
 7.          Z = W(I)
 8.          DO 30 IP = IUA,IUB
 9.   30     Z = Z-UN(IP)*W(JU(IP))
10.          W(I) = Z
11.   40   I=I  —  1
12.      IF(I.GT.0)GOTO      20
```

Этот алгоритм аналогичен той части алгоритма из § 7.28, которую составляют строки 2—3 и 14—23. Фактически здесь выполняется обратная подстановка для векторов-столбцов x и w (см. § 7.27 [1]).

9.6. АЛГОРИТМ УМНОЖЕНИЯ ТРАНСПОНИРОВАННОЙ И ОБРАЩЕННОЙ ВЕРХНЕЙ ТРЕУГОЛЬНОЙ МАТРИЦЫ *V* С ЕДИНИЧНОЙ ДИАГОНАЛЬЮ НА ЗАПОЛНЕННЫЙ ВЕКТОР

Этот алгоритм вычисляет вектор $w = U^{-T}x$.

Вход: IU, JU, UN матрица U в форме RR (U) U
X заданный вектор
N порядок матрицы U
Выход: W вектор-произведение.

[1]) Откуда следует, что массивы X и W при желании можно совместить.— *Прим. перее.*

```
1.              NM=N-1
2.              DO 10 I=1,N
3.     10       W(I) = X(I)
4.              DO 30 I = 1,NM
5.              IUA = IU(I)
6.          IUB=IU(I          + 1)-1
7.         IF(IUB.LT.IUA)GOTO          30
8.              Z-W(I)
9.              DO 20 IP = IUA,IUB
10.             J=JU(IP)
11.    20       W(J) = W(J)-Z*UN(IP)
12.    30       CONTINUE
```

Этот алгоритм эквивалентен первой части алгоритма, описанного в § 7.28 [1]). Выполняемые здесь вычисления иллюстрируются уравнениями параграфа 7.27.

9.7. РЕШЕНИЕ ЛИНЕЙНОЙ СИСТЕМЫ ИТЕРАЦИОННЫМ МЕТОДОМ ГАУССА—ЗЕЙДЕЛЯ

В этом параграфе мы рассмотрим итерационный метод Гаусса—Зейделя для решения системы из N линейных уравнений

$$Ax = b \tag{9.7}$$

с разреженной матрицей A. Известно, что метод сходится для симметричной и положительно определенной матрицы A [Fox, 1965]. Мы можем переписать n-е уравнение системы (9.7) в виде:

$$\sum_{j=1}^{n-1} a_{nj}x_j + a_{nn}x_n + \sum_{j=n+1}^{N} a_{nj}x_j = b_n. \tag{9.8}$$

Если $a_{nn} \neq 0$, то отсюда следует:

$$x_n = a_{nn}^{-1}\left(b_n - \sum_{j=1}^{n-1} a_{nj}x_j - \sum_{j=n+1}^{N} a_{nj}x_j\right). \tag{9.9}$$

Если A положительно определена, то выполнено неравенство $a_{nn} > 0$. Уравнение (9.9) решается итерационно. Может быть задано начальное приближение x_j^0, $j = 1, 2, \ldots, N$; в противном случае можно положить

$$x_j^0 = a_{jj}^{-1}b_j, \quad j = 1, 2, \ldots, N. \tag{9.10}$$

Далее на m-м итерационном цикле вычисляем

$$x_n^m = a_{nn}^{-1}\left(b_n - \sum_{j=1}^{n-1} a_{nj}x_j^m - \sum_{j=n+1}^{N} a_{nj}x_j^{m-1}\right). \tag{9.11}$$

В некоторых случаях сходимость можно ускорить, используя прием верхней релаксации. Здесь

$$x_n^m = x_n^{m-1} + f\left(x_n^{m*} - x_n^{m-1}\right); \qquad (9.12)$$

x_n^{m*} — значение, полученное согласно (9.11). Релаксационный множитель f обычно выбирается в пределах от 1 до 2.

9.8. АЛГОРИТМ ИТЕРАЦИОННОГО РЕШЕНИЯ ЛИНЕЙНОЙ СИСТЕМЫ МЕТОДОМ ГАУССА — ЗЕЙДЕЛЯ

Этот алгоритм решает систему $Ax = b$.

Вход:	IA, JA, AN	заданная матрица A в форме RR (LU) U
	AD	диагональные элементы A; все они не равны нулю
	B	вектор правых частей b
	N	порядок системы
	F	релаксационный множитель
	EPS	допуск на абсолютную величину приращения в тесте на сходимость
Выход:	X	вектор неизвестных x

```
1·          DO 10 I = 1,N
2.    10    X(I)=B(I)/AD(I)
3.          IT = 0
4.    20    IT = IT+1
5.          IEND=0
6.      DO      40    I=1,N
7.    IAA   =       IA(I)
8.      IAB         = IA(I+1)-1
9.          IF(IAB.LT.IAA)GOTO40
10.         U = B(I)
11.         DO 30 J = IAA,IAB
12.   30    U=U-AN(J)*X(JA(J))
13.         U=U/AD(I)-X(I)
14.         IF(ABS(U).GT.EPS)IEND=1
17.           X(I)=X(I)+F*U
16.   40    CONTINUE
17.         IF(IEND.EQ.I)GOTO 20
```

В цикле DO 10 (строки 1—2) вычисляются начальные значения для вектора неизвестных X. Этот цикл можно опустить, если в X уже задано начальное приближение. IT — это счетчик итераций. В строке 3 он получает начальное значение, а в строке 4 увеличивается. Переменная IT включена в алгоритм, чтобы при необходимости можно было выводить промежуточные результаты, проверять условие сходимости и задавать границу для допустимого числа итераций. В цикле DO 40 по очереди обрабатываются N уравнений. Цикл DO 30 выполняет вычисления, указываемые уравнением (9.11) предыдущего параграфа. В строке 13 найденное

значение приращения $x_i^{m*} - x_i^{m-1}$ присваивается переменной U. Затем проверяется условие выхода (строка 14), и если оно не удовлетворено, то парамегр IEND, которому в строке 5 было придано значение 0, переопределяется на 1. В строке 15 в соответствии с уравнением (9.12) вычисляется и помещается в X (1) значение x_i^m. F — релаксационный множитель. Наконец, если условие выхода не выполнено, то управление будет передано из строки 17 в строку 4, и начнется новая итерация. Заметим, что алгоритм завершает работу лишь в том случае, если тест в строке 14 удовлетворен для *каждого* из N уравнений.

Эффективность алгоритма можно повысить, если (как это и имеет место на многих машинах) умножение выполняется быстрее, чем деление. Если в массиве AD находятся числа, обратные к диагональным элементам, то деления в строках 2 и 13 нужно заменить умножениями.

9.9. ПРОВЕРКА ПРЕДСТАВЛЕНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ

Положения, обсуждаемые в этом параграфе, могут быть полезны при отлаживании новой программы, проверке входной информации, проверке промежуточных результатов для новой задачи и т. д. Необходимость в алгоритме для подобных целей проистекает из того факта, что многие программы для разреженных матриц *требуют,* чтобы представления входных данных были стандартными. Вследствие таких особенностей, как наличие указателей, косвенная адресация и т. д., программы очень чувствительны даже к незначительным ошибкам входной информации и могут давать совершенно непредсказуемые результаты. Выявление ошибок в этих случаях затруднено; следовательно, первым шагом должна быть тщательная проверка входных данных. Разреженная матрица, задаваемая в строчном формате, описывается следующими характеристиками:

lA, JA, AN, AD	портрет и численные значения (см. описание в § 1.8—1.9)
NR	число строк
NC	число столбцов
KS	условленный признак, указывающий, в каком представлении задана матрица
KOD	условленный признак, указывающий, задан ли только портрет матрицы или и портрет, и численные значения

Алгоритм проверки должен включать в себя исследование по меньшей мере таких свойств:

(1) NR и NC должны быть положительны; для симметричных матриц должно быть NR = NC.

(2) Массив IA должен содержать NR + 1 положительных чисел; каждое ие может быть меньше предыдущего. В некоторых случаях требуется, чтобы IA (1) = 1.

(3) Целые числа, хранящиеся в JA, должны находиться в пределах: от 1 до NC — для формы RR (C) U; от J + 1 до NC — для формы RR (U) U; от I до NC — для формы RR (DU) U и т. д. Здесь I — номер строки.

(4) В представлении любой строки в JA не должно быть повторяющихся столбцовых индексов.

(5) Допускается присутствие нулей в массиве AN; однако пользователю полезно было бы знать их количество: слишком большое число нулей может свидетельствовать об ошибке или по крайней мере о нежелательном ходе процесса.

(6) Если матрица положительно определена, то в массиве AD должны храниться только положительные числа.

Мы не приводим здесь программу, потому что она существенно зависит от соглашений, принятых пользователем (например, относительно KS и KOD), и от предполагаемого характера сообщений об ошибках, выдаваемых ею. После освоения материала гл. 1 написание такой программы будет для читателя, знающего Фортран, несложным упражнением.

9.10. ВЫВОД РАЗРЕЖЕННОЙ МАТРИЦЫ НА ПЕЧАТЬ ИЛИ ЭКРАН

Может существовать ряд причин, побуждающих пользователя к визуальному изучению его разреженной матрицы. Но если он получит обычную распечатку массивов lA, JA, AN, AD, то скоро поймет, насколько трудно использовать такую информацию в разумных целях. Поэтому было бы полезно иметь программу, представляющую выводимую на печать информацию в некотором доступном для обозрения виде. Программа должна позволять пользователю указание тех строк и столбцов, которые он хотел бы исследовать. Для представления матрицы можно выбрать одну из следующих форм:

(1) Представление в виде полной матрицы. Для каждой строки значения ненулевых элементов загружаются в полный вещественный массив, которому предварительно придано нулевое начальное состояние. Строка массива выводится на печать или дисплей и алгоритм переходит к обработке следующей строки матрицы. Очень удобно было бы иметь возможность различать визуально внутрипортретные нули (т. е. нули, помещенные в AN или AD вследствие взаимного сокращения при вычислениях) и внепортретные нули

(т. е. элементы, о которых заранее известно, что они будут точными нулями, и которые поэтому не включаются в JA). В позициях, соответствующих внепортретным нулям, можно печатать нечисловой символ, например *. Разумеется, практически этот метод приложим только к ситуациям, когда достаточно исследовать малую часть матрицы.

(2) Для каждой строки печатается ее номер, а затем ненулевые элементы этой строки и за каждым из них в скобках соответствующий столбцовый индекс. Еще лучше было бы упорядочить ненулевые элементы перед печатанием. Достоинство этого метода в том, что сокращается пространство, занимаемое выводимой строкой; однако он не дает такого ясного представления о взаимном расположении элементов соседних строк, как первый метод.

(3) Портрет матрицы можно вывести на устройство с высокой разрешающей способностью, например дисплей или матричное печатающее устройство. Нужно только, чтобы матрица была не слишком велика или чтобы было достаточно рассматривать ее частями. Такие портреты стали весьма популярными; см., например, [Duff, 1981a]. Они дают только глобальную информацию относительно числа и распределения ненулевых элементов.

Составление соответствующей программы — простая задача для программиста. Хорошее обсуждение этой темы можно найти в [Gentleman, George, 1976].

9.11. АЛГОРИТМ ПРЕОБРАЗОВАНИЯ ПРЕДСТАВЛЕНИЯ СИММЕТРИЧНОЙ МАТРИЦЫ ИЗ ФОРМЫ RR(C)U В ФОРМУ RR(U)U

Вход:	IA, JA, AN	заданная разреженная симметричная матрица в форме RR (C) U
	N	порядок заданной матриины
Выход:	IA, JA, AN, AD	заданная матрица в форме RR (U) U

```
1              JP = J
2.             IAB-IA(I)-1
3.             DO 40 I = 1, N
4.             AD(I)=0.
5.            IAA = 1AB+1
6.             IAB = IA(I + 1)-1
7.             IF(IAB. LT. IAA)GO TO 40
8.             DO 30 IP = IAA,IAB
9.             J=JA(IP)
10.            IF(J-l)3O,10,20
11.    10      AD(I) = AN(IP)
12.            GOTO30
13.    20      JA(JP)=J
14.            AN(JP) = AN(IP)
15.           JP=JP+1
16     30      CONTINUE
17.    40  IA(I + l) = JP
```

В этом алгоритме цикл DO 40 по очереди обрабатывает N строк заданной матрицы. Для каждой строки I вычисляются указатель начала IAA и указатель конца IAB описания этой строки в массивах JA и AN (строки 5—6 программы). Заметим, что первоначальное значение указатель IAB получил в строке 2. На входе массив AD не используется, поскольку заданное представление полное. AD используется только на выходе, и поначалу в него должны быть засланы нули (строка 4), так как *заранее не* известно, имеется ли в каждой строке ненулевой диагональный элемент. В цикле DO 30 просматриваются ненулевые элементы строки I; при этом J (строка 9) — столбцовый индекс такого элемента. Если J < I (ненулевой элемент левее диагонали), то управление передается нз строки 10 прямо в строку 16, и этот элемент не учитывается. Если J = I, то управление будет передано в строку 11, и в AD вставляется ненулевой диагональный элемент. При J > I ненулевой элемент записывается в массивы JA и AN (строки 13—14). Переменная JP является указателем для обоих массивов; начальное значение она *получает в* строке 1, а увеличивается — в строке 15. Она же используется в строке 17 для записи в IA указателя начала следующей строки I + 1.

9.12. АЛГОРИТМ ЛЕВОГО УМНОЖЕНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ *A* НА ДИАГОНАЛЬНУЮ МАТРИЦУ D

Вход: IA, AN заданная матрица *A в* форме RR (C) U. Заметьте, что массив JA не нужен
D полная диагональ заданной диагональной матрицы
N порядок *D* и число строк *A*

Выход: IA, AN матрица-произведение *B — DA:* заметьте, что результат записывается в прежний массив AN. Прежний же массив JA будет обслуживать матрицу *B*.

```
1.          DO 20  I=1,N
2.          IAA = IA(I)
3.       IAB=IA(I       + 1) - 1
4.          IF(IAB.LT.IAA)GOTO 20
5.          DD = D(I)
6           DO  10 J = IAA,IAB
7.    10    AN(J) = AN(J)*DD
8.    20    CONTINUE
```

В цикле DO 20 по очереди обрабатываются N строк заданной разреженной матрицы. Для каждой строки I переменной DD присваивается (строка 5 программы) значение множителя D (I). Затем в цикле DO 10 просматриваются и умножаются на DD (строка 7) ненулевые элементы строки I. Этот алгоритм не меняет порядка, в котором заданы ненулевые элементы.

9.13. АЛГОРИТМ КОПИРОВАНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ ИЗ IA, JA, AN В IB, JB, BN

Вход:	IA, JA, AN	заданная матрица в произвольном строчном представлении
	N	число строк заданной матрицы
Выход:	IB, JB, BN	заданная матрица, перенесенная в новые массивы.

```
1.          NH=N + I
2.          DO 10 I = l,NH
3     10    IB(I)=IA(I)
4.    IAA   =     IA(I)
5.          IAB = IA(NH)-1
6.          DO 20  I=IAA,IAB
7.          JB(I) = JA(I)
8.    20    BN(I) = AN(I)
```

Этот простой алгоритм легко приспособить к решению других задач: изменению знака у элементов заданной матрицы, умножению заданной матрицы на константу и т. д.

ОГЛАВЛЕНИЕ