Дж. РАЙС

МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ И МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ


ИЗДАТЕЛЬСТВО
МИР

MATRIX COMPUTATIONS AND MATHEMATICAL SOFTWARE

John R. Rice

Professor of Mathematics and Computer Science
Purdue University

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá
Hamburg Johannesburg London Madrid Mexico Montreal
New Delhi Panama Paris São Paulo Singapore Sydney Tokyo Toronto

1981

Дж. Райс

МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ И МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Перевод с английского
О. Б. Арушаняна

под редакцией
В. В. Воеводина

МОСКВА «МИР» 1984

ББК 22.19
Р 18
УДК 512.8+518.12

Райс Дж.

Р 18 Матричные вычисления и математическое обеспечение: Пер. с англ.— М.: Мир, 1984.— 264 с., ил.

В книге рассматриваются этапы конструирования практических вычислительных алгоритмов на примере решения систем линейных уравнений. Материал изложен просто и понятно. Приводится описание нескольких пакетов и библиотек программ, созданных в последние годы в США и Великобритании и нашедших широкое практическое применение.

Для математиков-прикладников, программистов, студентов и аспирантов университетов.

Р $\frac{1702070000\text{-}145}{041\ (01)\text{-}84}$ 39-84 ч. 1.

ББК 22.19
518

Редакция литературы по математическим наукам

Джон Райс

МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ И МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Научн. ред. К. Г. Батаев, И. А. Маховая. Мл. научн. ред. Т. А. Денисова. Художник В. В. Кошмин. Художественный редактор В. И. Шаповалов. Технический редактор М. А. Страшнова. Корректор С. А. Денисова

ИБ № 3865

Сдано в набор 25.07.83. Подписано к печати 29.12.83. Формат 60×90$^1/_{16}$. Бумага типографская № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 16,50. Усл. кр.-отт. 16,50. Уч.-изд. л. 15,63. Изд. № 1/2913. Тираж 14 000 экз. Зак. 2002. Цена 1 р. 40 к.

ИЗДАТЕЛЬСТВО «МИР» 129820, Москва, И-110, ГСП, 1-й Рижский пер., 2

Ордена Октябрьской Революции и ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

Предисловие редактора перевода

Известно, что в настоящее время стоимость математического обеспечения вычислительных машин составляет существенную часть общей стоимости вычислительных систем. Это соотношение, по-видимому, не только сохранится в будущем, но, скорее всего, изменится в сторону еще большего увеличения относительной стоимости математического обеспечения. Очевидно, что в этих условиях высокое качество становится одной из важнейших характеристик математического обеспечения, особенно с точки зрения гарантии достоверности его работы.

Математическое обеспечение создается специалистами, которых почему-то принято называть программистами, хотя, как правило, они имеют специальное математическое образование. Но если роль математического обеспечения постоянно возрастает, то этого никак нельзя сказать в целом о положении программиста в научном обществе. К сожалению, до сих пор для специалиста-математика считается престижнее доказать какую-нибудь теорему и построить какой-нибудь численный метод на бумаге, чем сделать и внедрить в практику хорошую программу. Причин подобного отношения к труду программистов довольно много, но главная заключается в устойчивой недооценке и даже непонимании тех трудностей, с которыми приходится сталкиваться при создании высококачественного математического обеспечения.

Вниманию читателя предлагается книга Дж. Райса «Матричные вычисления и математическое обеспечение». В ней автор попытался на примере достаточно простых задач, связанных с решением систем линейных алгебраических уравнений и проблемы среднеквадратичного приближения, рассказать и детально проанализировать особенности и трудности создания соответствующего математического обеспечения.

Эту книгу не стоит читать «по диагонали» — пользы от такого чтения будет немного. Внешне материал выглядит достаточно простым, математический аппарат хорошо известен, хотя некоторые аспекты, связанные с оценкой достоверности получаемых

результатов, могут быть интересны и квалифицированному специалисту. Главное заключается в другом — именно на этих хорошо известных в математике задачах показано, как трудно создать хорошую программу даже в той области, где кажется, что уже нет никаких проблем.

В книге затрагивается очень много вопросов, связанных с разработкой высококачественного математического обеспечения. Многие из них рассмотрены весьма схематично. Однако большим достоинством книги является то, что рассмотрен весь спектр основных вопросов. Эта книга, безусловно, будет интересна нашим читателям, особенно тем из них, кто хочет понять, почему до сих пор так мало хороших программ и как создавать хорошие программы.

В. В. Воеводин

Предисловие

Эта книга объединяет два очень разных, но дополняющих друг друга предмета: матричное исчисление и математическое обеспечение [1]). Решение линейных уравнений представляет собой типичный образец численных расчетов, которыми занимались еще в древности. Это основной «строительный блок» для алгоритмов решения задач науки, техники, промышленности и любой другой области человеческой деятельности, в которой используются математические модели. Математическое обеспечение стало наиболее распространенным средством проведения численных расчетов вследствие значительного повышения надежности и существенного снижения себестоимости его компонентов. Матричное исчисление может послужить идеальным введением для изучающих математическое обеспечение, поскольку оно является ясной, простой (но в то же время *не слишком* простой) областью знаний, полезной для всех технических приложений.

Цель той части книги, которая посвящена матричному исчислению, заключается в том, чтобы представить основные алгоритмы решения линейных систем и расчетов по методу наименьших квадратов (статистической регрессии). Более сложные задачи вычисления собственных значений и сингулярного разложения здесь опущены. Однако мы непосредственно сталкиваемся с наиболее трудной проблемой, возникающей в численных расчетах: *как узнать, что получены верные ответы?* И опять матричное исчисление оказывается идеальным средством для исследования этой центральной проблемы.

Особое значение придается пяти аспектам математического обеспечения: учету человеческого фактора, оценке рабочих характеристик математического обеспечения, технологии создания его компонентов, конструированию алгоритмов и обучению студентов навыкам разработки программ. Эти аспекты относятся к общим вопросам проектирования математического обеспечения, и здесь также матричное исчисление оказывается очень полезным.

Оценку рабочих характеристик математического обеспечения легче всего проводить на известных стандартных задачах; задачи

---

[1]) В данной книге под математическим обеспечением понимаются программные средства решения задач вычислительной математики, а также математической статистики.— *Прим. перев.*

матричного исчисления как раз подходят для этой цели. Однако даже в этой области нелегко дать такого рода оценку, поскольку цели, преследуемые при создании математического обеспечения, часто оказываются противоречивыми, а различные виды затрат трудно сопоставлять ввиду отсутствия единого критерия. В истории развития математического обеспечения было гораздо больше неудач, чем успехов. Тем не менее мы должны проявлять настойчивость — слишком высоки затраты кустарного производства. Кроме того, использование компонент математического обеспечения делает проверенные на практике идеи и опыт специалистов доступными даже для тех, кто недостаточно знаком со специальными приемами численных расчетов.

Учебные проекты, приведенные в гл. 12, преследуют три цели: (1) практика в использовании математического обеспечения, созданного специалистами, (2) приобретение опыта в практических вопросах проектирования математического обеспечения и конструирования алгоритмов, (3) ознакомление с более значительными программными комплексами. По каждому проекту учащимся следует написать хорошо организованный и аргументированный отчет. Рекомендуется большинство проектов давать на группу из двух-четырех студентов; это позволит им не только приобрести дополнительный опыт коллективной работы, но и участвовать в существенно более глубоких исследованиях.

Предполагается, что читатели знакомы с линейной алгеброй или теорией матриц в объеме вводного курса и имеют солидную подготовку в программировании — предпочтительно на Фортране, так как бóльшая часть существующего математического обеспечения написана на этом языке. Книга предназначена для студентов младших и средних курсов, однако старшекурсникам и аспирантам она также может быть полезной. При более высокой подготовке учащиеся могут, например, более глубоко проанализировать предлагаемые проекты.

Общий объем материала достаточен для курса длительностью в полсеместра или семестр. Книга пригодна также в качестве подготовительного этапа для более широкого или более специализированного курса. Она основана на курсе лекций, в котором часть материала читалась в течение 8 недель, а затем лектор переходил к линейному программированию и нелинейной оптимизации.

Я благодарю Карла де Бура, Ричарда Хэнсона, Роберта Линча и Гарольда Стоуна за ценные предложения как по деталям, так и по организации книги в целом. Я также благодарю Джона Рейда, предоставившего примеры разреженных матриц для четвертой главы. Разрешения на использование материала, включенного в Приложение, получены от издательств Academic Press, ACM и SIAM.

Джон Р. Райс

Глава 1

ВВЕДЕНИЕ

Цель изучения прикладной математики и вычислительных наук [1]) состоит главным образом в овладении средствами анализа математических моделей, возникающих в хозяйственной и организационной сферах деятельности. В этой книге рассматриваются вопросы анализа моделей простейшего вида, основанных на системах линейных уравнений. Матричное исчисление изучается здесь не только с алгоритмической точки зрения, но и с точки зрения разработки математического обеспечения. Таким образом, мы будем исследовать как алгоритмические средства, так и вопрос о том, как сделать эти средства легко доступными для использования. Прежде чем перейти к изложению этого материала, кратко обрисуем роль теории, методов и математического обеспечения в решении рассматриваемых задач.

Математические модели. Математические модели бывают двух разновидностей: статические (или аналитические) и динамические. *Статическая модель* представляет собой совокупность уравнений, формул, определений, таблиц, соотношений и данных, которые математически описывают ситуацию или явление в предположении об их завершенности. *Динамическая модель* является совокупностью тех же объектов, которые описывают, как изменяется ситуация от одного состояния к другому. В динамических моделях существенным параметром является протяженность процесса, и они применяются итерационно, начиная с некоторого начального состояния с последующим наблюдением за тем, что происходит на более позднем этапе. Не все, но большинство динамических моделей достаточно явно описывают переход от состояния к состоянию, и такой подход к построению динамических моделей часто легко сопоставить с анализом, требуемым для получения той же информации на основе статической модели (если она существует).

Цель значительной части математического и научного анализа состоит в преобразовании заданной модели в *явную* модель:

ответ = формула (данные, параметры, переменные).

К сожалению, это часто невозможно сделать, и некоторые простейшие статические модели определяют ответ А неявно. Простые

[1]) В оригинале — computer science.— *Прим. перев.*

примеры (B и C — исходные данные):

$$A^2 + A\cos A + 6.1 = 4.2\sqrt{B^2+1} - \frac{2.4}{C-5},$$

$$\frac{dA(t)}{dt} = 2A(t)\sin t - B + \frac{\cos(t)}{A(t)}, \qquad A(0) = 1.0.$$

Заметим, что второй пример определяет статическую модель, которая в действительности является известным математическим эквивалентом динамической модели.

С такого рода моделями поступают двояким образом. Первый путь состоит в решении задачи, т. е. проводятся различного рода манипуляции и применяются аппроксимации до тех пор, пока ответ не будет получен в явном виде. Такие манипуляции могут использовать конкретные значения данных и параметров, и поэтому их результатом не является построение явной математической модели; вместо нее строится частное решение. Это обычно происходит при применении численных методов.

Второй путь заключается в *оптимизации*. Некоторые параметры или переменные являются свободными, и их следует подобрать таким образом, чтобы получить «наилучшее» значение для некоторой другой переменной. Приведем простой пример:

$$X = H*R - R^2 + T*(R-5) - \min(0,\ 1.0 - 1.5R*H).$$

Здесь X — жалование, выдаваемое в конечном счете служащему на руки. Предположим, что мы можем управлять параметром R, который представляет собой номинальную ставку заработной платы без учета различного рода добавок или вычетов. Можно применить дифференциальное исчисление, чтобы получить уравнение, определяющее оптимальное значение R:

$$dX/dR = H - 2R + T*R - \frac{d}{dR}[\min(0,\ 1.0 - 1.5R*H)] = 0.$$

Значение R, которое является решением этого уравнения, должно определять наибольшее или наименьшее итоговое жалование X. В данном случае легко видеть, что R определяет наибольшее значение X.

Отметим пять обстоятельств, имеющих отношение к процессу оптимизации: (1) Оптимизация явной аналитической модели приводит к другой математической модели. Вторая модель редко бывает явной. (2) Почти *всякая* оптимизация выполняется посредством приравнивания производной нулю. Исключения сводятся к определенным тривиальным случаям или к тем случаям, когда каждый из них может быть исследован по отдельности. Это обстоятельство часто имеет скрытый характер, так как математические объекты, которые должны быть подвергнуты дифференцированию, недифференцируемы в обычном смысле (вспомним «min» в формуле вычис-

ления итогового жалования). (3) При оптимизации наблюдается тенденция к существенному увеличению сложности, и поэтому зачастую оптимизируемые модели менее сложны, чем решаемые. (4) Большинство методов анализа статических моделей применимо также к динамическим моделям. (5) Легко построить модели, решение или оптимизация которых невероятно трудны. Существуют примеры моделей исключительной важности, полный расчет которых (известными на настоящий момент методами) потребовал бы использования *всех* потенциальных ресурсов страны (человеческих и других) в целом в течение многих лет. Например, что произойдет, если какие-либо компоненты атомной станции выйдут из строя, в результате чего тепловыделяющие стержни расплавятся? Или какое влияние на экономику будет иметь сокращение налогов на 10 миллиардов долларов?

В действительности модели упрощаются таким образом, чтобы они стали доступными для обработки. Однако при этом упрощения не должны приводить к потере связи с реальностью. Часто бывает очень трудно обеспечить эту связь; трудно даже узнать, потеряна она или нет.

Средства анализа. Данная книга не касается моделирования как такового. Мы интересуемся *средствами* решения и анализа моделей. Эти средства распадаются на три основные категории:

Теория дает общее понимание модели и процесса решения задачи. Это один из путей достижения качественного представления о том, что происходит в действительности.

Методы дают средства решения задачи. Метод есть совокупность указаний или шагов решения задачи. Иногда (возможно, часто) можно решить задачу или исследовать модель, не достигнув полного понимания метода или самой задачи.

Математическое обеспечение — это законченный метод, воплощенный в программе для ЭВМ. В самом лучшем случае достаточно просто нажать кнопку РЕШИТЬ, чтобы получить ответ.

Глава 2

ОСНОВНЫЕ ПОНЯТИЯ ЛИНЕЙНОЙ АЛГЕБРЫ

2.А. СПИСОК ПОНЯТИЙ, КОТОРЫЕ ДОЛЖНЫ ЗНАТЬ СТУДЕНТЫ

Существуют три распространенных источника основных понятий линейной алгебры, относящихся к матричному исчислению: элементарный курс линейной алгебры, вводный курс в численный анализ и введение в линейный анализ. Обычно курсы линейной алгебры наименее удовлетворительны для наших целей, поскольку они делают упор на алгебраические аспекты предмета, а не на его анализ; курсы линейного анализа обеспечивают наилучшие предпосылки. Большинство студентов ощущают, что их подготовка в линейной алгебре ниже того уровня, который необходим для удовлетворительного (но неглубокого) понимания матричного исчисления. Материал этой главы изложен преднамеренно сжато в предположении, что читатель имеет возможность пользоваться справочниками.

2.А.1. Векторы

Векторы представляют собой *направленные отрезки* (они имеют длину, направление и положение) в N-мерном пространстве. Они рассматриваются как *векторы-столбцы*, если не оговорено противное:

$$\mathbf{y}=\begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_N \end{pmatrix}.$$

Операция *транспонирования*, которая меняет столбцы на строки и наоборот, обозначается верхним индексом Т. Векторы обычно выражаются через базис: выбирается стандартная совокупность векторов $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N$, и все другие векторы выражаются через базис $\mathbf{b}_i$, $i=1, 2, \ldots, N$,

$$\mathbf{y}=y_1\mathbf{b}_1+y_2\mathbf{b}_2+\ldots+y_N\mathbf{b}_N.$$

Коэффициенты y_i в этом представлении называются *компонентами*

вектора $\mathbf{y}$, а само представление записывается в компактной форме

$$\mathbf{y}=(y_1,\ y_2,\ \ldots,\ y_N)^T.$$

Базисные векторы определяют *систему координат*, и компоненты y_i являются координатами точки конца вектора в этой системе координат. Стандартные *арифметические операции* (для векторов $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ и скаляра a):

Сложение: $\mathbf{x}+\mathbf{y}=\mathbf{y}+\mathbf{x}=(x_1+y_1,\ x_2+y_2,\ \ldots,\ x_N+y_N)^T$,

$$\mathbf{x}-\mathbf{y}=-(\mathbf{y}-\mathbf{x});\quad (\mathbf{x}+\mathbf{y})+\mathbf{z}=\mathbf{x}+(\mathbf{y}+\mathbf{z}).$$

Умножение на скаляр: $a\mathbf{x}=(ax_1,\ ax_2,\ \ldots,\ ax_N)$,

$$a(\mathbf{x}+\mathbf{y})=a\mathbf{x}+a\mathbf{y}.$$

Совокупность векторов $\mathbf{x}_1,\ \mathbf{x}_2,\ \ldots,\ \mathbf{x}_m$ *линейно независима*, если ни одна их *линейная комбинация* $\sum_{i=1}^{m}\alpha_i\mathbf{x}_i$ не равна нулю, за исключением нулевой комбинации, т. е.

$$\sum_{i=1}^{m}\alpha_i\mathbf{x}_i=0 \text{ означает, что } \alpha_i=0 \text{ для всех } i.$$

Пространство *натянуто* на совокупность векторов $\mathbf{x}_1,\ \mathbf{x}_2,\ \ldots,\ \mathbf{x}_m$, если каждый вектор пространства может быть выражен через линейную комбинацию векторов $\mathbf{x}_1,\ \mathbf{x}_2,\ \ldots,\ \mathbf{x}_m$. *Размерность* векторного пространства есть минимальное число из этих векторов, требуемых для получения пространства; каждый базис N-мерного пространства должен иметь в своем составе N векторов. N-мерные векторные пространства для краткости часто называются N-*пространствами*, и N-*векторы* являются векторами в N-пространстве.

Скалярное, или *внутреннее*, *произведение* двух векторов $\mathbf{x}$ и $\mathbf{y}$ имеет вид

$$\mathbf{x}^T\mathbf{y}=\mathbf{x}\cdot\mathbf{y}=\sum_{i=1}^{N}x_iy_i,$$

где x_i, y_i — компоненты векторов $\mathbf{x}$ и $\mathbf{y}$. Два вектора *ортогональны* (перпендикулярны), если $\mathbf{x}^T\mathbf{y}=0$. Размер вектора может быть измерен *нормой* $\|\mathbf{x}\|$, где

$$\|\mathbf{x}\|^2=\mathbf{x}^T\mathbf{x}=\sum_i x_i^2.$$

В случае трех измерений это — обычная евклидова длина. Двойные вертикальные черточки обозначают знак нормы. Часто бывают удобны две другие нормы:

$$\|\mathbf{x}\|_\infty=\max_i|x_i|,$$

$$\|\mathbf{x}\|_1=\sum_i x_i.$$

Евклидова норма обозначается $\|x\|_2=[\sum x_i^2]^{1/2}$. Угол θ между двумя векторами определяется соотношением

$$\cos\theta=\frac{x^Ty}{\|x\|_2\|y\|_2}.$$

Формат записи векторов должен быть согласован с форматом записи матриц, поэтому векторы обычно записываются как *векторы-столбцы*, т. е. вектор $x=(2,1,4,-2)^T$ в действительности является вектором

$$x=\begin{pmatrix}2\\1\\4\\-2\end{pmatrix}.$$

Это усложняет изложение текста, поэтому мы будем записывать векторы горизонтально со знаком транспонирования T, если указание формата столбца не является необходимым. Иногда мы будем использовать также *векторы-строки*, которые представляют собой векторы, чья запись в матричном формате имеет на самом деле горизонтальный вид, например строка матрицы.

Мы ввели векторы как абстрактные объекты и затем перешли к их конкретному представлению с использованием базиса. Базис, который определяет систему координат, существен для вычислений (поскольку вычислительные машины не манипулируют с векторами непосредственно), однако абстрактное геометрическое рассмотрение часто более полезно для понимания того, что происходит в процессе вычислений.

Задачи подраздела 2.A.1

1. Вычислить $\|x\|_1$, $\|x\|_2$ и $\|x\|_\infty$ для каждого из следующих векторов:
   (а) $(1, 2, 3, 4)^T$, (б) $(0, -1, 0, -2, 0, 1)^T$,
   (в) $(0, 1, -2, 3, -4)^T$, (г) $(4.1, -3.2, 8.6, -1.5, -2.5)^T$.

2. Вычислить угол θ между следующими парами векторов:
   (а) $(1, 1, 1, 1)^T$, $(-1, 1, -1, 1)^T$, (б) $(1, 0, 2, 0)^T$, $(0, 1, -1, 2)^T$,
   (в) $(1, 2, -1, 3)^T$, $(2, 4, -2, 6)^T$, (г) $(1.1, 2.3, -4.7, 2\,0)^T$, $(3.2, 1.2, -2.3, -4.7)^T$.

3. Найти евклидову норму вектора $(2, -1, 4)$ и скалярное произведение векторов $(2, -3, 4, 1)^T$ и $(4, -3, 2)^T$.

4. Определить, принадлежит ли вектор $(6, 1, -6, 2)^T$ пространству, натянутому на векторы $(1, 1, -1, 1)^T$, $(-1, 0, 1, 1)^T$ и $(1, -1, -1, 0)^T$. Какова размерность пространства, натянутого на эти три вектора?

5. Образуют ли векторы
   $b_1=(1, 0, 0, -1)^T$, $b_2=(0, -1, 1, 0)^T$,
   $b_3=(0, 0, -1, 1)^T$, $b_4=(1, 1, 0, 0)^T$

базис для векторов размерности 4?

6. Доказать, что три вектора

$$\mathbf{b}_1=(1,\ 2,\ 3,\ 4)^{\mathrm{T}},\quad \mathbf{b}_2=(2,\ 1,\ 0,\ 4)^{\mathrm{T}},\quad \mathbf{b}_3=(0,\ 1,\ 1,\ 4)^{\mathrm{T}}$$

линейно независимы. Образуют ли они базис трехмерного пространства?

7. Предположим, что $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ линейно независимы, а $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m, \mathbf{x}_{m+1}$ линейно зависимы. Показать, что $\mathbf{x}_{m+1}$ является линейной комбинацией $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$.

8. Пусть $\mathbf{x}_1=(1, 2, 1)$, $\mathbf{x}_2=(1, 2, 3)$, $\mathbf{x}_3=(3, 6, 5)$. Показать, что эти векторы линейно зависимы, но на них может быть натянуто двумерное пространство. Найти базис этого пространства.

9. Показать, что для любых двух векторов $\mathbf{x}$ и $\mathbf{y}$ и любой из трех норм (1, 2 или ∞) имеет место неравенство

$$|\|\mathbf{x}\|-\|\mathbf{y}\||\leqslant\|\mathbf{x}-\mathbf{y}\|.$$

Это неравенство выполняется для всех векторных норм.

10. Показать, что для двух ненулевых векторов $\mathbf{x}$ и $\mathbf{y}$ равенство $\|\mathbf{x}+\mathbf{y}\|_2=\|\mathbf{x}\|_2+\|\mathbf{y}\|_2$ выполняется тогда и только тогда, когда $\mathbf{y}=a\mathbf{x}$, где a — неотрицательная константа.

11. Показать, что для двух n-векторов $\mathbf{x}$ и $\mathbf{y}$ выполняется неравенство $\|\mathbf{x}+\mathbf{y}\|\leqslant\|\mathbf{x}\|+\|\mathbf{y}\|$ для любой из трех норм (1, 2 и ∞). *Указание*: для норм 1 и ∞ докажите неравенство для двумерных векторов и затем примените индукцию.

2.А.2. Матрицы

Абстрактные объекты, которые приводят к матрицам, представляют собой *линейные отображения*, *преобразования*, или *функции*, определенные над векторами. Поскольку для векторов были введены координаты, то эти линейные функции могут быть конкретно представлены посредством двумерной таблицы чисел, т. е. *матрицей*. Например:

$$A=\begin{pmatrix}1 & 6 & -2\\ 4 & 17 & -12\\ 0 & 42 & 6.1\end{pmatrix}=(a_{ij}).$$

Если $\mathbf{y}$ есть линейная функция от $\mathbf{x}$, то каждая компонента y_k вектора $\mathbf{y}$ есть линейная функция компонент x_j вектора $\mathbf{x}$. Таким образом, для каждого k мы имеем

$$y_k=a_{k1}x_1+a_{k2}x_2+\ldots+a_{kN}x_N.$$

Коэффициенты собираются в матрицу A, которая и представляет линейную функцию (отображение, преобразование, или соотношение) между векторами $\mathbf{x}$ и $\mathbf{y}$. Линейная функция обозначается $A\mathbf{x}$.

Действия над матрицами определяются правилами, выполняемыми для линейных отображений. Так, $A+B$ есть представление суммы двух линейных функций, представленных в свою очередь

матрицами A и B. Имеем $A+B=C$, где $c_{ij}=a_{ij}+b_{ij}$. *Произведение* AB представляет собой эффект применения отображения B и последующего применения отображения A. Покажем, что $AB=C$, где $c_{ij}=\sum_k a_{ik}b_{kj}$. Для этого обозначим $y=Ax$ и $z=By$. Следовательно, нам необходимо определить матрицу C такую, что $z=Cx$. Выразим это соотношение на языке компонент:

$$y_k=\sum_{j=1}^{N} a_{kj}x_j, \qquad z_i=\sum_{k=1}^{N} b_{ik}y_k.$$

Таким образом,

$$z_i=\sum_{k=1}^{N} b_{ik}\left(\sum_{j=1}^{N} a_{kj}x_j\right)=\sum_{j=1}^{N}\left(\sum_{k=1}^{N} b_{ik}a_{kj}\right)x_j$$

и поэтому c_{ij} определяется указанной выше формулой. Элемент, стоящий на пересечении i-й строки и j-го столбца матрицы C, есть скалярное произведение i-й строки матрицы A и j-го столбца матрицы B.

Справедливы *арифметические* правила

$A+B=B+A$,

$AB\neq BA$, за исключением специальных случаев.

Транспонированная матрица A^T получается отражением матрицы A относительно ее *диагонали* (элементов a_{ii}), т. е. $a_{ij}^T=a_{ji}$. *Единичная* матрица имеет единицы на диагонали, а остальные ее элементы равны нулю:

$$I=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}.$$

Единичная матрица обязательно *квадратная*, т. е. имеет одинаковое число строк и столбцов. Легко видеть, что $IA=AI=A$. *Обратная* матрица A^{-1} есть такая матрица, для которой выполняется равенство $A^{-1}A=I$. Не всякая матрица имеет обратную. Действительно, произведение AB может быть равно нулю, даже если A или B не были нулевыми матрицами:

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

Если матрица A имеет обратную, то о ней говорят, что она *невырожденная*. Следующие высказывания эквивалентны:

матрица A — невырожденная;
обратная матрица A^{-1} существует;
столбцы матрицы A линейно независимы;
строки матрицы A линейно независимы;
равенство $Ax=0$ означает, что $x=0$.

Задача решения *линейных уравнений* состоит в нахождении

такого вектора $\mathbf{x}$ по данной матрице A и данному вектору $\mathbf{b}$, для которого выполняется равенство $A\mathbf{x}=\mathbf{b}$. Если матрица невырожденная (тем самым A — квадратная), то эта задача всегда имеет единственное решение для любого $\mathbf{b}$. Если A имеет больше строк, чем столбцов (т. е. существует больше уравнений, чем переменных), то обычно эта задача не имеет решения. Если A имеет больше столбцов, чем строк, то, как правило, существует бесконечно много решений. Самым древним и стандартным методом решения этой задачи является *метод исключения Гаусса* (мы оставляем в стороне *правило Крамера* из-за его крайней практической неэффективности). Система уравнений называется *однородной*, если правая часть равна нулю, например, $A\mathbf{x}=0$.

Пример 2.1: метод исключения Гаусса для системы третьего порядка

Исходная система

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= 5, \\ 6x_1 - x_2 + x_3 &= 7, \\ 4x_1 - 10x_2 - 12x_3 &= -4 \end{aligned}$$

преобразуется к системе

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= 5, \\ -7x_2 - 11x_3 &= -8 \quad \text{(вычитание первой строки, умноженной на 3, из второй строки)}, \\ -14x_2 - 20x_3 &= -14 \quad \text{(вычитание первой строки, умноженной на 2, из третьей строки)}, \end{aligned}$$

которая в свою очередь преобразуется к системе

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= 5, \\ -7x_2 - 11x_3 &= -8, \\ 2x_3 &= 2 \quad \text{(вычитание второй строки, умноженной на 2, из третьей строки)}. \end{aligned}$$

Матрица последней системы уравнений является *верхней треугольной* (*нижняя треугольная* означает, что все элементы выше диагонали равны нулю). Эта система может быть быстро решена *обратной подстановкой*:

$$x_3 = \frac{2}{2} = 1,$$

$$-7x_2 = -8 + 11x_3 = 3, \qquad x_2 = -\frac{3}{7},$$

$$2x_1 = 5 - 2x_2 - 4x_3 = \frac{13}{7}, \qquad x_1 = \frac{13}{14}.$$

Аналогичный процесс, примененный к системе с нижней треугольной матрицей, называется *прямой подстановкой*.

Матрица U *ортогональна*, если ее столбцы $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ ортогональны и их длина равна 1. Иными словами,

$$\mathbf{u}_i^T\mathbf{u}_j = 0, \text{ если } i \neq j, \text{ и } \mathbf{u}_i^T\mathbf{u}_i = 1.$$

Системы $U\mathbf{x}=\mathbf{b}$ с ортогональными матрицами легко решаются, так как U^TU — единичная матрица (строки U^T являются столбцами матрицы U). Если $U\mathbf{x}=\mathbf{b}$, то $U^TU\mathbf{x}=\mathbf{x}=U^T\mathbf{b}$ и $\mathbf{x}$ вычисляется непосредственно. Таким образом, если известен метод преобразования матрицы A к ортогональной матрице, то этот метод может быть применен для решения системы $A\mathbf{x}=\mathbf{b}$.

Матрицей перестановок называется такая матрица, у которой все элементы либо 0, либо 1, а в каждой строке и в каждом столбце имеется только одна 1. Например,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Умножение матрицы слева или справа на матрицу перестановок имеет результатом перестановку строк или столбцов исходной матрицы (см. задачи 5 и 6 подраздела 2.А.2). Это свойство дает им свое название; матрицы перестановок применяются в формулах для указания перестановок строк и столбцов и редко используются в вычислениях как таковых.

Собственные значения матрицы A — это такие числа λ, для которых выполняется равенство $A\mathbf{x}=\lambda\mathbf{x}$ для некоторого ненулевого вектора $\mathbf{x}$; вектор $\mathbf{x}$ называется *собственным вектором* матрицы A. Эффект линейного отображения собственного вектора состоит в непосредственном умножении этого вектора на константу λ, являющуюся собственным значением. Квадратная матрица порядка N имеет N собственных значений и, как правило, но не всегда, N собственных векторов. *Спектральный радиус* $\rho(A)$ матрицы A — это наибольшее по абсолютной величине собственное значение A. Спектральный радиус играет фундаментальную роль в исследовании сходимости итерационных процессов, включающих матрицы.

Существуют различные способы измерения величины, или *нормы*, матрицы. Мы будем использовать только одну норму, а именно:

$$\|A\| = \max_{\|x\|=1} \|A\mathbf{x}\|.$$

Норма — это наибольшая длина вектора, принадлежащего единичной сфере, которую он имеет после применения линейного преобразования, определяемого матрицей A. Эта норма определяется в терминах векторной нормы, и, следовательно, различные векторные нормы дают различные нормы для матрицы A. Норма может

быть выражена явно через три ранее введенные векторные нормы:

$$\|A\|_1 = \max_{\|x\|_1=1} \|Ax\|_1 = \max_j \sum_{i=1}^{N} |a_{ij}|,$$

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = (\text{наибольшее собственное значение } A^TA)^{1/2},$$

$$\|A\|_\infty = \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_i \sum_{j=1}^{N} |a_{ij}|.$$

1-норма и ∞-норма широко применяются, поскольку они легко вычисляются. Легко показать, что $\rho(A) \leqslant \|A\|$ для любой нормы, поэтому 1-норма и ∞-норма обеспечивают простую оценку спектрального радиуса матрицы.

Интерес к матрицам возникает из их связи с линейными функциями нескольких переменных (такие функции представляют собой естественный способ описания простых математических моделей объектов, зависящих от нескольких переменных). *Линейные функции со многими переменными* «существуют» как абстрактные функции точно так же, как векторы «существуют» как абстрактные

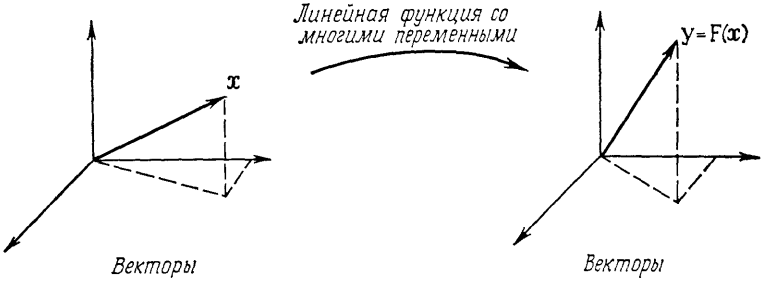


Рис. 2.1. Диаграмма эффекта функции со многими переменными.

объекты. Для векторов мы определили понятия, с которыми мы можем манипулировать и делать вычисления с помощью введения систем координат; то же самое имеет место и для функций со многими переменными, как показано на рис. 2.1. Если мы выразим **x** и **y** в обозначениях некоторой системы координат, то должна существовать формула для вычисления координат $(y_1, y_2, \ldots, y_N)^T$ через координаты $\mathbf{x}=(x_1, x_2, \ldots, x_N)^T$. Эта формула включает в себя двумерную таблицу чисел, скажем $A=(a_{ij})$, и на самом деле имеет вид

$$\mathbf{y}=A\mathbf{x}.$$

Таким образом, *матрица есть конкретное представление линейной функции со многими переменными, полученное от введения системы координат для векторов.* Различные выборы систем координат

дают различные матричные представления для одной и той же функции.

Если $B=C^{-1}AC$, то A и B называются *подобными матрицами*; эта формула выражает эффект изменения систем координат (или другого выбора базисных векторов). Поэтому *все* свойства матрицы A, которые являются свойствами лежащей в ее основе линейной функции F, не меняются после применения *подобного преобразования* $B=C^{-1}AC$. Эти свойства относятся к собственным значениям и собственным векторам.

Как пример полезности собственных значений и собственных векторов рассмотрим, что произойдет, если в качестве базисных векторов выбраны собственные векторы (предположим, что их достаточно для этой цели). В этом случае матричное представление линейной функции есть просто диагональная матрица! Далее, компоненты вектора $A\mathbf{x}$ являются компонентами вектора $\mathbf{x}$, умноженными на соответствующие собственные значения. Поэтому анализ матриц и матричное исчисление существенно упрощаются, если получить собственные векторы матрицы A и использовать их в качестве базисных векторов.

Задачи подраздела 2.A.2

1. Показать, что матрица $A=\begin{pmatrix}1 & 2\\ 2 & 4\end{pmatrix}$ вырождена.

2. Найти ненулевое решение системы

$$\begin{aligned} x_1-2x_2+\ x_3&=0,\\ x_1+\ x_2-2x_3&=0. \end{aligned}$$

3. Пусть

$$A=\begin{pmatrix}1 & 2 & 3\\ 0 & -1 & 2\\ 2 & 0 & 2\end{pmatrix},\quad B=\begin{pmatrix}1 & 1 & 2\\ -1 & 1 & -1\\ 1 & 0 & 2\end{pmatrix},\quad C=\begin{pmatrix}1 & 0 & 1\\ 0 & 1 & 2\\ 2 & 0 & 1\end{pmatrix}.$$

(а) Вычислить AB и BA и показать, что $AB\neq BA$.
(б) Найти $(A+B)+C$ и $A+(B+C)$.
(в) Показать, что $(AB)C=A(BC)$.
(г) Показать, что $(AB)^{\mathrm{T}}=B^{\mathrm{T}}A^{\mathrm{T}}$.

4. Показать, что матрица A вырождена:

$$A=\begin{pmatrix}3 & 1 & 0\\ 2 & -1 & -1\\ 4 & 3 & 1\end{pmatrix}.$$

5. Для матрицы A задачи 4 вычислить PA и AP, где P — матрица перестановок:

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

6. Найти матрицу перестановок P_1, которая для квадратной матрицы A четвертого порядка переставляет второй и четвертый столбцы. Найти P_2, которая переставляет первую и третью строки матрицы A.

7. Напишите следующую систему линейных уравнений в матричной форме $\mathbf{Ax}=\mathbf{b}$:

$$\begin{aligned} 2x_1+x_2-3x_3+4x_4-2&=0,\\ x_1+x_2-x_3&=1,\\ 2x_2+x_4-7x_3&=8,\\ 1+x_1+x_2+x_3+x_4&=5. \end{aligned}$$

8. Показать, что определенная выше матричная норма может быть также вычислена как

$$\|A\|=\max_{\|x\|\neq 0}\frac{\|Ax\|}{\|x\|}.$$

9. Матрица размеров $N\times 1$ является вектором-столбцом. Показать, что результат транспонирования вектора-столбца есть вектор-строка.

10. Пусть B — диагональная матрица. Показать, что матрица BA вычисляется умножением первой строки матрицы A на b_{11}, второй строки — на b_{22} и т. д. Как вычислить AB?

11. Проверить следующие системы уравнений на совместность (т. е. показать существование по крайней мере одного решения):

(а) $\begin{aligned} x+y+2z+w&=5,\\ 2x+3y-z-2w&=2,\\ 4x+5y+3z&=7, \end{aligned}$ (б) $\begin{aligned} x+y+z+w&=0,\\ x+3y+2z+4w&=0,\\ 2x+z-w&=0. \end{aligned}$

12. Показать, что матрица

$$A=\begin{pmatrix}1&1\\0&2\end{pmatrix}$$

не вырождена, вычислив обратную к ней матрицу.

13. Вычислить обратные матрицы для

$$A=\begin{pmatrix}1&1\\0&1\end{pmatrix},\qquad B=\begin{pmatrix}1&0\\1&1\end{pmatrix}$$

и для AB. Показать, что $A^{-1}B^{-1}\neq(AB)^{-1}$, но $B^{-1}A^{-1}=(AB)^{-1}$. Объяснить это, используя определение произведения матриц, представляющих композицию двух линейных функций.

14. Найти одно решение неоднородной системы:

$$\begin{aligned} x_1+2x_2&=3,\\ 2x_1+4x_2&=6, \end{aligned}$$

и затем решение соответствующей однородной системы:

$$\begin{aligned} x_1+2x_2&=0,\\ 2x_1+4x_2&=0. \end{aligned}$$

Показать, что еще одно решение неоднородной системы может быть получено прибавлением к уже найденному ее решению решения однородной системы умноженного на константу.

15. Составьте программу, выполняющую сложение или умножение матриц. Входными параметрами программы являются две квадратные матрицы A и B порядка N, а также переключатель MODE, определяющий вид работы: MODE=1

для сложения и MODE=2 для умножения. Выходным параметром должна быть матрица C=A+B или C=AB.

16. Найти число операций сложения и умножения, необходимых для умножения N×N-матрицы на N-вектор и перемножения двух N×N-матриц.

17. Вычислить $\|A\|_1$ и $\|A\|_\infty$ для матрицы

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

18. Найти ненулевую 3×3-матрицу A и ненулевой вектор **x**, такие, что $A\mathbf{x}=0$.

19. Пусть $A=I-2\mathbf{x}\mathbf{x}^T$, где **x** — вектор-столбец. Показать, что A — ортогональная матрица и $A^2=I$.

20. Для невырожденной матрицы A показать, что $\mathbf{x}^TA^TA\mathbf{x}=0$ тогда и только тогда, когда $\mathbf{x}^T\mathbf{x}=0$.

21. Пусть **x** — вектор-строка. Показать, что $\mathbf{x}\mathbf{x}^T=\mathbf{x}*\mathbf{x}=(\|\mathbf{x}\|_2)^2$. Чему равно $\mathbf{x}^T\mathbf{x}$?

22. Пусть для любого вектора **x** функция F определена одним из следующих способов:

(а) $F(\mathbf{x})=(x_3,\ x_1)^T$;
(б) $F(\mathbf{x})=(x_1+x_2,\ 0,\ x_3)^T$;
(в) $F(\mathbf{x})=(x_2,\ 0,\ x_1-x_2,\ x_3,\ 0)^T$.

Показать, что каждый из этих способов определяет F как линейную функцию.

23. Найти для каждой из линейных функций, определенных в задаче **22**, представляющую ее матрицу.

24. Рассмотрим матрицу

$$A=\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Дать геометрическую интерпретацию функции, представленной матрицей A. Показать, что A — ортогональна.

25. Описать геометрически эффект линейных функций, представленных следующими матрицами:

(а) $\begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$, (б) $\begin{pmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{pmatrix}$,

(в) $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, (г) $\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$.

26. Пусть A — ортогональная матрица и $A\mathbf{x}=\lambda\mathbf{x}$, где $\mathbf{x}\neq 0$. Показать, что $|\lambda|=1$.

27. Пусть вектор **x** имеет компоненты $(x_1,\ x_2)^T$ в обычной эвклидовой системе координат на плоскости [т. е. базисными векторами являются $(1,\ 0)^T$ и $(0,\ 1)^T$]. Какие компоненты имеет вектор **x** в базисе $\mathbf{b}_1=(1,\ 1)^T$, $\mathbf{b}_2=(1,\ -1)^T$? Дать геометрическую интерпретацию соотношения между системой координат, определенной базисом $\mathbf{b}_1$ и $\mathbf{b}_2$, и обычной системой координат.

28. Показать, что матрицы A и B равны тогда и только тогда, когда $\mathbf{Ax}=\mathbf{Bx}$ для всех векторов $\mathbf{x}$.

29. Пусть матрица A имеет самое большее один линейно независимый вектор-столбец. Показать, что тогда матрица A имеет только одну линейно независимую вектор-строку. Показать, далее, что A может быть записана в виде $\mathbf{xy}^T$ для некоторых векторов-столбцов $\mathbf{x}$ и $\mathbf{y}$.

30. Показать, что $\|AB\| \leqslant \|A\| \, \|B\|$ для матричной нормы, определенной в этой главе. Это соотношение имеет место не для всех матричных норм.

31. Доказать правильность данной формулы для нормы $\|A\|_\infty$.

32. Доказать правильность данной формулы для нормы $\|A\|_1$.

33. Показать для N×N-матрицы A, что

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

34. Рассмотреть

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

(а) Описать геометрически множество $S=\{Ax \mid \|x\|=1\}$, которое является образом единичной сферы при линейном отображении, представленном матрицей A.
(б) Что представляют собой $\|A\|_\infty$, $\|A^{-1}\|_\infty$, $\|A\|_1$ и $\|A^{-1}\|_1$?

35. Пусть в *правиле Крамера* каждый определитель вычисляется как сумма (n—1)! произведений, каждое из которых состоит из n множителей. Оценить, сколько тогда потребуется операций умножения, чтобы решить $\mathbf{Ax}=\mathbf{b}$ для n=10, 100 и 1000. Сколько потребуется на это решение машинного времени, если каждая операция умножения (и соответствующие другие операции) выполняется за 1 микросекунду?

36. Повторить решение задачи 35 при условии, что каждый определитель вычисляется за $n^3/3$ операций умножения.

37. Показать, что если строки и столбцы матрицы A линейно независимы, то A — квадратная невырожденная матрица.

38. Пусть A — представляет линейную функцию F [т. е. $y=F(x)$ может быть описана как $\mathbf{y}=A\mathbf{x}$]. Пусть изменения координат в каждом из двух векторных пространств выполняются так, что координаты $(w_1, w_2, \ldots, w_n)^T$ вектора $\mathbf{x}$ в новой системе координат связаны с координатами $(x_1, x_2, \ldots, x_n)^T$ соотношением $\mathbf{w}=B\mathbf{x}$ и подобным же образом новые координаты $(z_1, z_2, \ldots, z_m)^T$ вектора $\mathbf{y}$ вычисляются посредством соотношения $\mathbf{z}=C\mathbf{y}$. Показать, что CAB^{-1} есть матрица, которая представляет функцию F в двух новых координатных системах.

39. Пусть D — диагональная матрица. Показать, что DA=AD для *всех* матриц A тогда и только тогда, когда D=aI для некоторой константы a.

40. Определим e^A разложением $I+A+A^2/2!+A^3/3!+A^4/4!+\ldots$.
Доказать следующие свойства *матричной экспоненциальной функции*:
(а) $\|e^A\|_\infty \leqslant e^{\|A\|_\infty}$;
(б) $e^{A+B}=e^Ae^B$ тогда и только тогда, когда AB=BA;
(в) $e^Ae^{-A}=I$;
(г) $B=e^{At}$ удовлетворяет уравнению $\dfrac{dB}{dt}=AB$.

Глава 3

КЛАССЫ И ПРОИСХОЖДЕНИЕ ЗАДАЧ МАТРИЧНОГО ИСЧИСЛЕНИЯ

3.А. СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ Ax=b

3.А.1. Модели физических систем

Существуют многочисленные физические задачи, которые естественным образом сводятся к линейным уравнениям. Рассмотрим, например, баланс сил, действующих на какую-либо конструкцию (мост, строение, каркас самолета), как это показано на

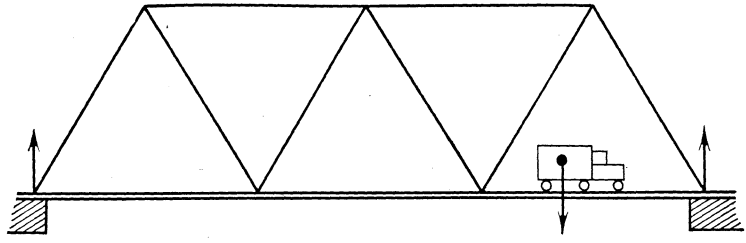


Рис. 3.1.

рис. 3.1. Силы веса моста и грузовика сбалансированы силами, приложенными в местах опоры моста. Эти силы распространяются вдоль железных балок, и в каждой узловой точке их равнодействующая должна равняться нулю (в противном случае мост должен прийти в движение). Если разложить силы на горизонтальные (**x**) и вертикальные (**y**) составляющие, то в каждом узле будем иметь два уравнения:

сумма горизонтальных (**x**) составляющих сил=0,
сумма вертикальных (**y**) составляющих сил=0.

Эти уравнения объединяются в одну большую линейную систему, которая может быть решена, если нужно вычислить неизвестные силы, действующие на различные балки. Некоторые из этих сил известны (веса балок и грузовика), и они образуют члены, которые переносятся в правую часть системы, так что система неоднородна.

Для электрической цепи (рис. 3.2) на основе законов Кирхгофа аналогично описывается баланс сил токов в узлах (точках разветвления) и напряжений по замкнутым контурам. Эти формулы линейны для значений сопротивлений и источников энергии

и сводятся к системе линейных уравнений относительно всех неизвестных величин.

В решении такого типа задач выделяются три этапа:

1. Формулируется *математическая модель* системы (конструкции здания, электрической цепи и т. д.). Явно вычисляются a_{ij}, основанные на физических законах, которым подчиняется система.

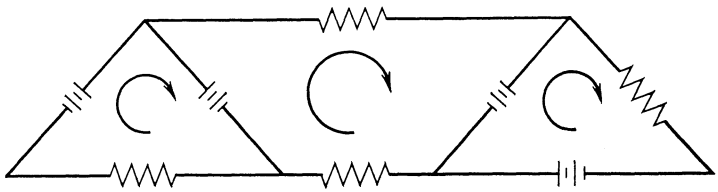


Рис. 3.2.

2. Описывается *вынуждающая функция* (внешние силы, действующие на конструкцию, или внешние нагрузки и подводимая мощность цепи). Явно вычисляются b_j, основанные на анализе внешних условий, характеризующих конкретную ситуацию.

3. Решается линейная система $\mathbf{Ax}=\mathbf{b}$.

Первые два этапа зависят от знания физического состояния исследуемого объекта (строительное дело, физика и т. д.). На третьем этапе используются алгоритмы численных расчетов; одна из целей математического обеспечения — предоставить надежные и эффективные средства для такого рода вычислений.

Многие физические системы моделируются дифференциальными уравнениями, например:

$$\frac{d^2y}{dx^2} + \cos(x)\, y = \log(x+4), \qquad \begin{matrix} y(0)=0, \\ y(1)=1, \end{matrix}$$

которые не могут быть решены аналитически. Приближение этих уравнений конечными разностями основано на дискретизации интервала [0, 1], как показано на рис. 3.3, и замене производной

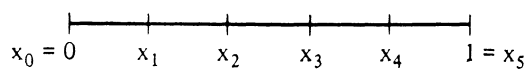


Рис. 3.3.

простой разностью, например:

$$\left.\frac{d^2y}{dx^2}\right|_{x=x_3} = \frac{y(x_2) - 2y(x_3) + y(x_4)}{(0.2)^2} \qquad \left(0.2 = \frac{1}{5} = x_4 - x_3\right).$$

Тогда аппроксимирующее разностное уравнение примет вид

$$\frac{y(x_2) - 2y(x_3) + y(x_4)}{(0.2)^2} + \cos(x_3)\, y(x_3) = \log(x_3 + 4).$$

В каждой точке дискретизации справедливо одно такое уравнение, которое приводит к линейной системе для приближенных значений решения дифференциального уравнения. Если умножить каждое уравнение на $(0.2)^2=0.04$, то линейная система из четырех уравнений примет вид [здесь $y_i=y(x_i)$]:

$$\begin{bmatrix} -2+0.04\cos(x_1) & 1 & 0 & 0 \\ 1 & -2+0.04\cos(x_2) & 1 & 0 \\ 0 & 1 & -2+0.04\cos(x_3) & 1 \\ 0 & 0 & 1 & -2+0.04\cos(x_4) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 0+0.04\log(x_1+4) \\ 0.04\log(x_2+4) \\ 0.04\log(x_3+4) \\ 1+0.04\log(x_4+4) \end{bmatrix}$$

Граничные условия $y_0=0$ и $y_5=1$ были использованы для исключения этих переменных; они вошли в правые части первого и последнего уравнения.

Как и в предыдущем случае, мы видим, что существует этап анализа (вывод дифференциального уравнения и затем нахождение аппроксимирующего разностного уравнения), который может быть достаточно сложным. Затем наступает этап применения алгоритма численного расчета для решения $\mathbf{Ax}=\mathbf{b}$, который может быть выполнен при помощи стандартной компоненты математического обеспечения.

Если дифференциальное уравнение нелинейно, то в результате получается нелинейная система уравнений. Например, предположим, что физическая ситуация приводит к модели, описываемой дифференциальным уравнением:

$$\frac{d^2y}{dx^2}+\cos(x)\,y^2=\log(x+4).$$

Для решения такого рода задач может быть применена следующая бесхитростная, но иногда эффективная стратегия.

Обозначим через $y^0(x)$ некоторое начальное приближение к $y(x)$.

Для $i=1, 2, \ldots,$ <предел> выполнить:

Решить для $y(x)$ уравнение $d^2y/dx^2+\cos(x)y^{i-1}y=\log(x+4)$ и положить $y^i(x)=y(x)$.

Если $\max|y^i(x)-y^{i-1}(x)|$ достаточно мал, то выйти из цикла. Конец цикла.

Согласно этому подходу, во внутреннем цикле вычислений используется процедура решения линейного дифференциального уравнения. Ясно, что процедура внутри цикла может быть выбрана более эффективной, чем многократное повторение решения линейной задачи. Здесь уместно отметить два момента:

1. Программное средство типа «решить $Ax=b$» может быть частью внутреннего цикла более общих вычислений и тем самым существенно, чтобы оно выполнялось эффективно.

2. То же самое программное средство может быть «скрыто» по отношению к решению общей проблемы, и важно, чтобы это средство было надежным. Оно может быть скрыто так глубоко, что программист в действительности и не знает, что используется данное средство, и любая ошибка в нем очень трудно диагностируема.

3.A.2. Выравнивание данных методом наименьших квадратов или регрессия

Пусть заданы 100 результатов наблюдений $d(t_k)$ (данные в 100 точках) и предположим, что функция $d(t)$ моделируется формулой

$$d(t)=c_1g_1(t)+c_2g_2(t)+\ldots+c_7g_7(t),$$

где $g_j(t)$ — известные функции переменной t. Коэффициенты или параметры $c_1, c_2, \ldots, c_7$ должны быть подобраны так, чтобы модель согласовывалась с данными в том или ином смысле. Теоретически имеем

$$d(t_k)=\sum_{j=1}^{7} c_jg_j(t_k) \quad \text{для } k=1, 2, \ldots, 100,$$

но в силу ошибок в регистрации наблюдений или плохого предположения о характере модели или того и другого, вместе взятых, точное равенство в этих 100 уравнениях не достигается для любого выбора семи параметров c_j. Можно, однако, определить c_j так, чтобы сумма $\sum_{k=1}^{100} r_k^2$ квадратов невязок

$$r_k=d(t_k)-\sum_{j=1}^{7} c_jg_j(t_k)$$

была минимальна. Это означает, что параметры c_j оптимизируются так, чтобы минимизировать среднеквадратическую ошибку. Набор параметров c_j, для которого

$$E(c_1, c_2, \ldots, c_7)=\sum_{k=1}^{100} r_k^2$$

минимальна, определяется системой уравнений

$$\frac{\partial E}{\partial c_i}=0, \qquad i=1, 2, \ldots, 7.$$

Имеем

$$\sum_{k=1}^{100}\left[d(t_k)-\sum_{j=1}^{7}c_j g_j(t_k)\right]g_i(t_k)=0, \qquad i=1,\ 2,\ \ldots,\ 7.$$

или, переписав эту систему,

$$\sum_{j=1}^{7}c_j\sum_{k=1}^{100}g_j(t_k)\,g_i(t_k)=\sum_{k=1}^{100}d(t_k)\,g_i(t_k), \qquad i=1,\ 2,\ \ldots,\ 7.$$

Эти выкладки приводят к системе уравнений

$$\mathbf{Gc}=\mathbf{d},$$

где G — матрица с элементами

$$g_{ij}=\sum_{k=1}^{100}g_i(t_k)\,g_j(t_k),$$

d — вектор с компонентами

$$d_i=\sum_{k=1}^{100}d(t_k)\,g_i(t_k),$$

c — вектор искомых параметров. Полученная система называется системой *нормальных уравнений.*

3.А.3. Организационные модели

Две рассмотренные физические системы (мосты и электрические цепи) представляют собой определенные формы организации физических величин. Многие другие формы организации также приводят к линейным моделям. Например, в экономике оплата счета имеет своим результатом увеличение суммы денег у одного лица и уменьшение — у другого. Деньги «делаются» различными путями (открытием золотого рудника, рисованием картин, сооружением письменного стола из пиломатериалов), и можно записать уравнения баланса потока денег (или эквивалентных форм богатства) для экономического региона (деревни, города, страны). Несмотря на то что существует немало неопределенностей в реальном составлении балансов, экономисты создали много больших математических моделей такого рода экономических процессов. В случае установившегося состояния (то есть в случае, когда с течением времени не происходит никаких изменений) эти модели часто представляют собой большие системы линейных уравнений.

Подобная ситуация имеет место и для инвентарного баланса автомобильной компании. Компания владеет автомобилями, находящимися на фабриках, складах, в пути и в розничной торговле. Существуют простые линейные соотношения, которые моделируют изменения в инвентаре. Несмотря на то что нет необходимости в

формулировке «решить задачу инвентаризации», существуют разнообразные манипуляции, которые нужно выполнять с этими линейными моделями. Например, компания нуждается в минимизации своих затрат на складирование и может попытаться найти такое распределение автомобилей, внесенных в инвентарный список, которое обеспечивает это. Очевидное решение — поместить все автомобили в наиболее дешевое место для хранения продукции, но если таким местом окажется сама фабрика, то ни один автомобиль не будет продан. Поэтому определенные *ограничения* должны быть установлены для процесса минимизации. Это — пример задачи *линейного программирования*, предмета, близко примыкающего к матричному исчислению.

Формулирование, анализ и манипулирование с линейными системами уравнений представляют собой составные части процесса решения всех задач такого рода.

3.Б. МАТРИЧНЫЕ УРАВНЕНИЯ AX=B

Матричное уравнение AX=B является в действительности просто множеством линейных систем

$$\mathbf{A}\mathbf{x}_1=\mathbf{b}_1, \quad \mathbf{A}\mathbf{x}_2=\mathbf{b}_2, \quad \ldots, \quad \mathbf{A}\mathbf{x}_m=\mathbf{b}_m$$

для различных векторов решений $\mathbf{x}_i$ и векторов правых частей $\mathbf{b}_i$. Матрица A в уравнении $\mathbf{A}\mathbf{x}=\mathbf{b}$ обычно представляет модель (конструкцию моста, схему электрической цепи, регрессионную модель), а $\mathbf{b}$ представляет вынуждающую функцию или внешние по отношению к модели объекты (нагрузки на мост и электрическую цепь, данные экспериментальных наблюдений). Поэтому для фиксированной модели (A) может возникнуть необходимость в рассмотрении многих различных воздействующих на модель объектов ($\mathbf{b}$), то есть в решении матричного уравнения AX=B. Существо дела состоит в том, что если решается линейная система с одной матрицей и многими правыми частями, то может иметь смысл преобразовать матрицу A к специальной простой форме, с тем чтобы облегчить процесс решения этих многих линейных систем.

Могут быть выделены следующие этапы решения этой задачи:

1. Вычислить матрицу A, моделирующую рассматриваемую систему.
2а. Вычислить вынуждающую функцию $\mathbf{b}_1$.
3а. Решить линейную систему $\mathbf{A}\mathbf{x}_1=\mathbf{b}_1$.
2б. Вычислить вынуждающую функцию $\mathbf{b}_2$.
3б. Решить линейную систему $\mathbf{A}\mathbf{x}_2=\mathbf{b}_2$
и т. д.

Эти этапы могут быть реорганизованы следующим образом:

1. Вычислить матрицу A, моделирующую рассматриваемую систему.

2. Вычислить вынуждающие функции $\mathbf{b}_1$, $\mathbf{b}_2$, . . . и сформировать матрицу B.
3. Решить матричное уравнение AX=B, где X — матрица со столбцами $\mathbf{x}_1$, $\mathbf{x}_2$, ...

Преимущество последнего подхода состоит в том, что на третьем этапе известно о предстоящих значительных вычислениях и они могут быть упорядочены для достижения большей эффективности. Первый подход может быть реализован так же эффективно, но это требует более тщательного программирования, чтобы избежать повторения работы на этапе 3б, которая уже выполнена на этапе 3а. Хорошая программа решает матричное уравнение AX=B эффективно, не вынуждая программиста вникать в детали процесса решения.

3.В. ОБРАЩЕНИЕ МАТРИЦ A^{-1}

В математических и технических текстах часто встречаются формулы вида:

$$\mathbf{x}=A^{-1}\mathbf{b}, \quad \mathbf{y}=B^{-1}(I+2A)\mathbf{b}, \quad \mathbf{z}=B^{-1}(2A+I)(C^{-1}+A)\mathbf{b},$$

которые, естественно, предполагают обращение матриц для вычисления векторов **x**, **y** и **z**. *Такой подход весьма неэффективен: почти никогда нет необходимости обращать матрицы, чтобы вычислить другие объекты.* В гл. 5 будет показано, что вычисление обратной матрицы требует почти в три раза больше операций и, быть может, в два раза больше памяти вычислительной машины, чем решение линейной системы уравнений. В частности, любое матричное выражение, примененное к вектору, может быть эффективно вычислено без обращения любой из входящих в это выражение матрицы независимо от того, сколько обратных матриц имеется в выражении. В большинстве случаев явное обращение матрицы приводит к *вычислительным потерям.*

Пусть решается матричное уравнение AX=B. Вы можете сказать: «Ну, хорошо, непосредственное вычисление A^{-1} — дорого, но уж получив обратную матрицу, я быстро смогу найти все $\mathbf{x}_i$ по формуле $\mathbf{x}_i=A^{-1}\mathbf{b}_i$». Даже эта аргументация неверна, поскольку существуют другие способы (например, LU-разложение матрицы A), которые гораздо дешевле для вычислений, чем A^{-1}, и которые позволяют получить каждое $\mathbf{x}_i$ из $\mathbf{b}_i$ так же быстро, как и умножением $\mathbf{b}_i$ на A^{-1}.

Однако существуют некоторые задачи статистики и техники, в которых цель вычислений состоит в нахождении обратной матрицы как таковой без последующего ее использования для определения чего-либо другого. В этих задачах элементы матрицы A^{-1} имеют смысл «влияющих параметров», которые показывают, как

вынуждающие объекты воздействуют на модель. Если нет необходимости в непосредственном исследовании элементов обратной матрицы, то весьма маловероятно, что следует вычислять обратную матрицу.

3.Г. ОПРЕДЕЛИТЕЛИ det(A)

Определители бесполезны для проведения вычислений в линейной алгебре. Исторически они были введены в математическую и научную терминологию в прошлом столетии, и сейчас они составляют обязательную часть системы образования и математического языка. Определители нужны только в очень немногих разделах теории матричного исчисления, и редко возникает необходимость вычислить в явном виде значение определителя. Например, правило Крамера чрезвычайно неэффективно для решения линейных систем уравнений (см. задачу 35 гл. 2). Часто можно слышать: «Я не могу решить систему, поскольку определитель равен нулю». Однако может ли быть решена система или нет, обнаруживается почти для всех способов вычисления определителя прежде, чем мы получим значение определителя. Сама по себе величина определителя не имеет отношения к достоверности вычислений. Можно быть в полной безопасности, когда $\det(A)=10^{-48}$, и можно получить совершенно неправильные результаты, когда $\det(A)=6.2$. Чтобы убедиться в этом, достаточно умножить каждое уравнение системы 20-го порядка на множитель, равный 100. В результате определитель системы увеличится в 10^{20} раз без какого-либо воздействия на вычислительные трудности или вырожденность системы.

Глава 4

ТИПЫ МАТРИЦ

4.А. СПЕЦИАЛЬНЫЕ МАТРИЦЫ, ВОЗНИКАЮЩИЕ ИЗ ПРИЛОЖЕНИЙ

4.А.1. Разреженные матрицы

Матрицы, в которых большинство элементов равно нулю, называются *разреженными*. Основным источником разреженности являются модели, в которых существуют локальные связи или локальные воздействия. Уравнения в узловых точках моста включают в себя только балки, которые пересекаются в этих точках. Их число в точках пересечения может быть одинаково для моста длиной 50 футов с 10 балками и для моста длиной 5 миль с 10 000 балок. Поэтому для большого моста большинство коэффициентов в каждом отдельном уравнении равно нулю. Такие же рассуждения справедливы для моделей зданий и других физических конструкций.

Разреженность моделей для мостов проистекает из очевидного физического факта, что ни одна балка не тянется вдоль всей длины моста. Этот факт не имеет места для электрической сети, и все же на практике модели электрических сетей также разрежены (чем больше модели, тем выше степень их разреженности), поскольку даже если в принципе две удаленные друг от друга точки сети могут быть соединены проводом, то в большинстве сетей это редко делается. Другой пример. Когда дифференциальные уравнения аппроксимируются разностными уравнениями, разностные аппроксимации полностью локальны. Для дифференциального уравнения в подразделе 3.А.1 соответствующие разностные уравнения имеют только три ненулевых коэффициента в каждом уравнении независимо от их числа.

На практике были решены линейные системы со 100 тысячами неизвестных. Матрица такой системы состоит из 10^{10} элементов, и, если бы она была полной, ее невозможно было бы поместить в память существующих вычислительных систем, не говоря уже о реализации самого процесса решения уравнений. Такая большая система может быть решена тогда, когда она разрежена (или когда применимы весьма специальные формулы). Существуют решенные задачи линейного программирования с 1 миллионом неравенств и 50 тысячами неизвестных. Матрица такой задачи состоит из $5*10^{10}$ элементов, которые почти все должны быть нулевыми, чтобы задача могла быть решена. На рис. 4.1 показан пример расположения ненулевых элементов в двух больших разреженных матрицах, полученных из реальных приложений.

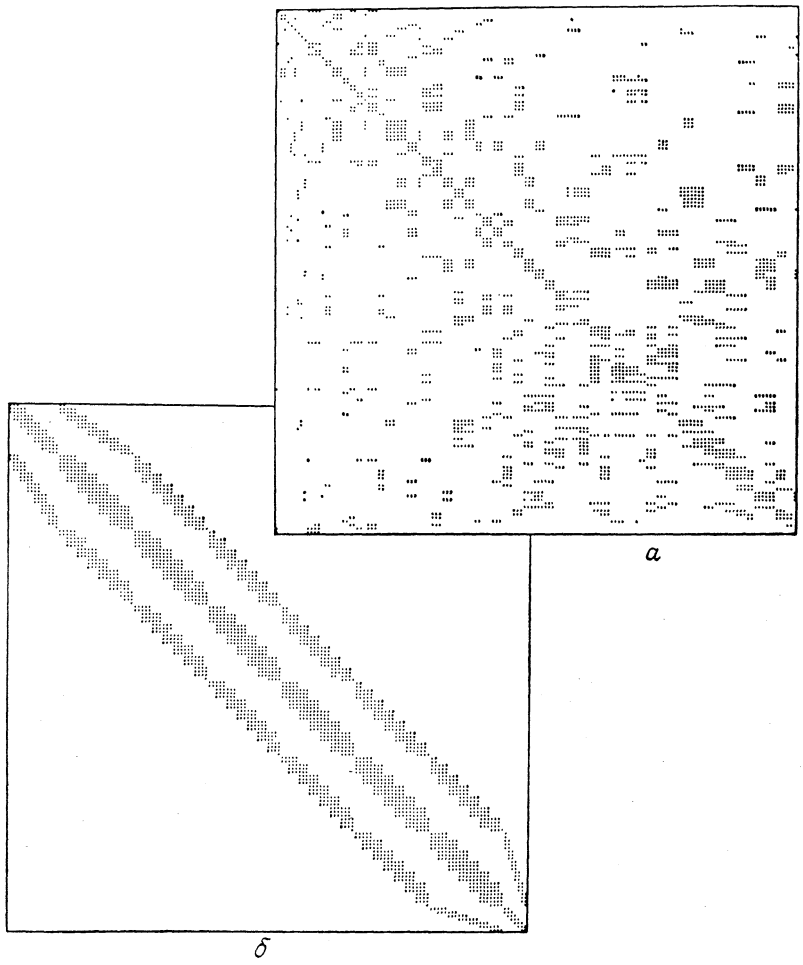


Рис. 4.1. Пример расположения ненулевых элементов в двух больших разреженных матрицах. Каждая точка указывает ненулевой элемент. На практике возникают гораздо большие разреженные матрицы, однако если их структуру уменьшить до данных размеров, то точки были бы не видны [пример любезно предоставлен Джоном Рейдом (John Reid, A.E.R.E. Harwell)].

4.А.2. Упорядоченная разреженность

Многие приложения приводят к матрицам, которые не только разрежены, но и имеют особую структуру расположения ненулевых элементов. Наиболее простые из них — *ленточные матрицы*, в которых $a_{ij}=0$ для $|i-j|>$ширины ленты (рис. 4.2). Например, матрица конечных разностей в подразделе 3.А.1 имеет ширину ленты, равную 1. Такие матрицы называются *трехдиагональными*.

Любая модель, в которой существует локальное воздействие ее составных частей, будет приводить к ленточной матрице, если уравнения и неизвестные соответствующим образом пронумеро-

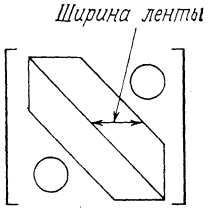


Рис. 4.2.

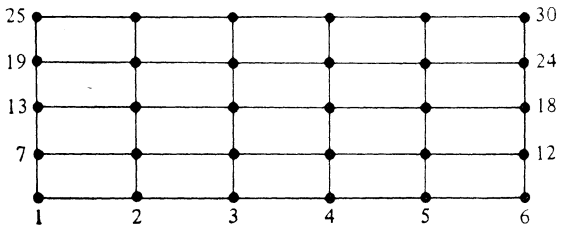


Рис. 4.3.

ваны. Рассмотрим регулярную прямоугольную структуру (рис. 4.3), в которой уравнение в каждой узловой точке связывает переменные, «примыкающие» к этой точке (четыре переменные). Структура ненулевых элементов показана на рис. 4.4. Если лежащий в основе модели физический объект не столь регулярен, то матрица будет хотя и ленточной, но не имеющей систематизированной структуры ненулевых элементов.

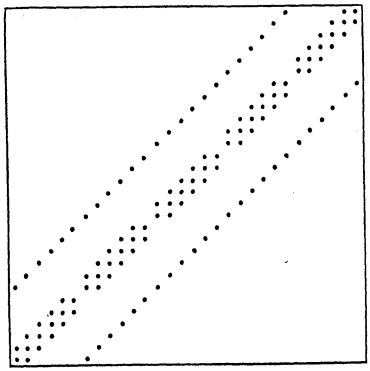


Рис. 4.4.

Вообще говоря, матрицы конечных разностей имеют тенденцию быть систематизированными и ленточными. Модели физических конструкций типа мостов или зданий имеют тенденцию быть ленточными, но с менее систематизированными структурами. Матрицы для электрических цепей наименее структурированы.

4.А.3. Симметричные матрицы

Матрица *симметрична*, если $a_{ij}=a_{ji}$ для всех i и j, или, что эквивалентно, $A=A^T$. Нормальные уравнения симметричны и многие физические задачи равновесия также приводят к симметричным матрицам. Преимущества действий с симметричными матрицами по сравнению с полными матрицами общего вида состоят в том, что для их хранения требуется в два раза меньше памяти вычислительной машины и для большинства случаев в два раза меньше трудозатрат при проведении вычислений. Свойства *положительной определенности* ($\mathbf{x}^T\mathbf{A}\mathbf{x}>0$ для всех ненулевых векторов $\mathbf{x}$) и *положительной полуопределенности* ($\mathbf{x}^T\mathbf{A}\mathbf{x}\geqslant 0$ для всех $\mathbf{x}$) часто связаны с симметричными матрицами.

4.Б. СПЕЦИАЛЬНЫЕ МАТРИЦЫ, ВОЗНИКАЮЩИЕ ИЗ АНАЛИЗА

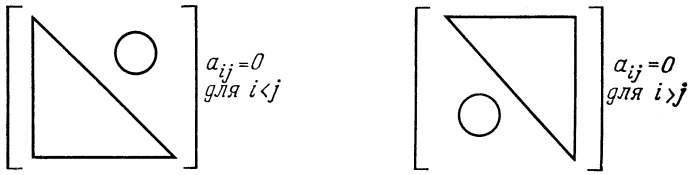
В методах для численных расчетов рассматриваются матрицы, чья специальная форма позволяет легче проводить вычисления. Так, например, цель метода исключения Гаусса — получить матрицу простой структуры, для которой уравнения могут быть легко решены. Рассмотрим некоторые специальные матрицы.

4.Б.1. Диагональные матрицы

Для диагональных матриц $a_{ij}=0$ при всех i и j, кроме i=j. Собственные векторы интересны именно потому, что если они выбраны в качестве базиса (или координатных векторов), то представление соответствующей линейной функции со многими переменными описывается диагональной матрицей.

4.Б.2. Треугольные матрицы

Два вида треугольных матриц показаны на рис. 4.5. Они обладают тем свойством, что над ними легко выполнять различные действия. Линейные системы с треугольными матрицами могут быть



а Нижняя треугольная *б Верхняя треугольная*

Рис. 4.5.

решены прямой или обратной подстановкой. Собственные значения треугольной матрицы суть диагональные элементы.

4.Б.3. Ортогональные матрицы

Если A — ортогональная матрица, то $A^TA=I$ или $A^T=A^{-1}$. В этом случае легко решать линейные системы $Ax=b$, так как $x=A^Tb$. Ортогональные матрицы не меняют длину ($\|x\|_2$) любого вектора. Действительно, пусть $y=Ax$. Тогда

$$\|x\|_2^2=x^Tx=x^TIx=x^TA^TAx=(Ax)^TAx=y^Ty=\|y\|_2^2.$$

Это важно в численных расчетах, когда нежелательно, чтобы выполняемые действия увеличивали как ошибки, возникающие при задании параметров исходной задачи, так и ошибки округления, возникающие в процессе вычислений.

4.Б.4. Трехдиагональные матрицы

Это особый вид ленточной матрицы, для которой $a_{ij}=0$, когда $|i-j|>1$. Такие матрицы возникают и в приложениях, и в анализе. Многие способы вычисления собственных значений матриц (тема, не рассматриваемая в настоящей книге) включают в себя сведение матрицы к эквивалентной трехдиагональной форме.

4.Б.5. Матрицы перестановок

Матрица перестановок P имеет только нули и единицы, причем в каждой строке и в каждом столбце находится только одна единица. Произведение PA имеет те же строки, что и A, но в другом порядке (строки переставлены), в то время как AP — это A с переставленными столбцами. Матрицы перестановок используются главным образом в теоретических исследованиях; на практике используется вектор указателей (косвенная адресация) вместо реальной перестановки строк или столбцов матрицы.

4.Б.6. Разложимые матрицы

Квадратная матрица A называется разложимой, если ее строки и столбцы могут быть переставлены таким образом, что матрица A примет вид

$$A=\begin{pmatrix} B & C \\ 0 & D \end{pmatrix},$$

где B и D — квадратные матрицы. Это свойство позволяет разбить задачу на две меньшие задачи, что обычно дает существенную экономию трудозатрат. К сожалению, такие матрицы редко встречаются на практике; заметим, что диагональная матрица относится к классу разложимых матриц.

Задачи гл. 4

1. Доказать, что сумма и произведение двух нижних (верхних) треугольных матриц имеют своим результатом нижнюю (верхнюю) треугольную матрицу.

2. Доказать, что матрица, обратная к нижней (верхней) треугольной матрице, является нижней (верхней) треугольной матрицей.

3. Пусть A — ленточная $N\times N$-матрица с шириной ленты $K<N/2$. Показать, что A^2 имеет ширину ленты $2K$.

4. Пусть трехдиагональная матрица A факторизована в виде $A=LU$, где L — нижняя треугольная и U — верхняя треугольная матрицы. Показать, что L и U — также трехдиагональны.

5. Показать, что треугольная симметричная матрица является диагональной.

6. Постройте пример, показывающий, что результат произведения двух симметричных матриц не обязательно симметричная матрица.

7. Пусть A — симметричная матрица, а B — любая другая матрица. Показать, что B^TAB — симметричная матрица.

8. Показать, что произведение двух симметричных матриц есть симметричная матрица тогда и только тогда, когда матрицы коммутативны, т. е. $AB=BA$.

9. Пусть A — нижняя треугольная ортогональная матрица. Показать, что A — диагональная матрица. Какие у нее диагональные элементы?

10. Показать, что если столбцы матрицы A линейно независимы, то A^TA — положительно определенная матрица (т. е. $\mathbf{x}^TA^TA\mathbf{x}>0$ для всех $\mathbf{x}\neq 0$). Для матрицы

$$A=\begin{pmatrix} 1000 & 1020 \\ 1000 & 1000 \\ 1000 & 1000 \end{pmatrix}$$

вычислить $B=A^TA$, используя арифметику с четырьмя десятичными цифрами. Показать, что B не является положительно определенной матрицей, рассмотрев вектор $\mathbf{x}=(1, -1)^T$. Объяснить это кажущееся противоречие.

11. Рассмотрим систему уравнений, полученную дискретизацией дифференциального уравнения в подразделе 3.A.1. Показать, что при дискретизации любого дифференциального уравнения вида

$$a(x)\,y''+b(x)\,y'+c(x)\,y=f(x), \qquad y(x_0)=y_0,\quad y(x_1)=y_1,$$

соответствующая линейная система — трехдиагональна.

12. Показать, что нормальные уравнения, выведенные в подразделе 3.A.2, имеют симметричную матрицу коэффициентов.

13. Дать интуитивные объяснения, почему организационные модели должны быть разреженными для каждой из следующих структур: (а) электрическая сеть для коммунальных услуг; (б) региональная система канализации и очистки воды; (в) министерство почт; (г) сборочный конвейер для автомобилей.

14. Показать, как вычислить вектор

$$z=B^{-1}(2A+I)(C^{-1}+A)b$$

без явного вычисления обратных матриц, используя только операции матричной и векторной арифметики и решение систем линейных уравнений.

15. Рассмотрим системы уравнений:

(а) $ix_i=b_i$, $i=1, 2, \ldots, M$;
(б) $x_i/i=c_i$, $i=1, 2, \ldots, M$;
(в) $x_i/10=d_i$, $i=1, 2, \ldots, M$.

Выписать коэффициенты матриц и вычислить их определители для $M=5$, 10, 20, 50. Предполагаете ли вы встретить какие-либо затруднения при решении этих систем уравнений?

Глава 5

МЕТОД ИСКЛЮЧЕНИЯ ГАУССА И LU-РАЗЛОЖЕНИЕ

5.А. ОСНОВНОЙ АЛГОРИТМ И ТЕОРЕМА

Алгоритм исключения неизвестных, рассмотренный в гл. 2, датируется по крайней мере 250 годом до нашей эры, хотя и носит имя Гаусса. Его цель состоит в манипуляциях с уравнениями (в комбинировании строк матрицы A) с тем, чтобы получить эквивалентную задачу (т. е. задачу с тем же самым решением), в которой может быть использована обратная подстановка (матрица A преобразуется в треугольную матрицу). То, что этот процесс всегда может быть выполнен, утверждается следующей теоремой:

Теорема 5.1. *Пусть A — квадратная матрица. Тогда существуют матрицы* U (*верхняя треугольная*), M (*невырожденная нижняя треугольная*) *и* P (*матрица перестановок*), *такие, что*

$$U=MPA.$$

Пусть L *— нижняя треугольная матрица, обратная к* M; *тогда* $LU=PA$.

Метод Гаусса представляет собой алгоритм для определения U, M и P. Напомним, что именно P переставляет строки матрицы A. *Главными минорами* матрицы $\{a_{ij}\}$, $i, j=1, 2, \ldots, n$, называются определители матриц $\{a_{ij}\}$, $i, j=1, 2, \ldots, k$, где $k=1, 2, \ldots, n$. Предыдущий результат может быть выражен в более общей форме:

Теорема 5.2. *Если все главные миноры матрицы* A *отличны от нуля, то матрица* P, *определенная в теореме 5.1, может быть выбрана единичной, и мы получаем*

$$LU=A.$$

При условии нормализации диагональных элементов $l_{11}=1$ *это разложение матрицы* A *единственно.*

Покажем, как теорема 5.1 применяется к задаче решения линейной системы $\mathbf{Ax}=\mathbf{b}$. Если U, M и P уже вычислены, то, умножив обе части системы на MP, получим

$$MPA\mathbf{x}=MP\mathbf{b},$$

что эквивалентно

$$U\mathbf{x}=MP\mathbf{b}.$$

Эти уравнения могут быть решены *обратной подстановкой*. Отметим, что вычисление MPb в точности соответствует прямому ходу.

Пример 5.1: *Решение вырожденной совместной системы.* Формулировка теоремы 5.1 позволяет нам сделать попытку решения вырожденной системы, такой, как

$$\begin{aligned} x+\ y+\ z+\ w&=4,\\ 2x+3y+\ z+\ w&=7,\\ x+\ y+\ z+\ w&=4,\\ x+2y+2z+2w&=7. \end{aligned} \quad \text{или } \mathbf{Ax}=\mathbf{b},$$

После первого шага исключения Гаусса имеем

$$\begin{aligned} x+y+z+w&=4, && \text{множители строки 1,}\\ y-z-w&=-1, && -2,\\ 0&=0, && -1,\\ y+z+w&=3, && -1. \end{aligned}$$

После следующего шага имеем

$$\begin{aligned} x+y+z+\ w&=4, && \text{множители строки 2,}\\ y-z-\ w&=-1, &&\\ 0&=0, && 0,\\ 2z+2w&=4, && -1. \end{aligned}$$

Теперь переставим местами третье и четвертое уравнения, чтобы получить

$$\begin{aligned} x+y+z+\ w&=4,\\ y-z-\ w&=-1,\\ 2z+2w&=4,\\ 0&=0. \end{aligned}$$

Матрица перестановок, которая выполняет эту операцию, имеет вид

$$\mathbf{P}_3=\begin{pmatrix} 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 \end{pmatrix}$$

В этом частном примере мы не будем выполнять действия последнего шага. Матрица M определяется множителями, использованными на последовательных шагах метода исключения Гаусса. Напомним, что основной шаг исключения Гаусса, такой, как «умножить первую строку на m_4 и прибавить ее к четвертой строке», выполняется умножением слева на матрицу

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m_4 & 0 & 0 & 1 \end{pmatrix}$$

которая является единичной с одним дополнительным элементом m_4, расположенным на пересечении первого столбца и четвертой строки. Легко проверить, что такие матрицы подчиняются простому правилу умножения:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m_4 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ m_2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ m_3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}=\begin{pmatrix} 1 & 0 & 0 & 0 \\ m_2 & 1 & 0 & 0 \\ m_3 & 0 & 1 & 0 \\ m_4 & 0 & 0 & 1 \end{pmatrix}$$

Таким образом, эффект выполнения указанных выше манипуляций со строками может быть выражен в матричных обозначениях произведением

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

Исключение **z** — *Перестановка третьей и четвертой строк* — *Исключение* **y** — *Исключение* **x**

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{pmatrix}$$

Суммарный эффект

Легко проверить, что матрица, представляющая «суммарный эффект», имеет вид

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Последняя запись показывает то, что интуитивно ясно: мы могли бы выполнить все перестановки строк заранее (предполагая, будто мы их знаем) и затем применить исключение Гаусса (т. е. вычислить матрицы-множители). Окончательный результат для данного примера имеет вид

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

M	P	A	=	U
Суммарный эффект множителей	*Суммарный эффект перестановок*	*Исходная матрица*		*Верхняя треугольная матрица*

Чтобы продолжить решение, мы вычислим произведение MPb, которое является вектором-столбцом $(4, -1, 4, 0)^T$. Отметим, что на практике мы *не* вычисляем M, а храним именно множители, поскольку мы можем вычислить Mb на их основе так же быстро, как и на основе самой матрицы M.

Теперь попытаемся выполнить обратную подстановку, чтобы найти решение. Мы никоим образом не можем решить последнее уравнение, однако оно автоматически удовлетворяется при любом значении w. Поэтому можно присвоить w произвольное значение. Продемонстрируем обратную подстановку для трех случаев:

Случай $w = 0$:
$2z = 4$, т. е. $z = 2$,
$y = -1 + 2 = 1$,
$x = 4 - 3 = 1$;

Случай $w = 1$:
$2z = 4 - 2$, т. е. $z = 1$,
$y = -1 + 2 = 1$,
$x = 4 - 3 = 1$;

Случай $w = 6.7$:
$2z = 4 - 13.4$, т. е. $z = -4.7$,
$y = -1 + 2 = 1$,
$x = 4 - 3 = 1$.

Это, конечно, иллюстрирует тот факт, что вырожденная система $\mathbf{Ax}=\mathbf{b}$, которая является совместной (т. е. имеет хотя бы одно решение), имеет бесконечно много решений.

Отметим, что если четвертая компонента вектора b *не* равна 7, то последнее уравнение становится уравнением «нуль = что-то ненулевое», которое невозможно решить, и таким образом вырожденная система $\mathbf{Ax}=\mathbf{b}$ совсем не будет иметь решения.
Несложные рассуждения показывают, что:

1. A вырождена, если U имеет нуль на диагонали.
2. Для вырожденной матрицы A система $\mathbf{Ax}=\mathbf{b}$ совместна, если при обратной подстановке нули на диагонали появляются только в уравнениях вида $0=0$. Неизвестные, соответствующие диагональным нулям в этих уравнениях, могут быть определены произвольно.

Доказательство теоремы 5.1. Рассмотренный выше простой пример фактически только иллюстрирует механизм доказательства, однако формальное математическое доказательство все же необ-

ходимо. Доказательство проводится посредством математической индукции по k столбцам матрицы A. Представим матрицу на промежуточном шаге в расчлененном виде так, как показано на рис. 5.1.

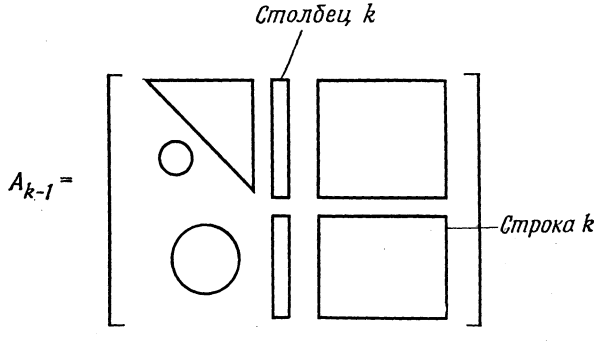


Рис. 5.1.

Пусть по предположению математической индукции $A_{k-1}=$ $=M_{k-1}P_{k-1}A$. Покажем, что k—1 в этом выражении может быть заменено на k.

Существуют два случая (обозначим через a_{ij} элементы матрицы A_{k-1} для упрощения записи):

Случай 1: Предположим, что $a_{ik}=0$ для всех $i\geqslant k$. Теорема доказана.

Случай 2: Предположим, что $a_{jk}\neq 0$ для некоторых $j\geqslant k$. Применим матрицу перестановок P'_k, которая переставляет строки k и j. Теперь мы можем предположить, что $a_{kk}\neq 0$. Вычислим элементы строк от k+1 до n по правилу

$$a_{lm} \leftarrow a_{lm} - \frac{a_{lk}}{a_{kk}} a_{km}, \qquad \begin{matrix} l = k+1, \ \ldots, \ n, \\ m = k, \ \ldots, \ n. \end{matrix}$$

Ясно, что $a_{lk}=0$ для $l\geqslant k$. Это правило состоит в умножении строки k на множители $-a_{lk}/a_{kk}=m_l$, и тем самым его эффект является таким же, что и при умножении на матрицу

$$M'_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & M''_k \end{pmatrix} \qquad M''_k = \begin{pmatrix} 1 & & & 0 \\ m_{k+1} & 1 & & \\ m_{k+2} & 0 & 1 & \\ \vdots & & & \\ m_n & 0 & 0 & 1 \end{pmatrix}$$

где I_{k-1} — единичная матрица порядка k—1. Матрица M'_k не вырождена, так как она является нижней треугольной с единицами на диагонали. Матрицы M_{k-1} и P'_k могут быть представлены в виде

$$M_{k-1} = \begin{pmatrix} L_{k-2} & 0 \\ B & I_{n-k+1} \end{pmatrix} \qquad P'_k = \begin{pmatrix} I_{k-1} & 0 \\ 0 & Q \end{pmatrix}$$

где L_{k-2} — нижняя треугольная матрица порядка k—2 и Q — матрица перестановок порядка n—k+1. Можно проверить, что матрица $P'_k M_{k-1} P'_k$ — нижняя треугольная (результатом этого произведения является воздействие Q на B). Тогда мы имеем

$$P'_k M_{k-1} = P'_k M_{k-1} P'_k P'_k = M^*_{k-1} P'_k,$$

где M^*_{k-1} — нижняя треугольная. Это дает нам возможность записать

$$A_k = M'_k P'_k M_{k-1} P_{k-1} A = M'_k M^*_{k-1} P'_k P_{k-1} A = M_k P_k A,$$

что и требовалось получить. Здесь $P_k = P'_k P_{k-1}$. Правильность гипотезы математической индукции для k=1 устанавливается из тех же соображений, и доказательство завершено.

Доказательство теоремы 5.2. Прежде всего необходимо показать, что отличие от нуля главных миноров означает, что не требуется проводить перестановки строк. Доказательство проводится также по индукции относительно k. Согласно гипотезе математической индукции $a_{11} \neq 0$, так что матрица P_1 не нужна.

Напомним, что треугольная матрица вырождена тогда и только тогда, когда хотя бы один из ее диагональных элементов равен нулю. Тем самым верхняя треугольная подматрица в A_{k-1} не вырождена (по предположению относительно главных миноров) и никаких перестановок строк не требуется. Если $a_{kk}=0$, то главный минор порядка k равен нулю, что является противоречием. Таким образом, $a_{kk} \neq 0$ и доказательство по индукции протекает без перестановок строк.

Построение, используемое при доказательстве теоремы 5.1, дает однозначно определенное разложение, за исключением, возможно, различных выборов строк для перестановок. Далее, матрица M, а тем самым и обратная к ней матрица L имеют единицы на диагонали, т. е. $l_{ii}=1$ для всех i. Поскольку никаких перестановок не нужно производить, они и не требуются в теореме 5.2; полученное разложение единственно и доказательство закончено.

5.Б. ВЫБОР ВЕДУЩЕГО ЭЛЕМЕНТА В МЕТОДЕ ИСКЛЮЧЕНИЯ ГАУССА

Процесс перестановки строк в методе исключения Гаусса, в результате которого получаются ненулевые диагональные элементы, называется *выбором ведущих элементов*, а элементы матрицы, используемые для исключения, называются *ведущими элементами*. Отличие от нуля ведущего элемента достаточно для теоретической правильности метода исключения Гаусса, однако для получения надежного результата требуется бóльшая осторожность. Это видно из следующего примера.

Пример 5.2: Влияние малых ведущих элементов на процесс

исключения Гаусса. Рассмотрим простую систему

$$\begin{aligned} 0.000100x + y &= 1, \\ x + y &= 2, \end{aligned} \quad \text{решение:} \quad \begin{aligned} x &= 1.00010, \\ y &= 0.99990, \end{aligned}$$

которая решается с использованием арифметики с тремя десятичными цифрами. Это означает, что оставляются только три старшие значащие десятичные цифры любого результата арифметических операций. Допустим, что результат округляется. Согласно исключению Гаусса умножим первое уравнение на —10 000 и прибавим ко второму:

$$\begin{aligned} 0.000100x + y &= 1, \\ -10000y &= -10000, \end{aligned} \quad \text{решение:} \quad \begin{aligned} x &= 0.000, \\ y &= 1.000. \end{aligned}$$

Имеет место вычислительная катастрофа.

Переставив уравнения (т. е. выполнив процесс выбора ведущего элемента), получим

$$\begin{aligned} x + y &= 2, \\ 0.000100x + y &= 1, \end{aligned}$$

и исключение Гаусса даст систему (в тех же предположениях о точности вычислений):

$$\begin{aligned} x + y &= 2, \\ y &= 1, \end{aligned} \quad \text{решение:} \quad \begin{aligned} x &= 1.00, \\ y &= 1.00. \end{aligned}$$

Это решение вычислено с той точностью, которую можно получить для арифметики с тремя десятичными знаками.

Вывод из рассмотрения этого примера таков: *недостаточно избегать только нулевых ведущих элементов, необходимо также избегать выбора относительно малых ведущих элементов.*

Существуют две стандартные стратегии выбора. Первая стратегия состоит в *частичном выборе ведущего элемента*: на k-м шаге прямого хода в качестве ведущего берется наибольший (по абсолютной величине) элемент в неприведенной части k-го столбца, т. е.

$$|a_{kk}| = \max |a_{ik}|, \quad i = k, \ k+1, \ \ldots, \ n.$$

Вторая стратегия состоит в *полном выборе ведущего элемента*: на k-м шаге прямого хода в качестве ведущего берется наибольший (по абсолютной величине) элемент в неприведенной части матрицы, т. е.

$$|a_{kk}| = \max |a_{ij}|, \quad i = k, \ k+1, \ \ldots, \ n; \quad j = k, \ k+1, \ \ldots, \ n.$$

Ошибки округления влияют на численные расчеты двояко: чувствительность к ошибкам округления присуща некоторым задачам, а некоторые алгоритмы чрезмерно увеличивают любые

ошибки. Мало что можно сделать, чтобы обойти первую трудность. Можно лишь попытаться идентифицировать такого рода задачи. Выбор ведущего элемента представляет собой технический прием для устранения второй трудности, связанной с эффектом увеличения ошибок округления. В примере 5.2 именно надлежащий выбор ведущего элемента устранил эту трудность.

Не так легко ответить на общий вопрос: насколько хорошо выбор ведущего элемента регулирует увеличение ошибок? В одной из наиболее известных статей фон Неймана и Гольдстайна в 1947 г. показано, что можно ожидать увеличения ошибок в 10^{12} или большее число раз при решении скромной, скажем 20×20, системы методом Гаусса. Казалось бы, безнадежно решать многие линейные системы такого или большего порядка. Однако отдельные исследователи игнорировали этот факт и пытались так или иначе решать большие системы. Оказалось, что можно с достаточной точностью решать системы из 100, 200 и даже 400 уравнений, если позволяли затраты машинного времени. Это открытие привело к попыткам лучше понять метод Гаусса, и к концу 60-х годов были выяснены следующие четыре факта:

Факт 1 (доказанный). Полный выбор ведущего элемента представляет собой надежную стратегию, поскольку ошибки никогда чрезмерно не возрастают. *Коэффициент увеличения* ошибок не превосходит

$$f_n = (n * 2 * 3^{1/2} * 4^{1/3} * 5^{1/4} * \ldots * n^{1/(n-1)})^{1/2}$$

для любой системы уравнений порядка n. Нижеследующая таблица показывает, что f_n растет довольно медленно с возрастанием n

n	5	10	15	20	30	50	80	100
f_n	5.73	18.30	39.09	69.77	155.5	536.17	1915.51	3552.41

и любая ошибка (или ошибки в коэффициентах системы, или ошибки в процессе вычислений), влияющая на решение, умножается самое большее на коэффициент увеличения.

Факт 2 (замеченный экспериментально). Вероятность того, что при использовании частичного выбора ведущего элемента возникнут трудности, связанные с ростом ошибок, весьма мала. В действительности скорость увеличения ошибок для частичного выбора редко превосходит более чем в 2—4 раза скорость роста ошибок для полного выбора. Наблюдаемая на практике величина коэффициента увеличения f_n обычно не превосходит 8 и часто близка к 1, особенно для плохо обусловленных задач.

Факт 3 (на основе примера). Была построена специальная $n\times n$-матрица, для которой при использовании частичного выбора веду-

щего элемента ошибки растут как 2^n. Как видно из нижеследующей таблицы, коэффициент увеличения растет довольно быстро:

n	5	10	15	20	30	50	80	100
2^n	32	1024	32768	10^6	10^9	10^{15}	10^{24}	10^{30}

Если бы издержки полного и частичного выборов ведущего элемента были почти одинаковыми, то всегда следовало бы применять полный выбор. Однако издержки полного выбора сравнимы со всем остальным процессом решения. Вычитания и проверки по абсолютной величине для полного выбора ведущего элемента должны выполняться над всеми элементами матрицы, которые участвуют в исключении Гаусса, и можно прийти к интуитивному выводу, что полный выбор приблизительно удваивает затраты на исключение Гаусса. С другой стороны, издержки частичного выбора почти незначительны.

Факт 4. Таким образом, затраты, связанные с полным обеспечением надежности вычислений, весьма высоки, и можно задать естественный вопрос: «Почему следует идти на двукратные издержки только для того, чтобы оградить себя от столь редких ситуаций, на поиск примеров которых потребовались годы для специалистов по численному анализу?» Именно поэтому полный выбор ведущего элемента редко используется. Это не просто расчетливый риск. Как мы увидим позже, существуют другие способы избежать чрезмерного роста ошибок.

Задачи подразделов 5.А и 5.Б

1. Применить исключение Гаусса и показать, что следующая система не имеет решения

$$\begin{aligned} 3x_1 + x_2 &= 1.5, \\ 2x_1 - x_2 - x_3 &= 2, \\ 4x_1 + 3x_2 + x_3 &= 0. \end{aligned}$$

2. При помощи калькулятора решить следующую систему, используя только три десятичные цифры на каждом шаге. Проверить ответ подстановкой его в уравнения и оценить погрешность. Точное решение системы есть вектор $(1, 1, 1, 1)^T$

$$\begin{aligned} 0.31x_1 + 0.14x_2 + 0.30x_3 + 0.27x_4 &= 1.02, \\ 0.26x_1 + 0.32x_2 + 0.18x_3 + 0.24x_4 &= 1.00, \\ 0.61x_1 + 0.22x_2 + 0.20x_3 + 0.31x_4 &= 1.34, \\ 0.40x_1 + 0.34x_2 + 0.36x_3 + 0.17x_4 &= 1.27. \end{aligned}$$

3. При помощи калькулятора решить систему, используя только четыре правильно округляемые цифры на каждом шаге. Проверить ответ подстановкой его в уравнения и оценить погрешность. Точное решение системы есть вектор $(10, 1)^T$

$$\begin{aligned} 0.0003x_1 + 1.566x_2 &= 1.569, \\ 0.3454x_1 - 2.436x_2 &= 1.018. \end{aligned}$$

4. При помощи калькулятора решить систему, используя только четыре правильно округляемые цифры на каждом шаге. Решить систему три раза: два раза с ведущими элементами a_{11}, a_{21} и один раз с полным выбором ведущего элемента. Сравнить полученные ответы с точным решением $(1.000, 0.5000)^T$

$$0.002110x_1+0.08204x_2=0.08415,$$
$$0.3370x_1+\ 12.84x_2=6.757.$$

5. Показать, что матрица A не вырождена, но не может быть представлена произведением LU нижней и верхней треугольных матриц. Объяснить, почему.

$$A=\begin{pmatrix}1&2&3\\2&4&1\\-1&0&2\end{pmatrix}.$$

6. Рассмотреть теорему 5.1 и объяснить, (а) какова связь теоремы с обычным исключением Гаусса; (б) когда необходимо осуществлять выбор ведущего элемента; (в) почему теорема полезна для решения системы $\mathbf{Ax}=\mathbf{b}$.

7. Дать пример линейной системы $\mathbf{Ax}=\mathbf{b}$, у которой $\det(A)$ много меньше ошибок округления (по сравнению с величиной элементов матрицы A), но для которой нет вычислительных трудностей, имеющих место при решении систем $\mathbf{Ax}=\mathbf{b}$.

8. Выполнить исключение Гаусса для решения системы

$$2x+3y-\ z+\ w=5,$$
$$x+2y+3z-\ w=-6,$$
$$4x+5y-9z+6w=28,$$
$$x+\ y-4z-2w=7$$

и получить *все* ее решения.

9. Матрица A называется матрицей с *диагональным преобладанием*, если $|a_{ii}|>\sum_{i\neq j}|a_{ij}|$ для всех i. Пусть A — симметричная матрица с диагональным преобладанием. После первого шага исключения Гаусса матрица примет вид

$$\begin{pmatrix}a&y\\0&A_1\end{pmatrix}.$$

(а) Показать, что A_1 — матрица с диагональным преобладанием.

(б) Показать, что для симметричной матрицы с диагональным преобладанием процесс исключения Гаусса не зависит от того, производится ли выбор ведущего элемента или нет.

10. Показать, что (а) для матрицы с диагональным преобладанием процесс исключения Гаусса без выбора ведущего элемента не может потерпеть неудачу; (б) если A — матрица с диагональным преобладанием, то A — невырожденная матрица. *Указание*: использовать задачу 9.

11. Пусть A — симметричная положительно определенная матрица (т. е. $\mathbf{x}^T\mathbf{Ax}>0$ для всех $\mathbf{x}\neq 0$). Показать, что (а) A^{-1} — положительно определенная матрица; (б) A может быть представлена в виде LL^T, где L — нижняя треугольная матрица с положительными диагональными элементами; (в) для этой матрицы A процесс исключения Гаусса без выбора ведущего элемента не может потерпеть неудачу. *Указание*: использовать задачу 9.

12. Можно показать, что *коэффициент увеличения* ошибок округления мал для некоторых матриц специального вида. Доказать, что для следующих случаев имеют место указанные предельные значения коэффициентов увеличения ошибок:

(а) для трехдиагональной матрицы коэффициент увеличения $\leqslant 2$;

(б) если A^T — матрица с диагональным преобладанием, то коэффициент увеличения $\leqslant 2$;

(в) для положительно определенной матрицы коэффициент увеличения $\leqslant 1$.

13. Решить систему

$$x_1 + 1.001 x_2 = 2.001,$$
$$x_1 + x_2 = 2.$$

Вычислить невязку (Ay—b) для y=(2, 0) и сравнить относительную величину отклонения от решения с относительной величиной невязки по отношению к правой части. Решить еще одну систему:

$$x_1 + 1.001 x_2 = 1,$$
$$x_1 + x_2 = 0.$$

Вычислить невязку для z=(—1001, 1000)^t и сравнить относительную величину ошибки в решении с величиной невязки по отношению к правой части. Какой вывод можно сделать о невязке, как о критерии оценки погрешности вычисленного решения?

5.В. АЛГОРИТМ ГАУССА: НЕКОТОРЫЕ СВОЙСТВА И МОДИФИКАЦИИ

5.В.1. Исключение Гаусса с частичным выбором ведущего элемента

Представим алгоритм Гаусса на некотором языке программирования в компактной естественной форме. Рассмотрим решение системы $Ax=b$, где A — квадратная $n \times n$-матрица. Алгоритм Гаусса строит верхнюю треугольную матрицу U и матрицу множителей на месте матрицы A, так что элементы матрицы A не сохраняются.

Алгоритм 5.1: Исключение Гаусса с частичным выбором ведущего элемента для решения системы $Ax=b$.

ЦИКЛ ПО СТОЛБЦАМ A

For k=1 to n—1 do

ПОИСК МАКСИМАЛЬНОГО ПО МОДУЛЮ ЭЛЕМЕНТА В СТОЛБЦЕ

imax = индекс строки такой, что $|a_{i\,max,\,k}| = \max |a_{ik}|$, $i \geqslant k$

ПРОВЕРКА НА НУЛЬ ВЕДУЩЕГО ЭЛЕМЕНТА (НА ВЫРОЖДЕННОСТЬ).

BUFFER — МАШИННО-ЗАВИСИМАЯ КОНСТАНТА, ОЦЕНИВАЮЩАЯ ВЕЛИЧИНУ ОШИБОК ОКРУГЛЕНИЯ

If $|a_{i\,max,\,k}| \leqslant$ BUFFER, то перейти на конец цикла по k

ПЕРЕСТАНОВКА СТРОК С ИНДЕКСАМИ K И IMAX

For j=1 to n переставить a_{kj} и $a_{i\,max,\,j}$

ПЕРЕСТАНОВКА СООТВЕТСТВУЮЩИХ ПРАВЫХ ЧАСТЕЙ

переставить b_k и $b_{i\,max}$

ЦИКЛ ПО СТРОКАМ

For $i = k + 1$ to n do
ВЫЧИСЛЕНИЕ МНОЖИТЕЛЕЙ И ЗАПОМИНАНИЕ ИХ В СТОЛБЦЕ К
$m = a_{ik} = a_{ik}/a_{kk}$
ЦИКЛ ПО ЭЛЕМЕНТАМ СТРОКИ I
For $j = k + 1$ to n do
$a_{ij} = a_{ij} - m * a_{kj}$
ВЫЧИСЛЕНИЕ ПРАВОЙ ЧАСТИ СТРОКИ I
$b_i = b_i - m * b_k$
конец цикла по i
конец цикла по k
ОБРАТНАЯ ПОДСТАНОВКА
ЦИКЛ ПО СТРОКАМ В ОБРАТНОМ ПОРЯДКЕ
For $i = n$ to 1 do
ПОДСТАНОВКА ИЗВЕСТНЫХ КОМПОНЕНТ РЕШЕНИЯ В ПРАВУЮ ЧАСТЬ

$$r = b_i - \sum_{j=i+1}^{n} a_{ij}x_j$$

ПРОВЕРКА НА НУЛЬ ВЕДУЩЕГО ЭЛЕМЕНТА
If ($|a_{ii}| > $ BUFFER) then $x_i = r/a_{ii}$
ПРОВЕРКА НА НУЛЬ ПРАВОЙ ЧАСТИ
ПРИСВОИТЬ НУЛЬ КОМПОНЕНТЕ РЕШЕНИЯ И ПЕЧАТЬ СООБЩЕНИЯ
else if ($|r| \leqslant$ BUFFER) then $x_i = 0$
ИЛИ ПЕЧАТЬ СООБЩЕНИЯ ОБ ОШИБКЕ
else печать «несовместная система, A — вырождена»
конец цикла по i

Укажем, как матрицы M и U хранятся на месте матрицы A. Каждый множитель помещается на месте обнуляемого элемента A_{ij}. Поскольку диагональные элементы M равны единице, то они не хранятся явно. Хотя запоминать матрицы M и U не нужно, когда решается лишь система $\mathbf{Ax} = \mathbf{b}$, существуют другие ситуации, когда весьма полезно сохранять обе матрицы M и U.

5.B.2. Масштабирование и анализ машинного нуля

Алгоритм Гаусса требует анализа ситуаций, когда некоторые получаемые числа равны нулю. С математической точки зрения было бы правильным положить BUFFER = 0, однако на практике этого сделать нельзя. Числа, которые при использовании точной арифметики должны быть нулями, почти наверняка не равны нулю при использовании арифметики с плавающей точкой из-за ошибок округления. Чтобы определить, является ли число нулем, необ-

ходимо рассмотреть два вида масштабов: масштаб представления чисел в исходной задаче и масштаб представления чисел в машине.

Машинный масштаб определяется точностью (числом оперируемых цифр) самой машины. Поэтому для машины, которая выполняет действия с 10 десятичными цифрами, ведущий элемент $1.2*10^{-10}$ может рассматриваться как результат влияния ошибок округления и по праву может быть положен равным нулю. Однако это было бы абсолютно неправильным, если все элементы матрицы A имели порядок 10^{-8}. Таким образом, целесообразно масштабировать систему так, чтобы все элементы были порядка 1, и тогда было бы правильным считать, что малые числа порядка $1.2*10^{-10}$ представляют собой помехи из-за ошибок округления.

Существуют два естественных способа масштабирования системы $A\mathbf{x}=\mathbf{b}$. Мы можем умножить уравнения на некоторый множитель (например, на 10^8, если элементы a_{ij} имеют порядок 10^{-8}) для выравнивания коэффициентов. Мы можем умножить также столбцы матрицы A: это соответствует изменению единиц измерения неизвестных (например, переходу от дюймов к милям). К сожалению, существуют задачи, для которых масштабирование не столь легкий процесс, и удовлетворительный метод полностью автоматического масштабирования не найден. Ниже приводятся два примера матриц, масштабирование которых вызывает трудности. Ни один из двух способов масштабирования, ни их комбинация не позволяют провести должное масштабирование этих матриц.

$$\begin{pmatrix} 1 & 10^{10} & 10^{20} \\ 10^{10} & 10^{30} & 10^{50} \\ 10^{20} & 10^{40} & 10^{80} \end{pmatrix} \qquad \begin{pmatrix} 1 & 10^{20} & 10^{10} & 1 \\ 10^{20} & 10^{20} & 1 & 10^{40} \\ 10^{10} & 1 & 10^{40} & 10^{50} \\ 1 & 10^{40} & 10^{50} & 1 \end{pmatrix}$$

К счастью, большинство задач могут быть масштабированы без труда. Решающим является выбор единиц измерения (они определяют величину изменения масштаба), которые естественны для задачи и которые не искажают соотношений между размерами объектов. Указанные выше примеры показывают, что это не всегда может быть сделано, однако, как правило, для реальных задач это может быть сделано легко. Разумно проводить действия с числами обычной величины, и наиболее простой способ масштабирования состоит в таком умножении каждой строки (уравнения), чтобы наибольший элемент строки был равен 1. В предположении, что это сделано, перейдем к выбору величины BUFFER. Если машина имеет t разрядов с основанием b, то в качестве значения BUFFER может быть взято $c*b^{-t}$, где c — некоторое небольшое число. Значение c должно быть равно 3 или 4 и должно постепенно расти с увеличением порядка матрицы. Над каждым элементом

a_{ij} выполняется самое большее $2n^2$ арифметических операций, поэтому $c=n^2$ — самое большее значение, которое следует рассматривать. Однако можно ожидать, что будет иметь место значительное взаимное уничтожение ошибок округления и потому $c=n$ — достаточно надежный выбор. Большинство численных расчетов проводится с относительно высокой точностью (от 10 до 15 десятичных цифр), и в этих случаях можно выбирать значения с достаточно произвольно.

Подведем итоги, отметив следующие три факта:

Факт 1. Вычисления более устойчивы (менее чувствительны к изменениям), если все элементы матрицы приблизительно одинаковы.

Факт 2. Не известны способы масштабирования матриц в общем случае, и существуют матрицы, которые не могут быть удовлетворительно масштабированы.

Факт 3. На практике обычно масштабируют строки (делением каждого уравнения на его наибольший коэффициент). Это достаточно надежный способ для обычно встречающихся задач, и есть основания полагать, что апостериорные проверки, которые будут рассмотрены в гл. 8, дают предупреждающие диагностики, если возникают затруднения при вычислениях.

5.B.3. Модификации алгоритма Гаусса

Существуют много модификаций основного алгоритма исключения Гаусса. Выбор конкретной модификации представляет собой одну из задач проектирования математического обеспечения. Проблема выбора будет рассмотрена подробнее в гл. 6. Здесь мы перечислим некоторые более очевидные модификации:

1. Элементы матрицы A сохраняются; должны быть выделены дополнительные участки памяти в качестве рабочих вместо использования для этих целей частей матрицы A.

2. При выборе ведущих элементов строки матрицы A явно не переставляются; порядок исключения неизвестных определяется совокупностью указателей.

3. Матрица A факторизуется к виду LU; множители L и U используются на следующем этапе для решения системы.

4. Модификация, позволяющая решать систему со многими правыми частями, т. е. матричное уравнение $AX=B$.

5. Вычисление A^{-1} посредством решения матричного уравнения $AX=I$. Здесь правые части суть последовательно столбцы единичной матрицы, а решения уравнений образуют столбцы матрицы A^{-1}. В задаче 19 этой главы показано, что вычисление A^{-1} может

быть выполнено за $n^3/3+2n^3/3=n^3$ операций вместо ожидаемых $n^3/3+n^3=4n^3/3$ операций.

Менее очевидной и более интересной является модификация Краута, основанная на изменении порядка выполнения арифметических операций. Заметим, что в основном алгоритме Гаусса каждый элемент a_{ij} в процессе вычислений изменяется много раз. Легко показать, что не обязательно изменять a_{ij} так, как это было показано выше. Поскольку в любом случае вся информация о процессе исключения сохраняется, можно выполнить все операции над a_{ij} за один прием. Ниже приводится алгоритм модификации Краута без выбора ведущих элементов, обратной подстановки и проверки на нуль ведущих элементов.

Алгоритм 5.2: Модификация Краута для факторизации A

For k = 1 to n do

ВЫЧИСЛЕНИЕ И ЗАПОМИНАНИЕ МНОЖИТЕЛЕЙ В ОСТАВШЕЙСЯ ЧАСТИ СТОЛБЦА К
ВЕДУЩИЕ ЭЛЕМЕНТЫ (ДИАГОНАЛИ U) РАВНЫ ЕДИНИЦЕ

For i = k to n do

$$a_{ik} = a_{ik} - \sum_{i=1}^{k-1} a_{ii}a_{ik}$$

конец цикла по i

ВЫЧИСЛЕНИЕ ЭЛЕМЕНТОВ В ОСТАВШЕЙСЯ ЧАСТИ СТРОКИ К

For j = k + 1 to n do

$$a_{kj} = \left(a_{kj} - \sum_{i=1}^{k-1} a_{ki}a_{ij}\right) \Big/ a_{kk}$$

конец цикла по j

конец цикла по k

Заметим, что этот алгоритм не вырабатывает те же числа, как рассмотренный выше основной алгоритм Гаусса. Различие состоит в способе нормирования разложения MA=U; в алгоритме Гаусса равными 1 получаются диагональные элементы матрицы M, а в алгоритме Краута — диагональные элементы матрицы U. Небольшими изменениями можно добиться того, чтобы с точностью до ошибок округления оба алгоритма давали одно и то же разложение матрицы A. Опять-таки оба множителя этого разложения помещаются на месте матрицы A, так что элементы A не сохраняются.

Даже несмотря на то что модификация Краута состоит лишь в другом порядке проведения операций по сравнению с исключением Гаусса, существуют два случая, когда эта модификация может иметь преимущества. Первое преимущество имеет место

тогда, когда неудобно хранить промежуточные результаты, например, в случае использования калькуляторов. Суммирование во внутреннем цикле можно выполнять без отдельной записи промежуточных частичных сумм. Это позволяет существенно экономить время и, кроме того, увеличить надежность вычислений, поскольку запись чисел от руки приводит к появлению ошибок.

Второе преимущество проявляется для вычислительных машин, обладающих сравнительно быстрой арифметикой с расширенной точностью или особенно быстрой операцией скалярного произведения. Пусть вычисление сумм

$$\sum_{i=1}^{k-1} a_{ij}a_{ik}, \qquad \sum_{i=1}^{k-1} a_{ki}a_{ij}$$

с удвоенной точностью относительно дешево и требует только на 5 процентов больше времени, чем вычисление с обычной точностью. Было бы правдоподобным предположить, что использование арифметики с удвоенной точностью позволяет исключить влияние большинства ошибок округления в вычислениях, и детальный анализ, который здесь опускается, доказывает, что это имеет место. Раньше существовали вычислительные машины, которые обладали быстрой арифметикой с расширенной точностью для вычисления такого рода сумм, и алгоритм Краута был для них весьма привлекателен. Весьма вероятно, что это свойство снова станет широко распространенным. Некоторые современные вычислительные машины допускают особенно эффективное вычисление вышеупомянутых сумм в режиме арифметики с обычной точностью, и алгоритм Краута выгоден для таких машин.

Наконец, отметим, что существует специальная модификация исключения Гаусса для положительно определенных симметричных матриц. Если A — симметричная матрица, то представляется вероятным, что при решении системы $\mathbf{Ax}=\mathbf{b}$ можно избежать половины трудозатрат по сравнению с решением системы с матрицей общего вида. Такой экономии достигает разложение Холесского $L^{-1}A=L^{T}$, для которого вычисление матрицы L может быть выполнено за половину операций, требуемых для LU-разложения.

Существо метода видно из доказательства возможности разложения методом математической индукции по размеру матрицы. Рассмотрим сначала случай 2×2-матрицы

$$A=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \qquad L=\begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix}.$$

Потребуем, чтобы $A=LL^{T}$, или, что то же самое,

$$a_{11}=l_{11}^2, \qquad a_{12}=l_{11}l_{21},$$
$$a_{21}=l_{21}l_{11}, \qquad a_{22}=l_{21}^2+l_{22}^2.$$

В силу положительной определенности матрицы A, элемент $a_{11}>0$ и извлечение квадратного корня $l_{11}=\sqrt{a_{11}}$ возможно. Второй элемент определяется из уравнения для a_{21} (который равен a_{12}, так как A — симметричная матрица). Наконец, l_{22} определяется из равенства

$$l_{22}=\sqrt{a_{22}-l_{21}^2}=\sqrt{a_{22}-a_{12}^2/a_{11}}\ .$$

Покажем, что извлечение корня возможно. В соотношении $\mathbf{x}^T\mathbf{A}\mathbf{x}>0$ возьмем в качестве $\mathbf{x}$ вектор $(a_{12}, -a_{11})^T$. Тогда $\mathbf{x}^T\mathbf{A}\mathbf{x}=a_{11}^2a_{22}-a_{12}^2a_{11}>0$, что эквивалентно $a_{22}-a_{12}^2/a_{11}>0$, и извлечение квадратного корня в формуле для l_{22} возможно.

В общем случае предположим, что возможно разложение $(m-1)\times(m-1)$-матрицы A в виде LL^T. Представим $m\times m$-матрицу A′ таким образом:

$$A'=\begin{pmatrix} A & \mathbf{y} \\ \mathbf{y}^T & a_{mm}\end{pmatrix},\qquad L'=\begin{pmatrix} L & 0 \\ \mathbf{w} & l_{mm}\end{pmatrix},$$

где $\mathbf{y}$ и w — $(m-1)$-векторы. Потребуем, чтобы

$$A=LL^T,\qquad \mathbf{y}=L\mathbf{w},$$
$$\mathbf{y}^T=\mathbf{w}^TL^T,\qquad a_{mm}=l_{mm}^2+\mathbf{w}^T\mathbf{w}.$$

По предположению математической индукции можно найти L, такую, что $LL^T=A$. Решая уравнение $\mathbf{y}=L\mathbf{w}$, находим $m-1$ компонент вектора $\mathbf{w}$. После этого l_{mm} определяется из формулы

$$l_{mm}=\sqrt{a_{mm}-\mathbf{w}^T\mathbf{w}}\ .$$

Покажем по аналогии с доказательством для 2×2-матрицы, что выражение под квадратным корнем положительно. В качестве вектора $\mathbf{x}$ возьмем $(A^{-1}\mathbf{y}, -1)^T$ и вычислим $\mathbf{x}^TA'\mathbf{x}$, обозначив для удобства $z=A^{-1}\mathbf{y}$:

$$\begin{aligned}\mathbf{x}^TA'\mathbf{x} &= \mathbf{z}^TA\mathbf{z}-2\mathbf{z}^T\mathbf{y}+a_{mm}\\ &= -\mathbf{z}^T\mathbf{y}+a_{mm}=a_{mm}-\mathbf{y}^TA^{-1}\mathbf{y}\\ &= a_{mm}-\mathbf{y}^T(LL^T)^{-1}\mathbf{y}=a_{mm}-(L^{-1}\mathbf{y})^T(L^{-1}\mathbf{y})\\ &= a_{mm}-\mathbf{w}^T\mathbf{w}\end{aligned}$$

Поскольку $\mathbf{x}^TA'\mathbf{x}>0$, то $a_{mm}-\mathbf{w}^T\mathbf{w}>0$, и доказательство по индукции закончено.

Это доказательство фактически показывает возможность симметричного разложения; алгоритм 5.3 разложения Холесского полностью соответствует ходу доказательства.

Алгоритм 5.3: Разложение Холесского симметричной матрицы

ЦИКЛ ПО СТОЛБЦАМ A

For $k=1$ to n do

ВЫЧИСЛЕНИЕ ОЧЕРЕДНОГО ВЕКТОРА-СТРОКИ, КРОМЕ ДИАГОНАЛЬНОГО ЭЛЕМЕНТА

For $i=1$ to $k-1$ do

$$a_{ki}=\left(a_{ki}-\sum_{j=1}^{i-1} a_{ij}a_{kj}\right)\Big/a_{ii}$$

конец цикла по i

ВЫЧИСЛЕНИЕ ДИАГОНАЛЬНОГО ЭЛЕМЕНТА

$$a_{kk}=\sqrt{a_{kk}-\sum_{j=1}^{k-1} a_{kj}^2}$$

конец цикла по k

ТЕПЕРЬ МНОЖИТЕЛЬ L РАСПОЛОЖЕН В НИЖНЕМ ТРЕУГОЛЬНИКЕ МАТРИЦЫ A

Алгоритм 5.3 может быть сделан более устойчивым заменой оператора, содержащего квадратный корень, следующим оператором:

$$t=a_{kk}-\sum_{j=1}^{k-1} a_{kj}^2,$$

$$\text{If } t>0 \text{ then } a_{kk}=\sqrt{t},$$
$$\text{else } a_{kk}=0.$$

Это позволит избежать останова программы, когда элемент близок к нулю или точно равен нулю и ошибки округления случайно делают t отрицательным.

5.В.4. Подсчет числа операций

Трудозатраты при решении системы $\mathbf{Ax}=\mathbf{b}$ с помощью исключения Гаусса могут быть оценены непосредственно путем подсчета арифметических операций. Это делается следующим образом: матрица A на k-м шаге имеет вид как на рис. 5.2.

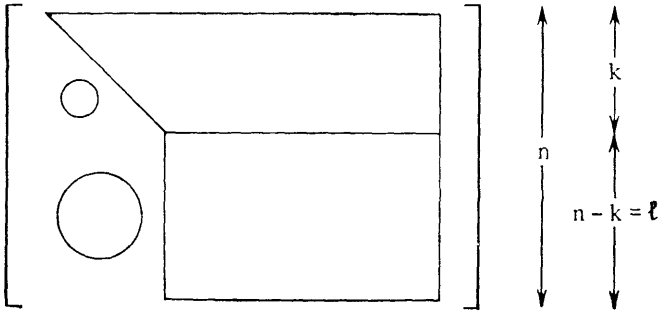


Рис. 5 2

Трудозатраты следующего шага исключения:

1. Множители: $m_i = a_{ik}/a_{kk}$, $i = k+1$ до n.
2. Исключение: $a_{ij} = a_{ij} - m_i * a_{kj}$, $i = k+1$ до n, $j = k+1$ до n.
3. Правая часть: $b_i = b_i - m_i * b_k$, $i = k+1$ до n.

На первом этапе этого шага исключения имеем l делений (или 1 деление и l умножений); на втором этапе — l*[l сложений+l умножений]; на третьем этапе — l*[1 сложение+1 умножение].

Введем обозначения: A — сложение, M — умножение, D — деление. Просуммируем для всех значений k число операций на каждом этапе.

$$\sum_{k=1}^{n-1} (\text{операции первого этапа}) = \left(\sum_{k=1}^{n-1} l\right) D = \left(\sum_{k=1}^{n-1} n-k\right) D$$
$$= \left(\sum_{i=1}^{n-1} i\right) D = \frac{(n-1)(n-2)}{2} D.$$

Заменив деления умножениями, получим

$$(n-1) D + \frac{(n-1)(n-2)}{2} M,$$

$$\sum_{k=1}^{n-1} (\text{операции второго этапа}) = (A+M) \sum_{k=1}^{n-1} l^2 = (A+M) \sum_{i=1}^{n-1} i^2$$
$$= \frac{n(n-1)(2n-1)}{6} (A+M).$$

Трудозатраты на разложение матрицы есть сумма двух последних выражений:

$$\left(\frac{n^3}{3} + \text{члены меньшего порядка}\right)(A+M).$$

Число операций для вычисления правой части равно

$$(A+M) \sum_{k=1}^{n-1} l = \frac{(n-1)(n-2)}{2} (A+M),$$

а число операций для обратной подстановки равно

$$\sum_{k=n}^{1} [l(A+M) + 1D] = \frac{n(n+1)}{2} (A+M) + nD.$$

Общее число операций, требуемых для получения правой части (прямой ход) и для обратной подстановки, равно

$$(n^2 + \text{члены меньшего порядка})(A+M).$$

Для получения LU-разложения исключением Гаусса требуется порядка $n^3/3$ операций сложения и умножения.

Процедура определения числа операций относится к анализу *вычислительной сложности*. При таком анализе оцениваются тру-

дозатраты на вычисление чего-либо (в нашем случае на решение системы $\mathbf{Ax}=\mathbf{b}$) в терминах основных машинных операций (в нашем случае арифметических операций). Очевидный и важный вопрос: является ли исключение Гаусса наиболее быстрым способом решения $\mathbf{Ax}=\mathbf{b}$? Для достаточно больших значений n применение алгоритма Штрассена приводит к методу, имеющему меньшее число операций. Он требует порядка $n^{2.7}$ сложений и умножений. В настоящее время проводится теоретическое изучение и экспериментирование по вопросу о том, насколько выгоден этот новый алгоритм на практике. Предварительные результаты показывают, что он становится более эффективным для n, больших 250.

Задачи разд. 5.В

1. Реализовать алгоритм 5.1 в виде подпрограммы на языке Фортран. Дать обоснования выбора последовательности параметров обращения. Объяснить выбор значения машинно-зависимой константы BUFFER.

2. Реализовать алгоритм 5.1 в виде трех подпрограмм на языке Фортран. В подпрограмме FACTOR LU-разложение формируется на месте матрицы A без выбора ведущего элемента. Подпрограмма FSUB использует результаты работы подпрограммы FACTOR для соответствующих преобразований правой части системы. Подпрограмма BSUB выполняет обратную подстановку. Дать полное описание последовательностей параметров обращений к подпрограммам. Составить набросок программы на Фортране, использующей эти три подпрограммы для решения следующих задач:

— решить две системы с матрицами A и B размеров 10×10 и 20×20;

— решить семь систем $\mathbf{Ax}_i=\mathbf{b}_i$ с векторами правых частей, расположенными в массиве B (10, 7);

— решить 16 систем $\mathbf{Dy}_i=\mathbf{e}_i$ с векторами правых частей, расположенными в массиве E (20, 16).

3. Рассмотреть две библиотечные подпрограммы FACTOR (A) и SOLVE (A, X, B), в которых для простоты различные аргументы, специфицирующие размеры A или системы, опущены. Подпрограмма FACTOR вычисляет треугольное разложение на месте матрицы A. Подпрограмма SOLVE решает систему $\mathbf{Ax}=\mathbf{b}$ в предположении, что матрица A представлена своим треугольным разложением.

Пусть требуется вычислить вектор y по формуле

$$\mathbf{y}=\mathbf{B}\,(2\mathbf{C}+\mathbf{D}^{-1})\mathbf{x}+(\mathbf{I}-\mathbf{D}^{-1}\mathbf{L})\mathbf{z}.$$

Составить набросок программы на Фортране, использующей FACTOR и SOLVE для вычисления y без явного обращения матриц. Для краткости можно использовать в операциях векторной и матричной арифметик операторы типа «прибавить вектор V к вектору S» или «умножить вектор V на матрицу M». Пояснить использование переменных и способы хранения информации.

4. На основе алгоритма 5.1 составить эффективный алгоритм приведения трехдиагональной матрицы к верхней треугольной форме без выбора ведущего элемента.

5. Составить набросок подпрограммы на Фортране, реализующей алгоритм задачи 4 в модификации, рассчитанной на экономию машинной памяти. Описать в деталях структуры данных и последовательность параметров обращений.

6. Обобщить алгоритм и программы задач 4 и 5 на случай ленточных матриц с шириной ленты K (без выбора ведущего элемента).

7. Включить частичный выбор ведущего элемента в формулировку задачи 6. Сколько дополнительной памяти это потребует?

8. На основе алгоритма 5.1 составить эффективный алгоритм решения линейной системы $\mathbf{Ax}=\mathbf{b}$ без выбора ведущего элемента, где A — симметричная матрица, заданная своим верхним треугольником.

9. Для алгоритма 5.2 составить диаграмму, иллюстрирующую последовательность обработки элементов матрицы A.

10. На основе алгоритма 5.2 составить эффективный алгоритм приведения трехдиагональной матрицы к верхней треугольной форме без выбора ведущего элемента.

11. Составьте набросок подпрограммы на Фортране, реализующей алгоритм задачи 10 в модификации, рассчитанной на экономию машинной памяти. Описать в деталях структуры данных и последовательность параметров обращений.

12. Обобщить алгоритм и программы задач 10 и 11 на случай ленточных матриц с шириной ленты K (без выбора ведущего элемента).

13. Включить частичный выбор ведущего элемента в формулировку задачи 12. Сколько дополнительной памяти это потребует?

14. На основе алгоритма 5.3 составить эффективный алгоритм приведения симметричной ленточной матрицы к верхней треугольной форме.

15. Составить набросок подпрограммы на Фортране, реализующей алгоритм задачи 14 в модификации, рассчитанной на экономию машинной памяти. Описать в деталях структуры данных и последовательность параметров обращений.

16. Составить четыре модификации алгоритма 5.1: (а) элементы матрицы A сохраняются, (б) вместо явной перестановки строк при выборе ведущего элемента используются указатели (это пример применения косвенной адресации или индексированных индексов); (в) решение матричного уравнения $AX=B$, где B — $n\times m$-матрица; (г) вычисление A^{-1}.

17. Объяснить соотношение между следующими тремя методами решения $\mathbf{Ax}=\mathbf{b}$: исключение Гаусса (LU-разложение), модификация Краута и разложение Холесского.

18. «x-матрица» имеет структуру ненулевых элементов «x», показанную на примерах «x-матриц» пятого и шестого порядков:

$$\begin{pmatrix} x & x & x & x & x \\ 0 & x & x & x & 0 \\ 0 & 0 & x & 0 & 0 \\ 0 & x & x & x & 0 \\ x & x & x & x & x \end{pmatrix} \qquad \begin{pmatrix} x & x & x & x & x & x \\ 0 & x & x & x & x & 0 \\ 0 & 0 & x & x & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 \\ 0 & x & x & x & x & 0 \\ x & x & x & x & x & x \end{pmatrix}$$

(а) На основе исключения Гаусса составить алгоритм приведения к этой форме матрицы общего вида посредством решения последовательности систем второго порядка.

(б) Показать, как может быть осуществлена обратная подстановка для «x-матрицы», полученной описанным в (а) способом;

(в) Подсчитать число операций для (а) и (б) и сравнить с обычным исключением Гаусса (без выбора ведущего элемента).

19. Пусть вектор $\mathbf{e}_i=(0, 0, \ldots, 0, 1, 0, \ldots, 0)^T$ имеет 1 на i-м месте, так что $\mathbf{e}_i$ представляет собой i-й столбец единичной матрицы. Столбцы $\mathbf{x}_i$ матрицы A^{-1}

могут быть определены решением систем $Ax_i=e_i$ для i=1, 2 , . . . , N. Пусть с помощью первой части алгоритма 5.1 матрица A уже представлена своим треугольным разложением. На основе той части алгоритма 5.1, которая осуществляет обратную подстановку, составить эффективный алгоритм вычисления x_i. При составлении алгоритма учесть специальный вид правых частей e_i.

20. Подсчитать число операций для алгоритма задачи 19 и оценить экономию трудозатрат по сравнению с решением систем $Ax_i=b_i$ для i=1, 2, . . . , N, где b_i — произвольные векторы правых частей.

Глава 6

ОБЩИЕ ПРОБЛЕМЫ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Прежде чем перейти к исследованию математического обеспечения, предназначенного для задач линейной алгебры, рассмотрим в целом «философию», цели и противоречивые тенденции в разработке математического обеспечения.

6.А. ЦЕЛИ И СРЕДСТВА

Выделим четыре аспекта, лежащие в основе разработки математического обеспечения:

1. *Общие цели*: создать более мощные вычислительные средства для решения задач пользователя и повышения используемого «языкового уровня».

2. *Средство повышения языкового уровня=мощные операции*: основная слабость современных языков (Фортран и др.) — нехватка мощных операций.

3. *Необходимые операции*: в каждой области вычислений существуют определенные типовые задачи, которые должны быть реализованы и представлены в виде языковых операций.

4. *В математическом обеспечении должны быть сконцентрированы знания экспертов*: каждая из таких задач должна быть тщательно исследована, и нужны эксперты, имеющие глубокие знания о том, как реализовать решение этих задач наилучшим образом. Эти эксперты должны разрабатывать математическое обеспечение и вкладывать в него свои знания для того, чтобы в дальнейшем все могли воспользоваться ими в качестве стандартных операций языков программирования.

6.А.1. Две альтернативы

Очевидный подход состоит в разработке нового языка программирования, имеющего много встроенных операций. Например, математический язык программирования общего назначения должен быть по крайней мере на уровне дифференциального и интегрального исчисления, а также матричной алгебры. Это позволило бы научным и инженерным работникам программировать, свободно

пользуясь обычным языком математики. Программы должны допускать применение операторов вида:

```
FUNCTION ARRAY: F(X,I,J)
POLYNOMIAL: F(X,I,J) FOR I,J≤10
A(I,J) = INTEGRAL (F(X,I,J), X = 1,2 TO 5.) FOR I,J = 1 TO 6
SOLVE A·X=B FOR X
FMIN(I) = MIN(MIN(F(X,I,J),X = 1,2 TO 3), J = 1 TO 6)
```

Такие языки создаются с начала 60-х годов на основе или численных, или символьных методов реализации операций. Практика показала, что создание такого рода языков возможно, хотя они, естественно, требуют значительно больше вычислительных ресурсов, чем Фортран, Алгол, Паскаль и др.

У современных языков общего назначения, по-видимому, нет хорошей перспективы. Преимущества, достигаемые за счет лучших структур управления, более гибких типов данных и более мощных операций, вероятнее всего, не смогут перевесить противодействующие факторы. Одним из таких факторов является возникновение значительных трудностей при внедрении новых универсальных языков. Трудно надеяться на то, что новый язык будет широко использоваться, несмотря на его преимущества. Другая проблема состоит в том, что специалисты в области развития языков программирования проявляют мало интереса к реализации новых операций вообще и математических операций в частности. Они рассчитывают на то, что достаточно будет улучшить структуры управления и типы данных наряду с культурой программирования для того, чтобы обойти узкие места. Однако мощные операции сокращают программы, что позволяет значительно упростить программирование и повысить производительность труда.

В то время как перспективы языков общего назначения, обладающих мощными операциями, туманны, во многих областях применения вычислительных машин появились языки специального назначения (или проблемно-ориентированные языки). Как правило, эти языки и системы разрабатываются людьми, не являющимися специалистами в области системного программирования, и потому часто не обладают той степенью изящества, которая привлекает системных программистов. Однако эти разработки подходят к решению центральной проблемы, которая состоит в том, чтобы дать возможность пользователю программировать в терминах своей предметной области. Язык, на котором хотел бы работать пользователь, должен содержать подходящие существительные (структуры данных) и глаголы (операции), а также много прилагательных. Такой подход наиболее развит в математической статистике, для которой создано несколько хороших систем. Некоторые из них позволяют пользователям, которые не знают практически ничего о программировании (с точки зрения

системных программистов), проводить обширные и осмысленные вычисления. Часто эти пользователи также мало знают о методах статистических вычислений.

Можно ожидать, что каждая область знаний будет иметь свою собственную систему, и в конце концов основная масса вычислений будет проводиться скорее всего на основе этих систем, а не посредством прямого использования языков общего назначения. Так же как статистические системы, завоевавшие широкую аудиторию пользователей, не являющихся специалистами в области математической статистики, большинство из таких систем найдет применение в тех областях знаний, для которых они были созданы. Строительство представляет собой пример интенсивного потребителя вычислительных мощностей, для которого проблемно-ориентированные системы играют большую роль. Ранним и характерным примером таких систем является система NASTRAN. Она насчитывает 300 000 операторов языка Фортран, имеет руководство для пользователя объемом в 2 000 с и введение в 1 200 с. Стоимость разработки системы превышает 10 000 000 долларов. Некоторые ее средства выглядят «ужасными», однако ею широко пользуются, поскольку она представляет собой шаг в правильном направлении: она позволила освободить инженеров от кодирования на Фортране и предоставила им более мощные средства, разработанные специалистами в области применения вычислительной техники в строительной индустрии.

Второй подход состоит в обеспечении доступа к мощным операциям через библиотеки программ. Если не разработан хороший проблемно-ориентированный язык (в большинстве случаев дело обстоит именно так), то в этих условиях библиотечные процедуры, состыкованные с языками общего назначения, представляют собой лучший способ использования знаний экспертов. Несмотря на то что проблемно-ориентированные языки могут стать «веянием будущего», основной упор в этой книге будет сделан на библиотеки как на подход, который возможен в настоящее время. В приложении представлены фрагменты из нескольких библиотек, содержащих программы матричного исчисления. Читатель может рассматривать их как типичные примеры состава и документации библиотек.

6.Б. ЦЕЛИ, ПРЕСЛЕДУЕМЫЕ ПРИ РАЗРАБОТКЕ АЛГОРИТМОВ

Ядром любой библиотечной подпрограммы является лежащий в ее основе алгоритм. Под алгоритмами мы понимаем вычислительные процедуры, основанные на известных процедурах: исключение Гаусса, правило Симпсона, метод Ньютона и т. д. Детали алгоритма могут оказывать значительное влияние на его примени-

мость, и цели, преследуемые при конструировании алгоритма, состоят в обеспечении (1) скорости; (2) экономии памяти; (3) надежности; (4) точности; (5) простоты и (6) робастности. Эти термины ясны сами по себе, за исключением, быть может, «надежности» и «робастности». Для некоторых задач существуют алгоритмы, которые всегда работают, или ситуации, для которых возможные неудачи столь маловероятны, что их можно игнорировать. Однако многие математические задачи алгоритмически неразрешимы [1]). Это означает, что для любого алгоритма найдется задача из той же предметной области, для которой этот алгоритм совершенно неприменим. Решение нелинейных уравнений и дифференциальных уравнений, вычисление интегралов — примеры такого рода задач. Поэтому надежность определяет вероятность неудачи алгоритма, которая состоит или в отсутствии результата (что не так плохо), или в получении ошибочного результата (что очень плохо). Надежность обычно измеряется некоторой комбинацией вероятности получения правильных результатов и вероятности получения неверного результата без его выявления.

Робастность тесно соотносится с надежностью, однако дополнительно заключает в себе знание того, где и по какой причине алгоритм потерпел неудачу. Можно сказать, что алгоритм является робастным, если все его неудачи не являются неожиданными и могут быть идентифицированы. Поведение робастного алгоритма непрерывно зависит от задачи; это означает, что нельзя найти две близкие задачи, одну из которых алгоритм совершенно не в состоянии решить, а другую — решает легко. Неудачи алгоритма идентифицируются, например, тогда, когда время счета устойчиво растет или оценка точности устойчиво падает, если задача становится более трудной. Робастный алгоритм не порождает неверных результатов (например, $\sqrt{-0.01}$ в вещественной арифметике), прерывающих вычисление в некоторой промежуточной точке. Обратите внимание на примечание к алгоритму 5.3, указывающее на одну из мер, принятие которой делает алгоритм робастным.

6.В. БИБЛИОТЕКИ МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Перечислим качества, которыми должна обладать библиотека программ:

Мощность
Гибкость
Легкость в использовании
Хорошие алгоритмы
Портабельность
Легкость модификации
Надежность в работе

[1]) То есть не существует универсальных алгоритмов решения этих **задач.**— *Прим. перев.*

Хорошая библиотека программ содержит хорошие алгоритмы (со всеми их достоинствами), реализованные в виде программ так, чтобы обеспечить и другие качества. Эти качества в какой-то степени очевидны, однако уместно пояснить их несколькими словами.

Мощность — главная цель библиотеки: она должна содержать программы, решающие трудные задачи широкого круга. Основным, но не единственным определяющим фактором этого качества является мощность лежащего в основе программы алгоритма.

Требование гибкости обусловлено тем обстоятельством, что даже стандартные задачи имеют вариации. Например, для решения системы линейных уравнений могут быть даны следующие формулировки:

Решить $A\mathbf{x}=\mathbf{b}$.
Решить $AX=B$ (несколько правых частей и решений).
Решить $AX=I$ (X — обратная к A матрица).
Решить $A\mathbf{x}=\mathbf{b}$ (где A — ленточная матрица).
Решить $A\mathbf{x}=\mathbf{b}$ (где a_{ij} заданы формулой).

Гибкость увеличивает возможности программы, однако ухудшает другие качества.

Легкость в использовании является существенным, но трудно достижимым качеством Существуют примеры программ, широко применяемых благодаря легкости в использовании, хотя они ненадежны или неэффективны. Программа *портабельна*, если она может работать на различных машинах без изменений или с небольшими изменениями. В задачах I и II (см. ниже) приведены примеры программ, не обладающих качеством портабельности. Очевидно, что следует избегать дублирования работы при производстве дорогостоящего математического обеспечения. Стоимость производства одного оператора хорошей программы составляет, минимум, от 15 до 20 долларов, однако и для плохой программы могут иметь место те же цифры. Эти цифры приведены в предположении, что при разработке алгоритма не возникло каких-либо технических затруднений. Некоторые специалисты используют термин «транспортабельность» для программ, требующих небольшого числа изменений при переносе на другие машины, и прибегают к термину «портабельность» в тех случаях, когда программа может работать на различных машинах без всяких изменений.

Легкость модификации необходима для сопровождения программы, поскольку любая библиотечная программа, какой бы степенью гибкости она ни обладала, не может удовлетворять всем требованиям пользователей. *Сопровождение* состоит в обеспечении соответствия программы изменениям в языках программирования, компиляторах, операционных системах, стандартах документирования и т. д., а также в повышении удобства доступа к программе

или ее надежности, устранении мелких неполадок и внесении изменений для повышения эффективности.

Надежность в работе означает не только надежность и робастность, но также и общую защиту от сбоев программы. Многие библиотечные программы глубоко погружаются в главную программу или систему и тем самым становятся «невидимыми» для пользователя. Никто не захочет получить сообщение вида ОСТАНОВ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА В ЯЧЕЙКЕ 094637, который, как оказывается, произошел в подпрограмме ТАИНСТВ, причем пользователь о ней никогда не слышал.

6.В.1. Противоречивость целей в разработке математического обеспечения

Цели, преследуемые при разработке хороших библиотек алгоритмов и программ, иногда оказываются несовместимыми. Укажем пять наиболее типичных противоречий:

1. Гибкость и легкость в использовании.
2. Портабельность и эффективность.
3. Надежность (робастность) и эффективность.
4. Особенность человеческого восприятия информации и лаконичность описания программы.
5. Ограничения языков программирования и легкость в использовании программ.

На самом деле почти все остальные требования вступают в противоречие с эффективностью, выраженной через машинное время. В основе нашего подхода лежит эффективность процесса решения задачи в целом. Тем самым предпочтение отдается увеличению этой общей эффективности по сравнению с эффективностью, выраженной затратами машинного времени.

Бóльшая гибкость означает бóльшую сложность, а пользователю нужны простые программы. Обычно гибкость достигается введением дополнительного числа параметров и переключателей для регулирования режимов работы программы. Пользователю приходится присваивать параметрам значения, даже если они не относятся к существу решаемой задачи. Отметим, что список параметров может содержать параметры, не относящиеся ни к обеспечению гибкости, ни к решаемой задаче. Обычно это параметры алгоритма, значения которых не зафиксированы автором программы, и проблема их выбора перекладывается на пользователя. Иногда пользователь слабо или совсем не представляет себе, как выбирать подходящие значения этих параметров. Разумная тактика для выхода из этого противоречия состоит в том, чтобы иметь программы максимальной гибкости (и длины списка передаваемых параметров) и некоторое количество более простых интерфейсных

программ, устанавливающих значения различных параметров по умолчанию и выполняющих другие сервисные функции.

Легко видеть, что требования портабельности, надежности и др. вступают в противоречие с эффективностью. Программа, составленная для решения частной задачи на конкретной машине, может не иметь проверок на надежность, поскольку неприятности при счете, которые *не возникают*, можно игнорировать, а для тех ошибок, которые *возникают*, можно предусмотреть те или иные дополнительные предосторожности.

Более того, особенности конкретных машин, компиляторов, операционных систем и решаемой задачи могут быть использованы для увеличения эффективности. Вопрос состоит не в том, чтобы пожертвовать эффективностью, а в том, в какой степени следует ею пожертвовать, чтобы улучшить другие показатели программы.

Пользователь может быстро убедиться в том, что описания большинства библиотечных программ сбивают с толку, трудно читаемы, двусмысленны и часто непонятны. Это справедливо даже в тех случаях, когда большие усилия были потрачены на то, чтобы сделать описания ясными, завершенными и легкими для понимания. Допускаются ошибки за ошибками в силу того, что пользователь не учитывает различные тонкости и не только тонкости работы библиотечной программы. Некоторые специалисты предпочитают потратить четыре дня на составление своей собственной программы, нежели потратить половину дня на освоение того, как должным образом использовать библиотечную программу.

Причина этой трудности до конца не ясна. Идеальное описание программы выделяет все, что следует знать о ее использовании, в короткой и ясной форме. Такие описания имеют высокую плотность информации, большая часть которой не знакома пользователю, и похоже, что человек не в состоянии воспринять всю информацию после первого (второго или третьего) чтения. Пользователь ухватывает общую идею и затем приходит к заключению, обычно неправильному, относительно специфических деталей программы. Процесс правильного восприятия всех деталей обычно включает в себя несколько запусков программы и немало разочарований. В описаниях программ следует проявлять осторожность при выборе формулировок и обозначений и т. д., чтобы увеличить вероятность того, что пользователь придет к правильным заключениям.

Рассмотренные противоречия естественны, но существуют другие противоречия, которые продиктованы реальностью проведения вычислений. Некоторые из них обусловлены тем, что языковые ограничения накладывают дополнительные сложности. Язык Фортран часто требует введения параметров, которые необходимы только для того, чтобы обеспечить работу программы [1]). Эти пара-

[1]) Например, рабочие массивы и их размерности.— *Прим. перев.*

метры неестественны для пользователя и чаще вызывают ошибки, чем другие типы параметров программы. Следует изучить примеры из библиотек, приведенные в приложении, чтобы понять, какие компромиссные решения выбрали разработчики этих библиотек для разрешения указанных противоречий.

Задачи разд. 6.Б и 6.В

1. Каковы цели разработки алгоритмов решения линейных уравнений?

2. Каковы цели разработки библиотечных программ решения линейных уравнений?

3. Дать список параметров хорошей библиотечной подпрограммы на Фортране для решения системы $Ax=b$. Явно обосновать причины выбора параметров. Указать приложения, для которых данный список параметров имеет недостатки по сравнению с альтернативными.

4. *Кратко* сформулировать сильные и слабые стороны двух процессов обучения: (а) как решать задачи или разрабатывать алгоритмы для решения задач; (б) как использовать математическое обеспечение для решения задач.

5. Выбрать описания двух библиотечных подпрограмм и критически рассмотреть их. Обратить внимание на следующее:

(а) Все ли переменные ясно определены и даны ли им хорошие имена?
(б) Имеет ли подпрограмма параметры, назначение которых не ясно?
(в) Дан ли пример использования подпрограммы?
(г) Дана ли ссылка на исходный алгоритм?
(д) Является ли типичной ситуация, когда пользователи должны модифицировать эти подпрограммы перед их использованием?
(е) Тратит ли подпрограмма впустую память на небольших задачах?
(ж) Легко ли может быть получен текст программы? Если да, то насколько хорошо она документирована?
(з) Должны ли входные данные быть представлены в неестественной форме?

6. Рассмотреть противоречие между обеспечением эффективности библиотечной подпрограммы и легкостью ее модификации.

7. Дать три примера, когда в связи с внешними факторами требуется сопровождение библиотечной подпрограммы. Пояснить кратко каждый пример.

8. Существуют четыре подхода к выявлению ошибок, обнаруживаемых в процессе работы библиотечной подпрограммы, глубоко погруженной в целевую программу:
(а) прервать вычисления и распечатать хорошее пояснение;
(б) прервать исполнение библиотечной подпрограммы (но не всего задания) и вернуться в вызывающую программу с признаком, что имела место ошибка;
(в) исправить ошибку наиболее подходящим образом и распечатать поясняющее сообщение о том, что произошло;
(г) исправить ошибку наиболее подходящим образом и вернуться в вызывающую программу с признаком, что ошибка имела место.
Рассмотреть сильные и слабые стороны каждой из этих альтернатив. Рассмотреть случай, когда при использовании проблемно-ориентированного языка высокого уровня для решения статистической задачи была сгенерирована система линейных уравнений и было обнаружено, что (1) матрица системы вырождена; (2) матрица коэффициентов идентична нулевой и вектор правой части представляет собой единичный вектор; (3) порядок системы $n=-2$.

9. Найти пример реальной библиотечной подпрограммы, которая имеет длин-

ный список параметров, и составить подпрограмму, предназначенную для решения более простой задачи посредством обращения к выбранной подпрограмме с некоторыми параметрами, определенными по умолчанию.

10. Выбрать подпрограмму решения системы линейных уравнений и проверить степень ее робастности заданием неверных или бессмысленных параметров, например:

(а) порядок матрицы равен 1, 0, —1 или вещественному числу;
(б) идентификатор матрицы является простой переменной;
(в) задать меньше параметров чем необходимо;
(г) в вызывающей программе массив, содержащий матрицу, описать как INTEGER;
(д) задать матрицу, идентичную нулевой;
(е) задать вектор правой части с ошибками вида (а), (б) и (г);
(ж) допустить различные несогласованности в описании размерностей.

11. Следующая программа на Фортране имеет по крайней мере 18 признаков, которые делают ее непортабельной. Выявить эти признаки и рассмотреть, какие из них приведут к появлению ошибок или диагностических сообщений при трансляции программы на машине, которой вы пользуетесь.

```
      INTEGER CLUE, CHARACT(7), POINT, BLANK
      REAL DIGITS (10)
      DATA (DIGITS(I), I = 1,10)/0,1,2,3,4,5,6,7,8,9/
      DATA MORE, BLANK/4HMORE, 1H /
      POINT = 1H.
    1 READ 5, CLUE, N, (CHARACT(I),I = 1,N)
    5 FORMAT (7A1)
      PI = 3 141592654
      DO 10 I = 1,N
   10 IF(CHARACT(I) NE BLANK)                GO TO 20
   12 PRINT 15, CHARACT, POINT
                                             GO TO 50
   15 FORMAT('0 ОШИБКА ***'7A2,A3)
   20 VAL = 0
      K = 1
      DO 30 J = 1,N+1-I
        DO 25 L = 1,10
          IF (CHARACT(N+1-J) NE DIGIT(L))    GO TO 25
                                             GO TO 26
   25   CONTINUE
                                             GO TO 12
   26   VAL = L * 10 **K+VAL
   30   K = K+1
      ANS = VAL*PI
      PRINT 40, ANS
   40 FORMAT(20X9H  ОТВЕТ = E40.11)
   50 IF(CLUE EQ MORE)                       GO TO 1
      STOP
      END
```

6.Г. РАЗРАБОТКА ИНТЕРФЕЙСА ДЛЯ ПРОГРАММЫ РЕШЕНИЯ ЛИНЕЙНЫХ УРАВНЕНИЙ

Рассмотрим несколько подробнее вопросы проектирования библиотечной подпрограммы решения систем линейных уравнений. Даже в этой относительно простой, стандартной задаче будут необходимы некоторые компромиссы, типичные при проектировании математического обеспечения.

Для решения системы $Ax=b$ естественным представляется

следующее обращение к подпрограмме на Фортране:

CALL SOLVE (A, X, B).

Однако такое обращение не сработает, поскольку отсутствует порядок системы N; поэтому оно должно иметь вид

CALL SOLVE (A, X, B, N).

Некоторые из недостатков такого способа:

1. Исходные элементы матрицы A уничтожаются, поскольку для подпрограммы SOLVE не выделен рабочий участок памяти для хранения множителей треугольного разложения.
2. Пользователь не предупреждается о случаях, когда матрица A вырождена или подпрограмма SOLVE не в состоянии решить задачу.
3. Не заданы размерности фортранных массивов для A, X и B.
4. Если решаются две системы $A\mathbf{x}_1=\mathbf{b}_1$ и $A\mathbf{x}_2=\mathbf{b}_2$ с той же самой матрицей, то разложение A производится дважды.
5. Вычисление обратной матрицы A^{-1} с помощью этой подпрограммы неэффективно.

Таким образом, несмотря на простоту этого способа, подпрограмма SOLVE не обладает гибкостью, широкими возможностями и надежностью, и ей присущи некоторые недостатки, связанные с природой Фортрана.

Следующие два варианта имеют меньше недостатков.

Подпрограмма решения линейных уравнений — Вариант 1

```
C БЕЗ РАБОЧЕЙ ПАМЯТИ. A-УНИЧТОЖАЕТСЯ
  CALL SOLVE (A,X,B,N,I,J,K,L,M,M1)
  ---
  SUBROUTINE SOLVE (A,X,B,N,I,J,K,L,M,M1)
  DIMENSION A(I,J), X(I,K), B(I,K)
  ---
C I = ЧИСЛО СТРОК, ОБЪЯВЛЕННЫХ В DIMENSION ДЛЯ A,X,B
C J = ЧИСЛО СТОЛБЦОВ, ОБЪЯВЛЕННЫХ В DIMENSION ДЛЯ A
C K = ЧИСЛО СТОЛБЦОВ, ОБЪЯВЛЕННЫХ В DIMENSION ДЛЯ X,B
C L = ЧИСЛО ПРАВЫХ ЧАСТЕЙ
C M = ПЕРЕКЛЮЧАТЕЛЬ ДЛЯ ВЫЧИСЛЕНИЯ A**-1
C M1 = ПРИЗНАК ВЫРОЖДЕННОСТИ A
```

Подпрограмма решения линейных уравнений — Вариант 2

```
C ПРЕДОСТАВЛЕНА РАБОЧАЯ ПАМЯТЬ W
C A-НЕ УНИЧТОЖАЕТСЯ
  CALL SOLVE (A,X,B,N,I,J,K,L,M,M1,W)
  ---
  SUBROUTINE SOLVE (A,X,B,N,I,J,K,L,M,M1,W)
  DIMENSION A(I,J), X(I,K), B(I,K), W(I,J)
```

Теперь мы достигли гибкости, однако пользователь скажет: «Хорошенькое дело, я никогда не пойму этого нагромождения!

Единственное, что я хочу,— это решить мою маленькую систему 5×5. Я составлю свою собственную программу».

Ситуация может быть в некоторой степени улучшена, если использовать лучшую мнемонику для имен переменных:

Вариант 1 — С хорошими именами переменных

```
SUBROUTINE SOLVE (A,X,B, NEQS, KDIMA, KDIMB, NSOLS, INVERS, ISING)
DIMENSION A(KDIMA, KDIMA), X(KDIMA, KDIMB), B(KDIMA, KDIMB)
```

Заметим, в данном случае массив A, в котором задается исходная матрица, является квадратным. Два аргумента KDIMA и KDIMB имеют искусственный характер в силу особенностей Фортрана. Другие параметры определяются существом задачи. Следует иметь в виду, что в Фортране (и в большинстве других языков) матрица A не передается в подпрограмму как набор чисел. В действительности передается только адрес первой ячейки массива (т. е. его начало в памяти), и подпрограмма типа SOLVE манипулирует с A в области памяти, выделенной в вызывающей программе, сама SOLVE не имеет памяти для хранения матрицы.

Можно устранить переключатель INVERS из списка параметров и рекомендовать пользователю обращать матрицу, например, следующим образом:

```
B =<ЕДИНИЧНАЯ МАТРИЦА>
CALL SOLVE( A,X,B,NEQS,KDIMA,KDIMB,NEQS,ISING)
IF( ISING .EQ. 0) GO TO<СООБЩЕНИЕ ОБ ОШИБКЕ>
```

Можно включить в библиотеку следующую простую подпрограмму:

```
      SUBROUTINE LINSYS (A,X,B,NEQS)
      REAL A(NEQS, NEQS), X(NEQS), B(NEQS)
      CALL SOLVE( A,X,B,NEQS,NEQS,NEQS, 1, ISING)
      IF ( ISING .EQ. 0) PRINT 10
10    FORMAT( 20X,20(1H*), 17H МАТРИЦА ВЫРОЖДЕНА)
      RETURN
      END
```

Таким образом, мы видим, что даже для такой стандартной задачи существуют нетривиальные вопросы проектирования подпрограмм. А ведь мы оставили в стороне вопрос о том, как может пользователь сообщить подпрограмме SOLVE, что матрица A — симметричная или ленточная и т. д.

6.Г.1. Распределение памяти и переменные размерности

При работе с подпрограммами матричного исчисления многие пользователи испытывают затруднения, вызванные различием между размером матрицы A в системе линейных уравнений $A\mathbf{x}=\mathbf{b}$

и размером A как фортранного массива. Прежде всего библиотечная подпрограмма *не в состоянии* отождествить эти размеры. Рассмотренная выше подпрограмма LINSYS в действительности не очень полезна, поскольку она не обладает гибкостью в использовании, когда число уравнений меняется в пределах одного задания. Проиллюстрируем это обстоятельство двумя фрагментами программ на Фортране, которые являются типичными для приложений:

```
      REAL A(50,50),R(50),B(50)
    5 READ 10, NEQS, ((A(I,J),I=1,NEQS),J=1,NEQS)
   10 FORMAT( I10/ (8F10.5))
      IF( NEQS .LE. 0) STOP
C     ДЕЛАЕМ КОЕ-ЧТО ЗДЕСЬ
      CALL SOLVE(A,X,B,NEQS, 50, 1, 1, ISING)
      IF( ISING .EQ. 0) PRINT 20
C     ПЕРЕХОДИМ К ДРУГОМУ СЛУЧАЮ
      GO TO 5
      END

      REAL A(50,50), X(50,10), B(50,10)
C     ГЕНЕРИРУЕМ ПЕРВУЮ ЗАДАЧУ
      CALL SOLVE(A,X,B,5,50,10,1,ISING)
C     ГЕНЕРИРУЕМ СЛЕД. ЗАДАЧУ - 2 ПРАВ. ЧАСТИ
      CALL SOLVE(A,X,B,NPROB2,50,10,2,ISING)
C     ПОСЛЕДНЯЯ ЗАДАЧА, 10 СЛУЧАЕВ
      CALL SOLVE(A,X,B,50,50,10,10,ISING)
      STOP
      END
```

Фортран обладает весьма ограниченными возможностями для действий с массивами различных размеров. Наиболее очевидный и простой подход — выбрать порядок матрицы A для подпрограммы SOLVE достаточно большим, скажем 100. При таком предположении в описании подпрограммы должно быть сказано, что массив A должен быть размеров 100×100, а SOLVE будет иметь вид

```
SUBROUTINE SOLVE (A, X, B, NEQS, ISING)
REAL A(100, 100), X(100), B(100).
```

Этот подход очень расточителен по памяти для тех, кто решает, скажем, четыре уравнения с четырьмя неизвестными. Пользователь, имеющий 102 уравнения, должен или составить свою собственную программу, или модифицировать библиотечную подпрограмму.

Использование средств Фортрана для управления переменными размерностями массивов помогает преодолеть отмеченные трудности за счет включения размерностей массивов в список параметров, передаваемых при обращениях к подпрограмме. Несмотря на то, что использование переменных размерностей является очевидным выбором для библиотечных подпрограмм, можно надеяться, что когда-либо Фортран (или какой-то другой общепринятый язык программирования) устранит это излишнее и запутывающее свойство.

Глава 7

ОЦЕНКА ХАРАКТЕРИСТИК МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

7.А. ВЛИЯНИЕ КОМПИЛЯТОРОВ НА ПОРТАБЕЛЬНОСТЬ И ЭФФЕКТИВНОСТЬ

Достигнуть высокого качества и портабельности математического обеспечения трудно, и здесь мы покажем, что невозможно избежать существенных потерь в эффективности при использовании одной и той же программы в различных вычислительных системах. Приводимые ниже данные взяты из работы (Parlett, Wang), приведенной в списке литературы в конце книги.

Рассмотрим два фундаментальных способа решения линейных уравнений и покажем, что независимо от того, как они запрограммированы на Фортране, имеют место значительные потери эффективности в некоторых вычислительных системах. Это означает: то, что хорошо для одних условий, плохо для других, и наоборот; более того, любой из этих двух способов плох для некоторых условий.

- Рассматриваются пять различных систем (или сред) проведения вычислений для двух различных вычислительных машин:

CDC 6400, компиляторы (с Фортрана) RUN, FUN и FTN,
IBM 360/50, компиляторы (с Фортрана) G и H.

Исследуются два фрагмента процесса решения системы.

1. *Реализация выбора ведущего элемента.* Выбор ведущего элемента может быть выполнен двояким образом: прямой перестановкой строк в памяти или использованием указателей (косвенная адресация) для хранения информации о порядке исключения.

2. *Внутренний цикл в модификации Краута.* В математических обозначениях внутренний цикл в модификации Краута метода исключения Гаусса имеет вид

$$a_{ij} \leftarrow a_{ij} - \sum_k a_{ik} * a_{kj}.$$

Это присваивание может быть закодировано в виде нескольких вариантов (в статье Парлетта и Уонга в этом цикле используется j+1 вместо j).

7.А.1. Влияние вычислительных условий на реализацию выбора ведущего элемента

Были составлены две программы, выполняющие только выбор ведущего элемента, но двумя различными способами. Фиксировалось время работы этих программ на каждой из упомянутых

выше вычислительных машин и для каждого компилятора. В табл. 7.1 представлены данные, характеризующие время исполнения для случая 75×75-матрицы по следующему правилу: берется отношение времени исполнения программы к лучшему времени исполнения и умножается на 100. Это означает, что число 100 представляет собой наилучший результат и присваивается наиболее быстрой программой реализации. Таким образом, в столбце 1 таблицы (IBM 360/50, компилятор G) показано, что явная перестановка строк для данных вычислительных условий представляет собой наиболее быстрый способ, тогда как использование указателей увеличивает время работы программы на 13%. Такого рода различия существенны и достигают 40%. Заметим, однако, что выбор ведущего элемента составляет небольшую часть исключения Гаусса, и даже 40% могут не оказать заметного влияния на весь процесс решения.

Таблица 7.1

Сравнение времени исполнения программ для двух реализаций выбора ведущего элемента в пяти условиях проведения вычислений Время нормализовано в каждом столбце так, чтобы число 100 характеризовало наиболее быстрое исполнение

	IBM 360/50		CDC 6400		
	компиляторы		компиляторы		
	G	H	RUN	FUN	FTN
Явная перестановка	100	143	100	125	140
Использование указателей	113	100	119	100	100

7.А.2. Влияние вычислительных условий на реализацию внутреннего цикла в модификации Краута

Рассмотрим три различных способа реализации этого цикла на Фортране.

Способ 1: *максимальное использование индексов*

```
      DO 1 K=1 M
1     A(I,J) = A(I J) - A(I K) * A(K J)
```

Способ 2: *вынесение члена с неизменяемыми индексами за границы цикла*

```
      SUM=A(I,J)
      DO 1 K=1,M
1       SUM = SUM - A(I,K) * A(K,J)
      A(I,J)    = SUM
```

Способ 3: способ 2 с добавлением временной переменной T

```
      SUM = A(I,J)
      DO 1 K=1,M
            T = A(I,J) * A(K,J)
1           SUM = SUM - T
      A(I,J)      = SUM
```

Были составлены три идентичные, за исключением этих фрагментов программы, решения систем линейных уравнений, время исполнения которых фиксировалось на каждой машине для каждого компилятора при решении систем порядка 25, 50 и 75. В табл. 7.2 представлены результаты для 75×75-матрицы (для других порядков результаты аналогичные). Как и в табл. 7.1, наиболее быстрой программе ставится в соответствие число 100, а более медленным — пропорционально бóльшие числа. Отметим, что время исполнения фиксировалось для всего процесса решения системы линейных уравнений, а не для выполнения внутреннего цикла. Конечно, внутренний цикл занимает значительную (вероятно, свыше 98%) часть времени работы всей программы.

Таблица 7.2

Сравнение времени исполнения программ, использующих три различных способа кодирования внутреннего цикла в модификации Краута
Время нормализовано в каждом столбце так, чтобы число 100 характеризовало наиболее быстрое исполнение

	IBM 360/50		CDC 6400		
	компиляторы		компиляторы		
	G	H	RUN	FUN	FTN
Способ 1:	149	117	115	119	100
Способ 2:	100	100	100	100	104
Способ 3:	102	105	190	190	107

Мы видим, что третий способ, который, несомненно, неестествен, никогда не оказывается наилучшим. Однако наказание за такую «небольшую» вариацию колеблется в пределах от 2 до 90%. Первый способ непосредственно реализует необходимые вычисления. Можно было бы предположить, что оптимизирующие компиляторы (до некоторой степени все эти компиляторы оптимизирующие) могут сгенерировать наилучший код для некоторых частных случаев вычислений, по крайней мере для цикла из двух строк. Однако только компилятор FTN сумел это сделать, а другие сгенерировали коды, которые существенно менее эффективны: от 15 до 49% потери эффективности.

Эти примеры показывают не только то, что не существует одного наилучшего способа программирования вычислений, они также показывают, что, казалось бы, небольшое различие в кодировании может привести к значительным различиям во времени исполнения программы.

7.А.3. Влияние компиляторов на эффективность программы

Рассмотренные данные позволяют исследовать значение различных уровней оптимизации в компиляторах.

Таблица 7.3

Сравнение времени исполнения трех программ на двух машинах для различных компиляторов
Время нормализовано так, чтобы число 100 характеризовало наиболее быстрое исполнение программы на данной машине

	IBM 360/50		CDC 6400		
	компиляторы		компиляторы		
	G	H	RUN	FUN	FTN
Способ 1:	395	100	165	170	100
Способ 2:	311	100	139	137	100
Способ 3:	304	100	257	255	100

Данные табл. 7.2 могут быть нормализованы так, чтобы наименьшее время исполнения каждой программы на каждой вычислительной машине равнялось 100; нормализованные таким образом данные помещены в табл. 7.3. Из этой таблицы видно, что компилятор H генерирует объектный код, который в три — четыре раза быстрее кода, сгенерированного компилятором G. Подобным образом, компилятор FTN генерирует код, который на 40—50% быстрее кода, генерируемого компиляторами RUN или FUN (они близки друг другу). Отсюда можно сделать вывод, что если даже небольшие изменения в кодировании могут привести к большим различиям в эффективности, то выбор компилятора может привести к гораздо большим различиям.

7.Б. ТЕСТИРОВАНИЕ И ОЦЕНКА ПРОГРАММ МАТРИЧНОГО ИСЧИСЛЕНИЯ

Обратимся к фундаментальному вопросу: *какая программа наилучшая?* Ограничимся здесь рассмотрением программ решения систем $Ax=b$.

7.Б.1. Предварительный отбор

Существуют определенные общие свойства, которыми должна обладать программа, прежде чем она будет рассматриваться как кандидат в математическое обеспечение. Таковыми свойствами являются:

1. *Хорошая структура и организация программы.* Программа должна иметь логически выдержанную и систематизированную организацию. Под этим обычно подразумевают модульное проектирование программы. Должна быть выдержана практика хорошего программирования, например, принципы структурного программирования и тщательное комментирование.

2. *Портабельность.* Обычно нет смысла в затрате сил на компоненты математического обеспечения, которые не будут широко использоваться. Даже патентованные программы, предназначенные для использования в пределах одной организации, должны быть защищены от возможных изменений в оборудовании или в вычислительной системе. Доказательство портабельности программы заключается в ее действительном использовании на нескольких типах машин или в результате применения к ней специальных проверяющих программ типа PFORT (которая проверяет фортранную программу на портабельность) (Ryder, 1974). Работа даже хорошей программы может зависеть от машины или операционной системы; решающий момент заключается в том, чтобы эти зависимости были хорошо идентифицированы и можно было их легко учитывать.

Следование стандартам языка является необходимым (но не достаточным) условием для портабельности. Переход с одной версии языка к другой (например, с Фортрана IV на Фортран 77) может создать дополнительные трудности, связанные с компиляторами.

3. *Хорошие руководства для пользователей.* Это требование кажется очевидным, и все же в руководствах часто оказывается много путаницы. Одна из причин этого в том, что автору программы иногда трудно представить себе, каким образом другой человек будет воспринимать его программу.

Программа, не обладающая этими свойствами, должна быть просто возвращена автору. Недостатки подобного рода, конечно, выявляются, если подвергнуть такую программу процедуре тестирования.

7.Б.2. Оценка рабочих характеристик программ решения Ax=b

В этом подразделе очерчивается материал, который будет рассмотрен значительно более детально и с большей общностью в конце настоящей главы.

Пусть имеется одна или больше программ решения системы $\mathbf{Ax}=\mathbf{b}$, и пусть необходимо оценить или сравнить их рабочие характеристики. Первый, и быть может решающий, вопрос: какие задачи следует рассматривать? Таким образом, необходимо определить «пространство задач» для проведения оценок. Можно выбрать 5, 50 или 500 модельных задач, составляющих это «пространство». Такой подход может оказаться наилучшим (или даже единственным), если речь идет об очень специализированном множестве задач (например, о задачах, возникающих в некоторых областях вычислений, таких, как расчеты особого вида мостов или численный анализ сетей). Чаще интерес представляет рассмотрение задач общего класса, который описывается в терминах характерных *признаков* или атрибутов. Типичные характерные признаки задачи $\mathbf{Ax}=\mathbf{b}$:

Размер матрицы	Разреженность
Число правых частей	Обусловленность
Симметрия	Погрешности задания матрицы
Ширина ленты	Масштаб матрицы

Эти признаки определяют численные координаты в *пространстве признаков*, и мы выбираем область в этих координатах для проведения оценок. Отметим, что пространство признаков имеет гораздо меньшую размерность, чем исходное пространство задач. Выше отмечены восемь измерений, тогда как множество элементов 10×10-матриц имеет 100 измерений, одно для каждого элемента матрицы.

Существуют три причины для исследования пространства признаков: (1) Когда возникает новая задача, обычно можно определить (или угадать) только некоторые из ее признаков и вряд ли возможно точно определить ее место по отношению к другим задачам пространства. (2) Предполагается, что программы сходным образом решают задачи со сходными признаками. Если это не так, то такие признаки являются «неадекватными» и не рассматриваются. (3) Мы должны сузить рамки исследований и множество задач для того, чтобы вообще сдвинуться с места.

Как только определены пространство задач и пространство признаков, мы должны перейти к выбору *критериев для оценки рабочих характеристик*. Четыре обычных критерия для $\mathbf{Ax}=\mathbf{b}$:

p_1 = время исполнения,	p_3 = достигнутая точность,
p_2 = размер памяти,	p_4 = надежность.

Понятие «наилучший» требует линейного упорядочения, поэтому необходимо выбрать какой-нибудь способ объединения этих, в основном, независимых критериев. Мы могли бы представить их численно и нормировать так, чтобы значения лежали в пределах от 0 до 1. Затем можно выбрать веса w_i, $i=1, 2, 3, 4$, $\sum w_i=1$, и

определить обобщенную рабочую характеристику как

$$\sum_{i=1}^{4} w_i p_i.$$

В действительности можно выбор весов w_i предоставить пользователю. В идеальном случае пользователь мог бы указать эти веса и затем выбрать программу, которая имеет максимальную обобщенную рабочую характеристику. Это не всегда возможно — например, когда производится выбор программы для включения ее в библиотеку программ вычислительного центра.

Перейдем к вопросу измерения рабочих характеристик программ. Было бы идеальным сделать это для всего пространства задач; в этом случае мы действительно «знали» бы характеристики программ. Почти всегда такой подход непрактичен, так как он требует слишком больших вычислений, и поэтому необходимо каким-либо способом отбирать задачи. Лучше всего было бы выбрать множество задач, признаки которых хорошо заполняют пространство признаков задач из рассматриваемого пространства. Однако даже такой подход часто оказывается непрактичным. Заметим, что, имея восемь признаков и десять значений для каждого из них, мы должны решить 10^8 задач. Даже пять признаков с пятью значениями для каждого приведут к 3 125 задачам, что может быть гораздо больше того числа задач, которые мы в состоянии решить. Поэтому следует использовать статистические приемы в отборе задач и в организации экспериментальных расчетов или по крайней мере весьма тщательно отобрать наиболее важные случаи.

Оценка характеристик может быть еще более усложнена неопределенностью в выборе типичной для данного пространства задачи. Так, при решении $\mathbf{Ax}=\mathbf{b}$ легко измерить время исполнения программы и требующуюся память. Часто эти два критерия зависят непосредственно от порядка матрицы и в незначительной степени от других обстоятельств. С другой стороны, точность не так легко измерить. Как мы увидим в гл. 8, понятие «правильный ответ» не так хорошо определено, как мы того хотели бы. Надежность еще труднее измерить. Помимо всего прочего, надежность зависит от всего пространства задач, так что выбор модельных задач для измерения надежности должен быть сделан с большой осторожностью.

При любой оценке рабочих характеристик программы необходимо представить в численном виде различные наблюдения и привести экспериментальные результаты к некоторой легко воспринимаемой форме. Например:

1. Ожидаемое влияние ошибок составляет 3×10^{-8}.
2. Максимальное влияние ошибок округления составляет $5\times\times10^{-7}$.
3. Требуемая точность достигается в 98,6% случаев.

4. Время исполнения растет как 0.073 N^3 для $N \times N$-матрицы.

Критерий, выраженный одним числом, как в первых трех примерах, предпочтительнее. Однако иногда одно число не может передать достаточно информации, и тогда для демонстрации результатов используется понятие *характеристического профиля* программы.

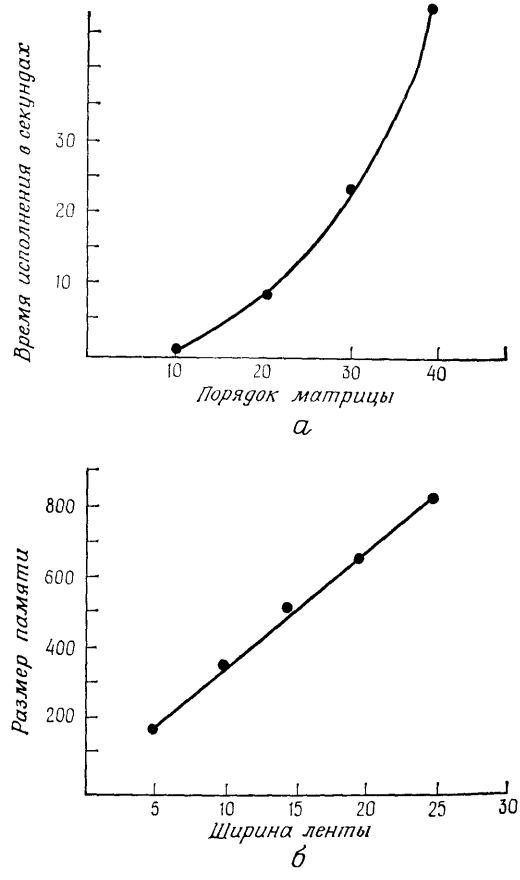
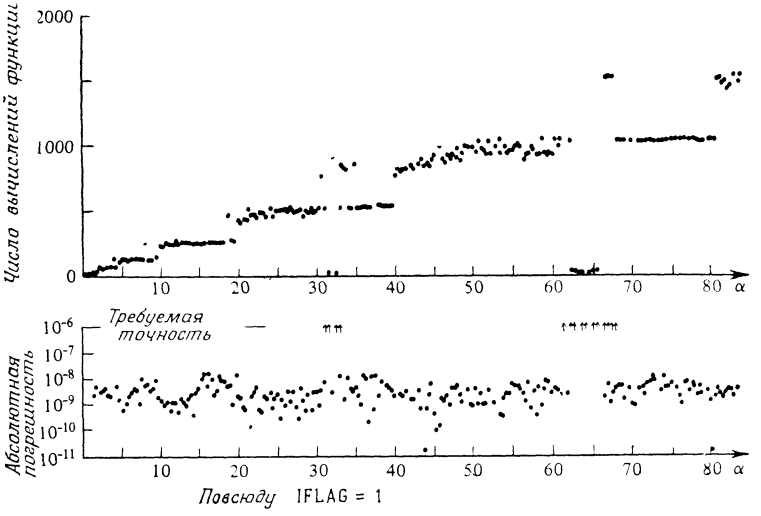
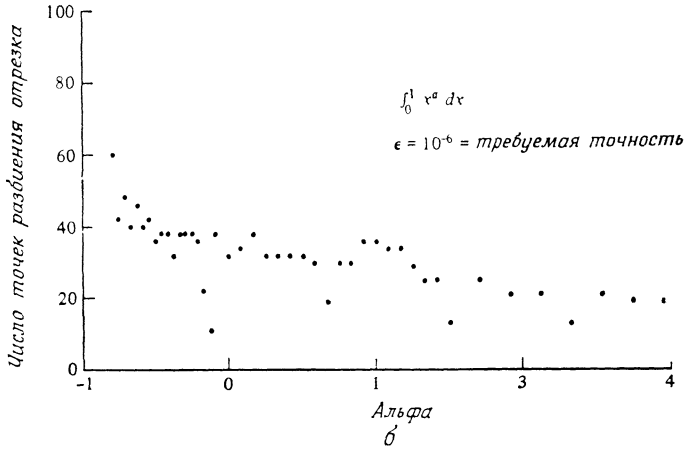


Рис 7.1. Два характеристических профиля программы решения $Ax=b$. (а) Зависимость времени исполнения программы стандартного исключения Гаусса от порядка матрицы. (б) Зависимость использованной памяти от длины ленты для программы решения систем с ленточными матрицами для случая 25×25-матрицы.

Характеристический профиль представляет собой график зависимости рабочей характеристики от одного признака задачи (или от одного параметра семейства задач). Графики зависимости: времени исполнения программы от порядка матрицы, памяти от порядка матрицы, точности от числа обусловленности матрицы и памяти от разреженности — весьма полезны для исследования задачи $Ax=b$. На рис. 7.1 приведены два примера для этой задачи. На рис. 7.2 представлены два характеристических профиля программ



а



б

Рис. 7.2. Характеристические профили для двух подпрограмм численного интегрирования. (а) Подпрограмма CADRE [de Boor, 1971] вычисляет $\int_0^1 [\cos(\alpha\pi x)+1]dx$. Этот профиль показывает, что трудозатраты увеличиваются с ростом осцилляции, но неожиданным образом. Большие ошибки вблизи $\alpha=32$ и 64 обусловлены взаимодействием алгоритма и частоты. (б) DQUAD [Blue, 1975] вычисляет $\int_0^1 x^\alpha dx$. Этот профиль показывает, что трудозатраты подпрограммы не определяются степенью особенности подынтегральной функции; трудозатраты медленно снижаются с увеличением α.

численного интегрирования (впервые понятие характеристического профиля было введено для численного интегрирования).

Отметим, что систематическая оценка рабочих характеристик программ оказалась весьма плодотворной для новых идей и разработки технических приемов создания хороших алгоритмов. Было также показано, что мнения экспертов в определении, какой метод работает на практике наилучшим образом, не очень надежны.

7.Б.3. Доказательство правильности программ

Несмотря на то, что программы решения задач вычислительной математики были первыми компонентами программного обеспечения, получившими интенсивное развитие, очень мало сделано для «доказательства их правильности», т. е. не были предприняты попытки доказательства правильности в каком-либо математическом смысле. Здесь можно выделить следующие причины:

(1) Трудно учесть в доказательствах неопределенность ошибок округления. Конечно, можно предположить, что ошибки округления отсутствуют и считать результаты точными. (2) Программы математического обеспечения, как правило, слишком велики для современных методов доказательства (как автоматических, так и ручных). Например, для программы, состоящей из 661 строки (368 строк комментариев, 39 декларативных и 254 выполняемых операторов), потребовалось 20 страниц ручного доказательства правильности работы, после того как на 15 страницах была доказана правильность ее более абстрактной версии (Rice, 1976). (3) Большинство программ содержит эвристические сегменты, рабочие характеристики которых не могут быть установлены в терминах соотношений между входными и выходными данными. Это означает, что вопрос состоит не в установлении правильности этих сегментов, а скорее в том, насколько хорошо выполняют они свои функции. Можно провести аналогию с программой игры в шахматы. Даже если и может существовать правильный ход во всех ситуациях, никто не знает, каким он должен быть, и правильность выбранного программой хода может быть оценена тем, насколько она успешно выступает на шахматных соревнованиях.

В численных расчетах наиболее распространенной ситуацией является проверка сходимости некоторого процесса. Во многих случаях может быть *доказано*, что ни одна проверка на сходимость не является надежной независимо от длины программы, реализующей эту проверку. Тем самым любая реально используемая программа должна удовлетворять двум требованиям — нужно, чтобы ее было трудно «одурачить» и чтобы она была «весьма надежной для реальных приложений». Эти требования неопределенные, и в попытках их достижения используются эвристические тесты. Значимость этих тестов определяется обычно удобством их использо-

вания; не существует способа математически сформулировать, что именно они должны обеспечить.

В некоторых ситуациях идея доказательства правильности программы может быть модифицирована. Например, можно было бы доказать следующее утверждение о подпрограмме:

«Эта подпрограмма находит такое значение x, что ABS(F(x))<EPS, или присваивает значение 1 параметру IFAIL».

Такого рода утверждение может нести решающую информацию для установления надежности программы, но не дает ключа к определению того, когда программа получает правильный результат [предполагается, что для правильного результата требуется выполнение условия |F(x)|<EPS]. Можно было бы попытаться сделать такое предположение относительно F(x), которое гарантирует выполнение ABS(F(x))<EPS. В такого рода предположениях могут фигурировать математические свойства F(x), которые трудно проверить на практике или которые ограничивают применимость доказательства тривиальным или вырожденным множеством входных данных. Только *проверяемые предположения* имеют значение для доказательства правильности программ.

7.В. ОТЧЕТ О ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТАХ

Оценка рабочих характеристик программ — прежде всего экспериментальная наука. Анализ алгоритмов и связанные с этим технические приемы могут дать хорошие указания на то, чего следует ожидать от многих программ, однако должны быть выполнены реальные измерения, чтобы быть уверенным в характеристиках программы. Например, время исполнения программы, реализующей метод исключения Гаусса, должно расти как N^3, где N — число уравнений. Однако и здесь могут наблюдаться существенные различия, например $0.05N^3$ или $500N^3$ (см. табл. 7.2). Для небольших значений N (скажем, $N \leqslant 6$) время исполнения этой программы почти наверняка лучше представляется формулой $a+bN+cN^2$ (с соответствующими коэффициентами a, b и c). Всегда есть вероятность того, что программистская ошибка совершенно изменит ожидаемое поведение программы.

Существуют определенные стандартные процедуры, применяемые в экспериментальных исследованиях для того, чтобы оценить степень достоверности основанных на экспериментах выводов. Самые высокие требования к этим процедурам: они должны быть систематичными, организованными, объективными, рассматривать все относящиеся к делу воздействия и точно сообщать, что было сделано и каковы результаты. В каждом экспериментальном исследовании развиваются свои собственные технические приемы, и материал этой главы представляет некоторые из таких приемов в области

Таблица 7.4

Контрольный список вопросов для отчета о вычислительных экспериментах

	Информация об алгоритме
Обязательно:	Полное описание алгоритма и класса предполагаемых для решения задач.
Желательно:	Информация о вычислительной сложности, скорости сходимости, оценках ошибки, числе операций на одном шаге и т д.
	Программа для вычислительной машины и вычислительные условия
Обязательно:	Язык программирования и компилятор. Вычислительная машина, операционная система и их версии. Форматы входных данных, допуски и другие соглашения о программе. Примененные специальные технические приемы или тактика.
Желательно:	Информация о доступности программы, руководства по использованию и т. д.
	Проектирование экспериментов и результаты
Обязательно:	Ясное формулирование целей экспериментов. Документация по использованным процедурам, предварительная обработка задач или последующая обработка данных. Описание ожидаемых данных, включая единицы измерения и точность. Рассмотренные семейства задач и использованный метод выборки
Желательно:	Применение стандартных задач, предварительных обработок, единиц измерения и т. д. Доступность задач, программ и процедур для использования другими.
	Отчет о результатах
Обязательно:	Полное описание плана проведения эксперимента и процедур. Обоснование плана экспериментов и использованных способов измерения рабочих характеристик. Как проводились измерения с детальным анализом (если нужно). Имевшие место неудачи, если возможно, с объяснениями. Использованные параметры программы (например, допуски на параметры, начальные значения, варианты, переключатели)
Желательно:	Влияние различных составляющих эксперимента, таких, как критерии окончания счета, допуски на параметры или способы измерении.

Продолжение

Сформулированные заключения	
Обязательно:	Ясное различие между полученными и предполагаемыми результатами.
	Описание сделанных предположений.
Желательно.	Указания относительно будущего исследования или улучшений алгоритма.
	Идентификация специальных классов задач, для которых наблюдались интересные случаи.

математического обеспечения. Кроме того, со статистической точки зрения существуют общие технические приемы проектирования экспериментов. Хотя эти статистические процедуры не рассматриваются в этой книге, они составляют существенную часть многих способов оценки рабочих характеристик программ.

Среди разработчиков алгоритмов и программ нет традиций проведения высококачественных оценок рабочих характеристик математического обеспечения. Этот раздел основан на работе (Crowder, Dembo and Mulvey, 1979), в которой дан обзор свыше 50 статей, посвященных оценкам алгоритмов и программ для определенных классов задач. Ни в одной из этих статей не были применены по-настоящему последовательные эксперименты; во многих отношениях большинство из них были признаны неудовлетворительными. Наиболее распространенный недостаток состоит в том, что выводы приводятся без адекватного описания экспериментов или полученных данных. Авторы цитируют одну статью, содержащую следующее утверждение: «Поскольку эти методы запрограммированы для различных машин на различных языках различными программистами, то не имеет большого смысла давать детальную оценку результатов, особенно потому, что так много задач оказались вырожденными. Однако результаты показывают, что...»

Такой эксперимент не может ничего показать, однако часто трудно выступать против сформулированных выводов, так как в статьях не приводятся данные, на которых эти выводы построены.

В табл. 7.4 приводится список вопросов для оформления отчетов о вычислительных экспериментах. Некоторые из этих вопросов не нужны для всех экспериментов и наоборот, некоторые эксперименты требуют других вопросов. Тем не менее этот список представляет собой хороший ориентир относительно количества и качества информации, которая требуется, чтобы вселить уверенность в выводах, полученных в результате экспериментов над программами.

7.Г. ПРОБЛЕМА ВЫБОРА АЛГОРИТМА

Проблема выбора эффективного (хорошего или наилучшего) алгоритма возникает во многих ситуациях. Частные особенности этих ситуаций часто затмевают общие характерные черты проблемы выбора, и целью настоящего раздела является построение абстрактной модели, соответствующей рассматриваемой проблеме.

Следует явно оговорить, что такого рода абстрактные модели не приводят непосредственно (при помощи их простой специализации) к процедурам наилучшего выбора. Всегда необходимо использовать специфические особенности конкретной ситуации. Тем не менее эти абстрактные модели проясняют рассмотрение проблемы выбора.

Процедура выбора часто заключается в присвоении значений параметрам некоторой общей «формы». Более точно, сама по себе процедура выбора — это алгоритм [1]), а каждый конкретный класс алгоритмов определяется свободными параметрами, которые в свою очередь выбираются так, чтобы удовлетворить (настолько хорошо, насколько они это могут) целям проблемы выбора. Примерами классических форм являются полиномы (с коэффициентами в качестве параметров) и линейные уравнения (с матричными коэффициентами или весами в качестве параметров). Другими формами, относящимися к проблеме выбора, являются деревья решений (обладающие размером, структурой и отдельными элементами дерева решений в качестве параметров) и программы (с различными составными частями программы в качестве параметров). Задание параметров последнего вида не типично для программ матричного исчисления, где выбор приходится делать среди небольшого набора программ.

Представленные здесь модели нацелены главным образом на проблему выбора алгоритмов по некоторым из трех указанных ниже характеристикам:

Пространство задач. Множество задач весьма обширно и довольно разнообразно. Это множество имеет большую размерность в том смысле, что существует много независимых характеристик задач, которые важны для выбора алгоритма и его работы. Обычно имеется значительная неопределенность в знаниях, касающихся этих характеристик и их значимости.

Пространство алгоритмов. Множество алгоритмов, которые необходимо рассматривать, обширно и разнообразно. Вообще говоря, оно могло бы насчитывать миллионы алгоритмов, но практически — десятки. При подсчете алгоритмов не делается различий между двумя алгоритмами, которые идентичны, за исключением значения *какого-либо* числового параметра. Тем не менее это мно-

[1]) А значит, процедура выбора принадлежит некоторому классу алгоритмов. — *Прим. перев.*

жество имеет большую размерность и неопределенность относительно влияния характеристик алгоритмов.

Измерение рабочих характеристик. Критерии для проведения измерений рабочих характеристик отдельного алгоритма для конкретной задачи сложны и трудно сопоставимы (например, сочетание быстрого исполнения, высокой точности и простоты). И опять же существует значительная неопределенность в назначении этих измерений и интерпретации их результатов.

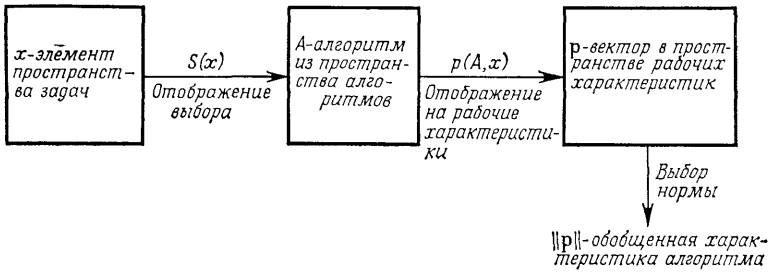


Рис. 7.3. Схема основной модели проблемы выбора алгоритма. Цель модели — определить S (x), такое, чтобы получить алгоритм с наилучшей обобщенной характеристикой.

Рис. 7.3 иллюстрирует основную абстрактную модель выбора, в которой:

Пространство задач = набор решаемых задач (данных программы).

Пространство алгоритмов = набор алгоритмов (программ), которыми можно пользоваться.

S = отображение, которое каждой конкретной задаче ставит в соответствие алгоритм ее решения (отображение выбора).

p = вектор значений (результатов измерений) рабочих характеристик алгоритма A, примененного к задаче x.

Норма дает *одно число* для обобщенной оценки рабочих характеристик алгоритма для конкретной задачи.

В завершение рассуждений определим теперь *проблему выбора алгоритма так:* по всем заданным элементам рассмотренной выше модели определить отображение выбора S (x).

Конечно, должны существовать критерии такого выбора. Приведем три основных критерия:

1. *Наилучший выбор.* Определить такое отображение выбора

B (x), которое дает алгоритм с наилучшими рабочими характеристиками для каждой задачи; иными словами, $\|p(B(x), x)\| \geqslant \|p(A, x)\|$ для всех возможных алгоритмов A.

2. *Наилучший выбор для подкласса задач.* Пусть надо выбрать только один алгоритм для решения каждой задачи из некоторого подкласса. Определить такой алгоритм A_0, который минимизирует ухудшение рабочих характеристик для задач этого подкласса [по сравнению с выбором B (x)]. Иными словами, A_0 дает рабочие характеристики близкие, насколько это возможно, к характеристикам наилучшего алгоритма, определенного B (x).

3. *Наилучший выбор из подкласса отображений.* Пусть надо ограничить отображение S (x) так, чтобы оно было определенного вида или было определено на некотором подклассе алгоритмов. Определить такое отображение выбора S*(x), которое минимизирует ухудшение рабочих характеристик алгоритма для всех задач.

Пример 7.1: Игровая задача. Требуется найти алгоритм для игры в крестики и нолики. Пространство задач представляет собой множество частных игровых вариантов. Хотя их число велико, в действительности существуют 28 различных имеющих смысл для рассмотрения игровых вариантов, если исключить грубые ошибки, симметрию и вращения игральной доски. Пространство алгоритмов может быть представлено как множество больших таблиц ответов для каждой игровой ситуации. Однако мы ограничим свой выбор подклассом алгоритмов, представляя алгоритм в виде дерева решений, которое включает в себя только позиции, дающие непосредственный выигрыш и различные виды свободных позиций. На рис. 7.4 представлен образец такого алгоритма. Всего существует 16 параметров a_i, которые принимают одно из пяти значений:

1. Выигрышный ход.
2. Блокировка выигрышного хода противника.
3. Ход в центральный квадрат.
4. Ход в угловой квадрат (первый свободный по часовой стрелке от правого верхнего квадрата).
5. Ход в боковой квадрат (первый свободный по часовой стрелке от правого квадрата).

Этот пример настолько прост, что можно немедленно присвоить определенные значения a_i. Эксперименты показывают, что разновидности грубых схем для вычисления a_i при выборе наилучшего алгоритма работают очень быстро. Тем не менее все же интересно подумать о том, как их выбирать, если не иметь никакого опыта в этой игре.

На практике выбор алгоритма обычно базируется на *свойствах* задачи, а не на задаче так таковой. Расширение абстрактной модели на этот случай показано на рис. 7.5.

Дополнениями в этой модели являются пространство векторов значений свойств задачи и F — отображение выделения свойств, которое связывает свойства с задачей. Заметим, что отображение выбора теперь зависит только от свойств f (x), в то время как отображение на пространство рабочих характеристик все еще зависит

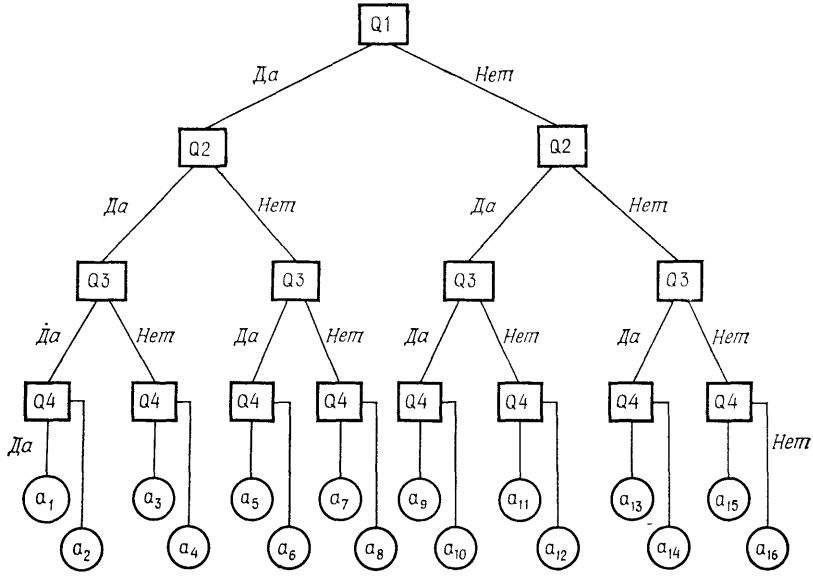


Рис. 7.4. Образец отображения выбора очередного хода для игры в крестики и нолики. Каждое из a_i означает один из пяти ходов. Q_1 — это вопрос.
Q1 : Имеется ли выигрышная позиция?
Q2 : Имеет ли противник выигрышную позицию?
Q3 : Свободен ли центр?
Q4 : Свободен ли угол?

от задачи x. Введение свойств может рассматриваться как способ, позволяющий систематизировать введение подкласса задач в основную модель.

Предыдущие рассуждения относительно проблемы выбора алгоритма и критерии выбора остаются справедливыми для расширенной модели. Определение свойств, которые будут использованы, является частью процесса выбора и часто одной из наиболее важных ее частей. Эти свойства могут рассматриваться как попытка ввести приблизительную координатную систему в пространство задач. В идеализированном случае любой рассматриваемый алгоритм должен иметь одинаковые рабочие характеристики на задачах с одинаковыми свойствами. Однако этот идеал редко достигается.

Определение наилучших (или хотя бы хороших) свойств является одним из наиболее важных и все еще неясных аспектов проблемы

выбора алгоритмов. Многие пространства задач формулируются в неясных терминах, и поэтому часто используется экспериментальный подход для оценки рабочих характеристик алгоритмов для

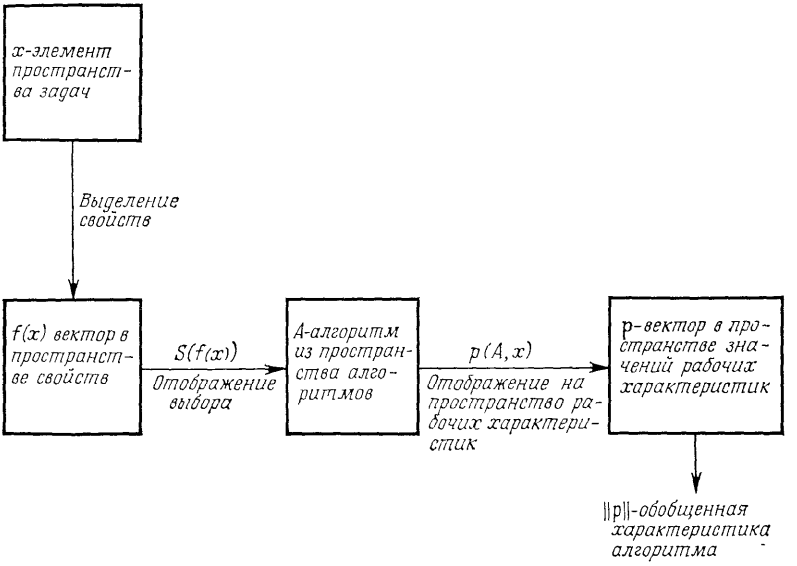


Рис. 7.5. Схема модели выбора алгоритма на основе свойств задачи. Отображение выбора зависит только от свойств f(x), хотя рабочие характеристики по-прежнему зависят от задачи x.

заданного пространства задач. Это означает, что выбирается и затем исследуется модельная задача. Очевидно, решающим для такого подхода является выбор подходящей модельной задачи, и если имеется хороший набор свойств задач, то по крайней мере можно выбрать модельную задачу так, чтобы она была представительной по отношению к этим свойствам. Заметим, что хорошие свойства представляют собой информацию, наиболее подходящую для оценки рабочих характеристик алгоритмов.

Для некоторых хорошо разработанных областей вычислений существуют общие соглашения относительно множества свойств. Например, для матричного исчисления эти свойства включают в себя такие понятия, как малость порядка, разреженность, ленточность, диагональное преобладание, положительная определенность, плохая обусловленность. При известных значениях этих свойств, опытный специалист может выбрать алгоритм, подходящий для этой задачи, со значительной степенью уверенности. Проблема выбора алгоритма численного интегрирования — уже гораздо труднее, а проблема решения совместной системы нелинейных уравнений вообще плохо понята. Если такая ситуация имеет место для

задач, которые исследовались в течение одного или двух столетий, то нас не должны удивлять трудности и неопределенности в задачах, появившихся в последние одно-два десятилетия.

До сих пор мы предполагали, что существует фиксированный способ измерения рабочих характеристик конкретного алгоритма для конкретной задачи. Существуют некоторые ситуации, когда разумно рассматривать критерии для измерения характеристик в качестве входных данных проблемы выбора. Рассмотрим, например,

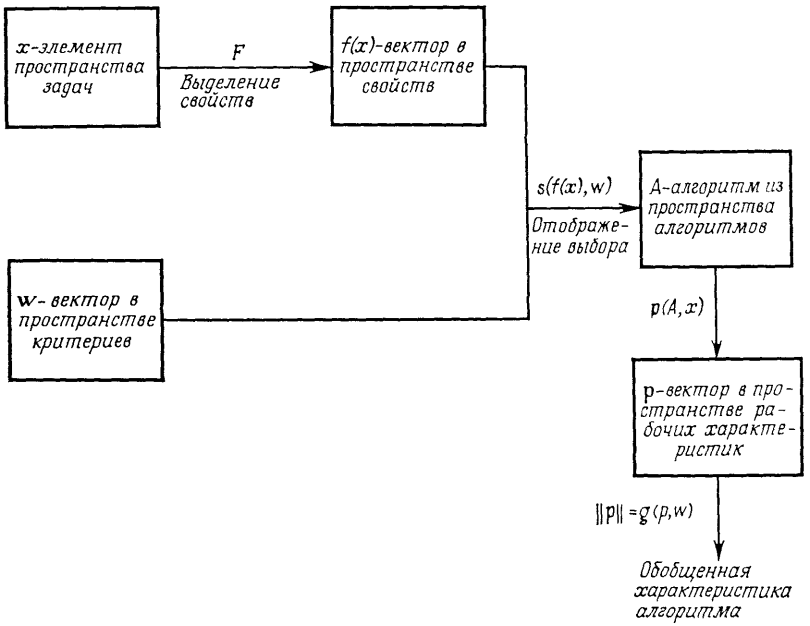


Рис. 7.6. Схема модели выбора алгоритма, основанная на свойствах задачи и переменных критериях для рабочих характеристик.

выбор программы решения обыкновенных дифференциальных уравнений и критерии: скорость, точность, надежность и легкость использования. В различных ситуациях веса для каждого из критериев могут варьироваться в пределах почти от 0 до почти 100%. Модель этой модификации проблемы выбора алгоритма показана на рис. 7.6.

В эту модель выбора введена дополнительная функция нормы g, зависящая от значений рабочих характеристик алгоритма и значений критериев. Она определяет обобщенную характеристику алгоритма **p** (A, x) с учетом вектора весов **w**.

Можно было бы сформулировать новые модификации модели для проблемы выбора алгоритма на основе пространства критериев. В этой формулировке переменными являются:

Подклассы задач: P_0.
Подклассы алгоритмов: A_0.
Подклассы отображений выбора: S_0.
Пространство свойств: F.
Выбор нормы: g.

Число интересных комбинаций довольно велико, но мы воздержимся от их рассмотрения.

Пример 7.2: Постановка на очередь заданий для вычислительной машины. В общем случае эта проблема выбора формулируется так: рассмотрим вычислительную установку фиксированной конфигурации и с рабочей нагрузкой, имеющей достаточно хорошо известные свойства. В какой последовательности следует распределить задания, чтобы добиться наибольшей производительности?

Тщательное исследование этой проблемы требует многих сотен страниц, что превосходит объем этого раздела. Мы сформулируем простой, но не тривиальный частный случай этой проблемы, оставаясь в рамках абстрактных моделей. Затем мы опишем упрощенную версию реального алгоритма распределения и после этого сформулируем гораздо более специальную проблему выбора алгоритма.

Абстрактная модель включает в себя пространство задач, пространства алгоритмов, свойств, критериев и значений рабочих характеристик. Опишем их следующим образом:

1. *Пространство задач.* Оно состоит из конфигураций находящихся в решении заданий, которые представляют собой смесь пакетных, дистанционно-пакетных и интерактивных заданий, а также заданий, выполняемых в режиме разделения времени. По своей природе эти смеси весьма динамичны, и, как правило, известны только средние значения их характеристик (большинство этих значений точно не известны). В дополнение к очень быстрым и существенным изменениям в характеристиках задач часто имеют место значительные и долговременные отклонения в средних значениях этих характеристик.

2. *Пространство свойств* — это комбинация свойств индивидуальных заданий. Свойства индивидуальных заданий, которые могли бы представлять интерес, указаны ниже ключевыми словами и короткими пояснениями:

Приоритет: значение приоритета устанавливается пользователем и вычислительным центром.

Время центрального процессора: время, установленное для задания пользователем, минус время, уже использованное на выполнение задания.

Память: длина участка памяти для задания, установленная пользователем и контролируемая операционной системой (могут

быть рассмотрены участки как оперативной, так и вспомогательной памяти).

Требования на устройства ввода/вывода: требования, установленные пользователем на применение стандартных устройств (принтеры, перфораторы, дисководы и т. д.).

Специальные устройства: указания на использование менее употребительных устройств (например, лентопротяжки, графопостроители, устройства ввода графической информации).

Местоположение в памяти и стабильность задания: указание на вероятность запросов дополнительных страниц памяти или на перемещения задания во внешнюю память.

Кроме того, могут быть рассмотрены следующие свойства конфигурации задач в целом:

Пакетная загрузка: длина очереди на выполнение и средние значения некоторых свойств заданий.

Оперативная загрузка: число пользователей терминальных устройств и средние значения свойств потока заданий, поступающих с терминалов.

Интерактивная загрузка: число пользователей и свойства используемых интерактивных систем.

Загрузка устройств ввода/вывода: длина очередей на различных устройствах ввода/вывода.

3. *Пространство алгоритмов.* Было предложено и подвергнуто анализу значительное количество разнообразных простых алгоритмов, определяющих порядок обслуживания заданий (например, обслуживание заданий по кругу, обслуживание первым первого поступившего задания, обслуживание по приоритету). Быстрота исполнения является важным свойством, присущим удачным алгоритмам (в противном случае операционная система тратит чрезмерно большое количество своих ресурсов на определение порядка обслуживания заданий вместо их выполнения). Это дает преимущество очень простым схемам, однако необходимо отдавать себе отчет в том, что и довольно сложные алгоритмы могут быть быстрыми в исполнении.

4. *Измерение рабочих характеристик.* Оценка характеристик операционной системы зависит от точки зрения конкретного человека — каждый пользователь хочет, чтобы его собственное задание выполнялось как можно быстрее, а директор вычислительного центра не хочет иметь неудовлетворенных клиентов. Ни одно из этих желаний не является слишком реалистичным, тем не менее попытки оценить прогресс, достигнутый на пути их удовлетворения, обычно заключаются в определении производительности вычислительной системы и времени ее ответа. Следующие измерения проводятся для различных классов заданий:

Пакет: время ответа для небольших заданий; среднее и максимальное время оборота заданий, требующих небольших ресурсов.
Пакет: время ответа для больших заданий; среднее и максимальное время оборота всех пакетных заданий, кроме заданий с небольшими ресурсами (или заданий со специальными запусками).
Оперативный ответ: среднее и максимальное время ответа для заданий, выполняющих общие сервисные функции (например, выборка файла, редактирование строки, запуск пакетного задания).
Интерактивный ответ: среднее и максимальное время ответа для стандартных коротких запросов.
Производительность: общее число заданий, выполненных за единицу времени; число часов центрального процессора, вырабатываемых за день и т. д.

5. *Пространство критериев.* Оно состоит из числовых весов, отражающих относительную важность значений рабочих характеристик. Значения некоторых из этих характеристик могут быть улучшены только за счет ухудшения других характеристик, и трудно сопоставлять их друг с другом. Простым, но часто удовлетворительным приемом, является масштабирование значений характеристик к стандартному интервалу (скажем, от 0 до 1) с последующим приписыванием им весов (которые в сумме дают 1).

Рассмотрим упрощенный вариант реального алгоритма расписания для заданий, применяемых в операционной системе Университета Пэрдью. Расписание составляется в соответствии с приоритетом. Это означает, что если ожидающее исполнения задание имеет приоритет QP_1, больший, чем приоритет QP_2 исполняемого задания, и если величина CM_2 оперативной памяти, занимаемой исполняемым заданием, достаточна для размещения ожидающего задания (которое требует участка оперативной памяти длины CM_1), то исполняемое задание прерывается и записывается во внешнюю память, а ожидающее задание считывается в оперативную память и начинает выполняться. Короче, если $QP_1 > QP_2$ и $CM_1 < CM_2$, то задание 2 записывается во внешнюю память и замещается заданием 1.

Приоритет QP представляет собой функцию следующих четырех параметров $\mathbf{r}=(r_1, r_2, r_3, r_4)^T$:

r_1 = параметр приоритета задания, указанный на карте JOB;
r_2 = размер оперативной памяти, требуемый заданием (в данный момент);
r_3 = отрезок времени центрального процессора, оставшийся от общего времени, заказанного для задания перед его исполнением;
r_4 = требуемые устройства ввода-вывода, оставшиеся от общего числа заказанных перед исполнением задания устройств.

Значение QP определяется линейной комбинацией

$$QP = \sum_{i=1}^{4} R_i(r_i),$$

где

$$R_1(r_1) = a_1 * r_1,$$
$$R_2(r_2) = a_2 * (150,\ 100 - r_2),$$
$$R_3(r_3) = a_3 * (a_4 - r_3)_+^0 + a_5 * (a_6 - r_3)_+^0 + a_7 * (a_8 - r_3)_+^0,$$
$$R_4(r_4) = a_9 * (a_{10} - r_4)_+^0 + a_{11} * (a_{12} - r_4)_+^0 + a_{13} * (a_{14} - r_4)_+^0.$$

Здесь используется обозначение

$$(x - c)_+^n = \begin{cases} (x-c)^n, & \text{если } x \geqslant c, \\ 0, & \text{если } x \leqslant c. \end{cases}$$

Значения R_3 и R_4 показаны на рис. 7.7. Функция QP включает в себя 14 коэффициентов, или параметров.

Рассмотрим алгоритм, который учитывает три свойства конфигурации заданий:

f_1 = число коротких заданий (заказано не более 30 секунд центрального процессора),
f_2 = оставшееся число заданий, ожидающих своего исполнения,
f_3 = число терминалов, находящихся в данный момент в работе (они могут использоваться в разных режимах).

Выберем трехмерный вектор пространства рабочих характеристик операционной системы:

$$\mathbf{p} = (p_1,\ p_2,\ p_3)^T,$$

где p_1 = (среднее время обработки короткого пакетного задания)/ /1000,
p_2 = (среднее время обработки других пакетных заданий)/4000,
p_3 = (среднее время ответа для стандартных коротких оперативно выполняемых заданий)/10.

Масштабирование выбрано таким образом, что вектор $\mathbf{p} = (1, 1, 1)^T$ приблизительно соответствует 15 минутам среднего времени для обработки вычислительной системой короткого пакетного задания, одному часу — для других пакетных заданий и 10 секундам — для ответа оперативно выполняемых заданий. Тогда обобщенная характеристика алгоритма измеряется нормой

$$\|\mathbf{p}\| = w_1 p_1 + w_2 p_2 + w_3 p_3,$$

где w_i — элементы трехмерного пространства критериев, $w_i \geqslant 0$ и $w_1 + w_2 + w_3 = 1$.

Коэффициенты алгоритма построения расписания для заданий определяются следующим образом:

1. Оператор вычислительной машины выбирает вектор критериев **w**.
2. Операционная система измеряет свойства f_1, f_2, f_3 конфигурации заданий.
3. По w_i и f_i вычисляется наилучший вектор коэффициентов **a**.

Таким образом, мы видим, что в действительности 14 коэффициентов a_i вектора **a** являются функциями независимых переменных w_i и f_i.

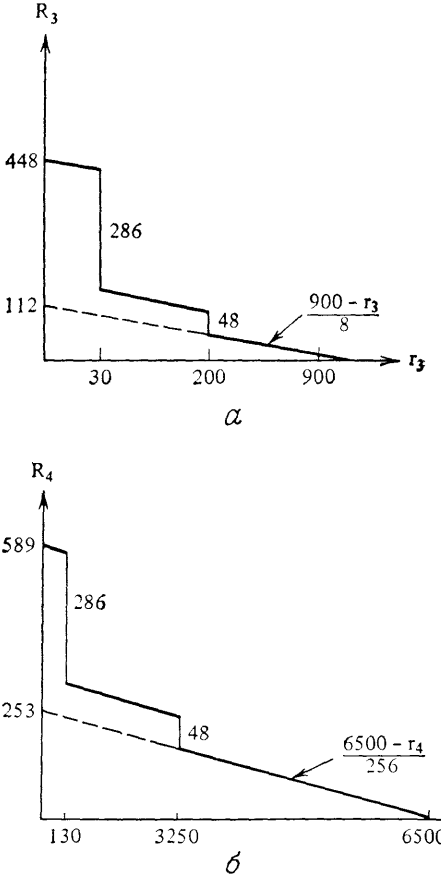


Рис 7.7 Графики функций $R_3(r_3)$ и $R_4(r_4)$. Горизонтальная ось проведена без масштаба Каждая функция кусочно-линейна
(а) Приоритет в использовании центрального процессора.
(б) Приоритет в использовании устройств ввода-вывода.

Рассмотрим теперь, как найти наилучший алгоритм расписания. Для этого представим себе идеальный ход выполнения вычислений. Основным должен быть блок вычисления наилучших a_i по заданным w_i и f_i. Этот блок определяется функцией OPT, т. е. OPT (**w**, **f**) устанавливает 14 наилучших коэффициентов по заданным векторам

w и **f**. Затем мы должны выбрать подходящие множества значений переменных w_i и f_i, скажем w_{il}, $l=1, \ldots, m_w$ и f_{ik}, $k=1, \ldots, m_f$, и выполнить операции:

For $l=1$ to m_w, $k=1$ to m_f do $a_i(l, k)=OPT(w_l, f_k)$.

На этом шаге мы составляем таблицу коэффициентов a_i как функцию w_i и f_i. Можно было бы просто проводить интерполяцию по этой таблице или, быть может, выбрать линейную форму для задачи выравнивания методом наименьших квадратов:

$$a_i = \alpha_{i0} + \sum_{j=1}^{3} (\alpha_{ij} f_j + \alpha_{i, j+3} w_j).$$

Рассмотрим случай, когда такой упрощенный подход может оказаться непригодным. Перечислим некоторые очевидные случаи (несомненно, при реальном применении этого подхода проявятся другие недостатки):

1. Функцию OPT слишком сложно вычислять. Скажем, количество вычислений функции в пределах от 50 до 200 (т. е. вычислений **p** как функции **a**) следует рассматривать как приемлемое. Необходимость выполнения более 500 или 1000 вычислений указывает на действительные трудности, а менее 50 вычислений — на наиболее удачный случай.

2. Форма, выбранная для представления QP как функции **a**, неадекватна. Однако это вряд ли имеет место, поскольку эта форма применяется в настоящее время на практике.

3. Линейная форма представления **a** как функции w_i и f_i неадекватна.

4. Отсутствует возможность изменять f_1, f_2 и f_3 в диапазоне значений, указанных в системе, и потому они непрерывно меняются и неуправляемы. Формирование конфигурации заданий по известным свойствам — весьма важная задача.

5. Измерение $\|\mathbf{p}\|$ недостоверно из-за динамической природы процесса обработки. Это означает, что в течение 15 мин, необходимых для обработки пакетного задания системой, могут быть широкие вариации в значениях **f** (из-за изменений конфигурации заданий) и значениях **a** (из-за изменений, вносимых функцией OPT).

Отметим, что трудности 2 и 3 относятся к формулировке проблемы выбора, а не к ходу вычислений, и поэтому могут быть здесь опущены. Трудности с OPT могут быть весьма реальны, однако можно надеяться, что хороший алгоритм минимизации будет обрабатывать эту часть вычислений, особенно после того, как будет получен некоторый опыт, который позволит выбирать хорошие начальные приближения. Остаются трудности 4 и 5, которые очень интересны и до некоторой степени необычны для стандартных оптимизационных задач. Они могут быть устранены, однако вопрос о том, как это дела-

ется, лежит за пределами этой книги. Дополнительные подробности см. в § 4 статьи (Rice, 1976). (См. список литературы в конце книги.)

Задачи гл. 7

1. Рассмотреть приведенную ниже программу на Фортране.

(а) Указать, какие необходимы в ней изменения, чтобы сделать ее легко переносимой на различные типы вычислительных машин; указать, какие изменения после этого все же необходимо выполнить для вашей вычислительной системы.

(б) Указать изменения, необходимые для представления программы в версии с удвоенной точностью.

```
      PROGRAM PROB1(INPUT,OUTPUT)
C
C       ЗАДАЧА 1, ГЛ. 7. ПОРТАБЕЛЬНОСТЬ
C
C          МЕТОД СЕКУЩИХ РЕШЕНИЯ УРАВНЕНИЯ
      DATA PI / 3.1415 92653 58979 /
      F(T) = .3*T*EXP(T) + COS(PI*T) - 3.
C
C                НАЧАЛО
      XSECOND = 1.4
      XFIRST = 1.2
      DO 10 J = 1,100
          BOTTOM = (F(XFIRST)-F(XSECOND))/(XFIRST-XSECOND)
C                       НЕ ДЕЛИТЬ НА НУЛЬ
          IF( ABS(BOTTOM) .LE. 1.E-14 )   GO TO 99
          XCHANGE = F(XFIRST)/BOTTOM
          XNEXT = XFIRST - XCHANGE
C                       ТЕСТ НА СХОДИМОСТЬ
          IF( ABS(XCHANGE) .LE. 5.E-14 )      GO TO 30
          XSECOND= XFIRST
          XFIRST = XNEXT
   10 CONTINUE
C                       НЕТ СХОДИМОСТИ
      PRINT 20, XFIRST,XCHANGE
   20 FORMAT(10(3H **),18H NO CONVERGENCE AT,F20.15,11HLAST STEP = ,E15.5)
      STOP
C
C                       СХОДИМОСТЬ
   30 PRINT 40, XFIRST,J
   40 FORMAT(20X,13HCONVERGED TO ,F20.15,3H IN,I3,10HITERATIONS )
      STOP
C
C                       ПОПЫТКА ДЕЛЕНИЯ НА НУЛЬ
   99 PRINT 100, XFIRST
  100 FORMAT(10(3H **),5X/ TRIED TO DIVIDE BY ZERO AT/F25.15/
     A       10(3H **),5X/ ITERATION ABORTED/)
      STOP
      END
```

(Текст в операторе 100: «Попытка деления на нуль»... «Итерации прерываются».)

2. Указать некоторые «расширения» языка Фортран, снижающие степень портабельности программ, написанных на этих диалектах языка.

3. Почему, вообще говоря, невозможно написать алгоритм на каком-либо языке высокого уровня (Фортран, Алгол, ПЛ/1, Паскаль и др.) так, чтобы он был и портабелен, и обладал наивысшей эффективностью?

4. Дать эскиз характеристического профиля хорошей программы решения систем линейных уравнений, показывающий зависимость времени исполнения программы от числа правых частей системы. Привести несколько разумных численных значений масштабов.

5 Рассмотреть новую быструю подпрограмму решения линейных систем MYLIN. Описать в деталях оценку надежности и робастности MYLIN по сравнению с какой-либо другой подпрограммой.

6. Составить множество служебных подпрограмм на Фортране, выполняющих следующие операции векторной и матричной арифметики:

(а) Подпрограмма-функция вычисления скалярного произведения двух n-векторов: ANS=DOT (X, Y, N).

(б) Подпрограмма вычисления линейной комбинации $y=ax+y$, где x, y — векторы: CALL AXY (A, X, Y, N).

(в) Подпрограмма вычисления нормы 1, 2 и ∞ n-вектора: ANS=XNORM (X, N, K), где K=1, 2, 3 указывает на нужную норму (K=3 соответствует ∞-норме).

(г) Подпрограмма вычисления произведения двух $n\times n$-матриц A=AB: CALL PROD (A, B, N).

(д) Подпрограмма вычисления произведения двух $n\times n$-матриц A=AT где T — верхняя треугольная матрица. Результат замещает матрицу A: CALL PRODT (A, T, N).

(е) подпрограмма вычисления произведения двух $n\times n$-матриц A=AR: CALL PRODR (A, R, N, M), где R имеет вид

$$\begin{pmatrix} I & 0 \\ 0 & B \end{pmatrix}$$

и B — $m\times m$-матрица.

7. Пусть необходимо составить библиотечную подпрограмму решения обыкновенного дифференциального уравнения

$$y''+a(x)y'+b(x)y=f(x), \quad x\in[\alpha, \beta], \quad y(\alpha)=y_0, \quad y(\beta)=y_1,$$

где a (x), b (x), f (x) — заданные функции, α, β — заданные границы интервала интегрирования; y_0, y_1 — краевые условия. Кроме того, задана точность вычисляемого решения задачи. Возможны два метода решения этой задачи: (а) Использовать линеаризацию конечными разностями и последовательно увеличивать число точек до тех пор, пока не будет достигнута заданная точность. (Предположить, что известен хороший способ оценки точности решения.) (б) Использовать разложение в ряд Тейлора и увеличивать число членов разложения до тех пор, пока не будет достигнута заданная точность. Описать вычислительный эксперимент сравнительной оценки двух программ, реализующих методы (а) и (б). Установить независимые переменные в этом эксперименте, величины, которые следует измерять, и критерии сравнения. Описать выбор конкретных примеров для расчетов по программам и дать приблизительную оценку трудозатрат (как человеческого, так и машинного времени), которые может потребовать этот эксперимент. Не учитывать затраты на подготовку программ для тестирования.

8. Следующим образом и следовать влияние компиляторов на время исполнения подпрограмм на Фортране.

(а) Рассмотреть две подпрограммы. Одна из них, которую вы, вероятно, составите сами, должна быть написана без ухищрений, т. е. содержать простые присваивания, циклы, проверки и т. д. Вторая — должна быть библиотечной подпрограммой из области матричного исчисления, например, решения системы линейных уравнений.

(б) Оттранслировать каждую подпрограмму на двух (или более) различных компиляторах

(в) Измерить время исполнения подпрограмм настолько точно, насколько это позволяет используемая вычислительная система. Отметим, что обычно невозможно получить точную оценку времени исполнения подпрограммы для очень коротких заданий. Следует поместить подпрограммы в цикл для многократного исполнения.

(г) Составить таблицу по примеру табл. 7.3.

9. Детально списать проектирование вычислительного эксперимента для

оценки и сравнения двух подпрограмм решения $Ax=b$, где A — ленточная матрица небольшого или среднего порядка (скажем, меньше 40). Конкретно: описать пространство задач, пространство свойств, пространство критериев оценки рабочих характеристик подпрограмм, множество фактически рассмотренных задач и какие вычислительные эксперименты предполагается выполнить.

10. Какие меры должны быть предприняты, чтобы исключить необъективность оценки рабочих характеристик подпрограмм матричного исчисления? Описать различные способы, при помощи которых можно было бы делать оценки желательным образом, без искажения экспериментальных данных.

11. Выбрать библиотечную подпрограмму решения систем линейных уравнений и получить следующие характеристические профили:

(а) Время исполнения в зависимости от порядка матрицы.

(б) Используемая память в зависимости от порядка матрицы.

(в) Время исполнения в зависимости от числа правых частей.

(г) Время исполнения в зависимости от числа нулевых элементов ленточной матрицы. Некоторые подпрограммы не учитывают преимущества, которые дает наличие нулевых элементов в матрицах.

Глава 8

КАК УЗНАТЬ, ЧТО ПОЛУЧЕНЫ ПРАВИЛЬНЫЕ ОТВЕТЫ?

В идеале окончание работы программы решения системы $\mathbf{Ax}=\mathbf{b}$ должно привести или (1) к получению правильного ответа (при условии умеренного влияния ошибок округления), или (2) к сообщению, что A — вырожденная матрица. Алгоритм исключения Гаусса (или его модификации) не может обеспечить такие результаты. Более того, матрицы бывают не просто вырожденные или невырожденные; между вырожденностью и невырожденностью существуют непрерывные вариации. В этой главе мы рассмотрим механизмы, которые, будучи включены в программу решения системы $\mathbf{Ax}=\mathbf{b}$, позволят судить, получила ли эта программа верный ответ.

К сожалению, практические задачи довольно часто приводят к очень трудным задачам матричного исчисления. Мы начнем с изучения примера из подраздела 3.А.2 и дадим три подхода к оценке точности вычисленного решения системы $\mathbf{Ax}=\mathbf{b}$.

8.А. ПОЛИНОМИАЛЬНАЯ РЕГРЕССИЯ И МАТРИЦА ГИЛЬБЕРТА

Пример 8.1. Предположим, что задан 101 результат наблюдений d_i на равномерной сетке отрезка [0, 1]:

$$(x_i, d_i) \text{ для } i=1 \text{ до } 101, \text{ где } x_i=(i-1)/100.$$

Предположим далее, что данные d_i изменяются с изменением x_i как полином степени k:

$$d_i \sim \sum_{j=0}^{k} c_j (x_i)^j = P_k (x_i).$$

Вспомним метод наименьших квадратов определения коэффициентов c_j, который минимизирует ошибку

$$\sum_i [d_i - P(x_i)]^2.$$

Как это было показано ранее (и как это опять будет рассмотрено в гл. 11), в результате минимизации суммы квадратов невязок получается система линейных уравнений порядка k+1 (система нормальных уравнений) для определения наилучших значений c_j^*

коэффициентов c_j; элементы матрицы системы имеют вид

$$a_{mn} = \sum_i x_i^{m-1} x_i^{n-1} \sim 100 \int_0^1 x^{m+n-2}\,dx = \frac{100}{m+n-1}.$$

Здесь сумма по i грубо аппроксимируется указанным интегралом. Тем самым система для определения c_j близка к системе:

$$\begin{aligned}
&c_0^* + \tfrac{1}{2}c_1^* + \tfrac{1}{3}c_2^* + \tfrac{1}{4}c_3^* + \cdots + \frac{1}{k+1}c_k^* = \Big(\sum_i d_i\Big)\Big/100\\
&\tfrac{1}{2}c_0^* + \tfrac{1}{3}c_1^* + \tfrac{1}{4}c_2^* + \tfrac{1}{5}c_3^* + \cdots + \frac{1}{k+2}c_k^* = \Big(\sum_i x_i d_i\Big)\Big/100\\
&\tfrac{1}{3}c_0^* + \tfrac{1}{4}c_1^* + \tfrac{1}{5}c_2^* + \tfrac{1}{6}c_3^* + \cdots + \frac{1}{k+3}c_k^* = \Big(\sum_i x_i^2 d_i\Big)\Big/100\\
&\quad\vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots\\
&\frac{1}{k+1}c_0^* + \frac{1}{k+2}c_1^* + \frac{1}{k+3}c_2^* + \frac{1}{k+4}c_3^* + \cdots + \frac{1}{2k+1}c_k^* = \Big(\sum_i x_i^k d_i\Big)\Big/100
\end{aligned}$$

Матрица

$$H_k = \left\{a_{mn} = \frac{1}{m+n-1}\right\}_{m,\,n=1}^{k}$$

называется *матрицей Гильберта* порядка *k*. Эта матрица известна тем, что она (или ее небольшие модификации) возникает из многих приложений и ее очень трудно использовать в практических вычислениях. Процесс решения системы с матрицей Гильберта порядка 10 (что соответствует полиному 9 степени) может (и это почти всегда имеет место) увеличить погрешность в данных d_1 на множитель, больший, чем $3*10^{12}$. Тем самым, даже если вычисления проводятся с 10 цифрами, только из-за ошибок округления погрешность в определении коэффициентов c_j^* полиномиальной аппроксимации возрастет по крайней мере в 300 раз. Более того, точность измеряемых данных редко соответствует такому большому увеличению погрешности. Другими словами, такая процедура регрессии бесполезна почти для всех случаев.

То обстоятельство, что невязка может быть достаточно мала, до некоторой степени скрывает бесполезность вычисленных результатов. Как это уже было отмечено, можно иметь маленькие невязки и большие ошибки, и это будет еще раз показано на примере в конце данной главы. Тем не менее многие пользователи вводятся в заблуждение тем, что невязки малы, скажем порядка 10^{-5}. Это наводит на мысль (в предположении, что используется арифметика с 14 десятичными цифрами), что 9 цифр потеряны в процессе вычислений и результаты вряд ли имеют смысл для задачи пользователя.

Существуют приложения, для которых достаточно получить небольшие невязки. Например, если необходимо только получить формулу, которая близко восстанавливает значения данных, то можно не обращать внимания на поведение формулы между значениями x_i и не пытаться интерпретировать c_j как параметры некоторой модели. Только небольшая часть приложений регрессии имеет такой характер.

8.Б. ЧИСЛА ОБУСЛОВЛЕННОСТИ

Первоначально число обусловленности матрицы было введено как средство *априорной* оценки того, насколько большими могут быть ошибки при решении системы $\mathbf{Ax}=\mathbf{b}$. Как мы увидим, в начале идея использования числа обусловленности была не очень надежной, однако последующие улучшения сделали ее более полезной. Главным образом эти улучшения возникли при использовании числа обусловленности для *апостериорной* оценки ошибок; т. е. сначала задача решается, а затем оценивается правильность полученного решения. При такой оценке бывает полезна различного рода информация, полученная в процессе решения системы $\mathbf{Ax}=\mathbf{b}$.

8.Б.1. Вывод трех типов чисел обусловленности

Рассмотрим три способа оценки влияния ошибок в задании исходной задачи на ее решение. Пусть имеется исходная задача $\mathbf{Ax}=\mathbf{b}$ и возмущенная задача $\mathbf{A}\bar{\mathbf{x}}=\mathbf{b}+\mathbf{r}$, где $\mathbf{r}$ описывает возмущение и правой части, и коэффициентов матрицы A. Это означает, что если возмущенная матрица имеет вид $\mathbf{A}+\delta\mathbf{A}$, то $\delta\mathbf{Ax}$ просто переносится в правую часть и включается в $\mathbf{r}$. Это не совсем правильно, поскольку $\delta\mathbf{A}$ умножается на $\mathbf{x}$, однако надо иметь в виду, что мы проводим сейчас грубые оценки. Наша цель состоит в оценке $\delta\mathbf{x}=\bar{\mathbf{x}}-\mathbf{x}$ через $\mathbf{r}$ и данные задачи, а более точно, в оценке того, как ошибка $\mathbf{r}$ в правой части системы $\mathbf{b}$ передается в $\delta\mathbf{x}$.

Прежде всего определим *естественное число обусловленности*. Имеем

$$\mathbf{Ax}=\mathbf{b}, \qquad \mathbf{A}\bar{\mathbf{x}}=\mathbf{b}+\mathbf{r}.$$

Вычитанием получим

$$\mathbf{A}(\bar{\mathbf{x}}-\mathbf{x})=\mathbf{r},$$

или

$$\bar{\mathbf{x}}-\mathbf{x}=\delta\mathbf{x}=\mathbf{A}^{-1}\mathbf{r}.$$

Возьмем норму от обеих частей этого равенства

$$\|\delta\mathbf{x}\|=\|\mathbf{A}^{-1}\mathbf{r}\|\leqslant\|\mathbf{A}^{-1}\|\,\|\mathbf{r}\|.$$

Неудивительно, что размер обратной матрицы $\mathbf{A}^{-1}$ оценивает,

насколько может увеличиться погрешность представленная **r**. Удобнее использовать относительные ошибки:

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{\|A^{-1}\|\|r\|}{\|x\|\|b\|} * \|b\| = \frac{\|A^{-1}\|\|b\|}{\|x\|} * \frac{\|r\|}{\|b\|}.$$

Это неравенство определяет естественное число обусловленности $\|A^{-1}\|\,\|b\|/\|x\|$.

Это число обусловленности называется «естественным», поскольку оно будет получено, если применить общую теорию обусловленности к частному случаю линейных уравнений. Однако оно не широко используется в линейной алгебре, вероятно, из-за желания иметь априорную оценку, а это число обусловленности зависит от $\|x\|$, которое не известно априори. Можно исключить зависимость от **x** и получить тем самым *стандартное число обусловленности*. Имеем

$$Ax=b,$$

откуда

$$\|b\| = \|Ax\| \leqslant \|A\|\|x\|,$$

или

$$\frac{1}{\|x\|} \leqslant \frac{\|A\|}{\|b\|}.$$

Подставим это неравенство в полученную ранее оценку для $\|\delta x\|/\|x\|$:

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{\|A^{-1}\|\|b\|}{\|x\|} * \frac{\|r\|}{\|b\|} \leqslant \frac{\|A^{-1}\|\|b\|\|A\|}{\|b\|} * \frac{\|r\|}{\|b\|}$$

$$= \|A^{-1}\|\|A\| * \frac{\|r\|}{\|b\|}.$$

Полученное неравенство определяет стандартное число обусловленности $\|A^{-1}\|\,\|A\|$. Отметим, что оба этих числа обусловленности представляют собой только оценки того, насколько могут быть увеличены погрешности в задании **b**. Они всегда дают завышенную оценку увеличения, иногда значительно завышенную, однако для специальных случаев они точны.

Оценки Эйрда — Линча более точны и для своего вывода требуют немного более длительных выкладок. Они дают и нижнюю, и верхнюю границы увеличения ошибки. Пусть C аппроксимирует матрицу, обратную к A (на практике C будет побочным продуктом процесса исключения Гаусса). Имеем

$$Cr=C\,(b-Ax)=CA\,(x-\bar{x}),$$

или

$$\|x-\bar{x}\| = \|(CA)^{-1}\,Cr\| \leqslant \|(CA)^{-1}\|\,\|Cr\|.$$

Используем матричное неравенство

$$\|B^{-1}\| \leqslant \frac{1}{1-\|B-I\|}, \quad \text{если } \|B-I\|<1$$

для $B=CA$.

$$\delta x \leqslant \frac{\|Cr\|}{1-\|CA-I\|}.$$

Обозначив $T=\|CA-I\|$, получим

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{\|Cr\|}{\|x\|(1-T)}.$$

На практике T обычно мало, поскольку в качестве C берется достаточно хорошее приближение к A^{-1}. Существенно, что $\|Cr\|$ не заменяется $\|C\|\,\|r\|$ как по соображениям точности, так и стоимости вычислений. Предоставим читателю в качестве упражнения вывести оценку нижней границы

$$\frac{\|Cr\|}{\|x\|(1+T)} \leqslant \frac{\|\delta x\|}{\|x\|}.$$

Заметим, что последняя оценка обеспечивает нижнюю границу ошибки в приближенном решении $\bar{x}$ через невязку $\mathbf{r}$.

8.Б.2. Вычисление чисел обусловленности

Недостаток каждой из этих оценок ошибки состоит в том, что они включают в себя A^{-1}. В нескольких случаях можно найти $\|A^{-1}\|$ без вычисления самой обратной матрицы A^{-1} (достаточно рассмотреть нормы матриц Гильберта), однако эти случаи так редки, что могут быть игнорированы. Особо рассмотрим следующую распространенную ситуацию: система $A\mathbf{x}=\mathbf{b}$ решена методом исключения Гаусса, и мы имеем приближенное решение $\bar{x}$, его невязку $\mathbf{r}$, а также множители M и U, такие, что $MA=U$. Предположив, что A—$n\times n$-матрица, оценим ошибку $\delta\mathbf{x}$ рассмотренными выше тремя способами.

Для естественного и стандартного чисел обусловленности единственная трудность состоит в вычислении $\|A^{-1}\|$. Мы имеем три альтернативы:

1. *Вычислить A^{-1} и ее норму.* Это потребует около n^3+2n^2 дополнительных операций и приблизительно в четыре раза увеличивает затраты на решение $A\mathbf{x}=\mathbf{b}$.

2. *Грубо оценить* $\|A^{-1}\|$. Заметим, что если $\mathbf{w}=A^{-1}\mathbf{y}$, то $\|\mathbf{w}\|\leqslant \leqslant\|A^{-1}\|\,\|\mathbf{y}\|$ и, следовательно, $\|A^{-1}\|\geqslant\|\mathbf{w}\|/\|\mathbf{y}\|$. Можно выбрать k векторов $\mathbf{y}_i$, $i=1, 2, \ldots, k$, решить системы $A\mathbf{w}_i=\mathbf{y}_i$ и положить

$$\|A^{-1}\| \sim \max_i \|\mathbf{w}_i\|/\|\mathbf{y}_i\|.$$

Если k мало, то это вычисление потребует около kn^2 операций, что немного по сравнению с общими трудозатратами на решение $A\mathbf{x}=\mathbf{b}$. Эвристически установлено, что если y выбирать случайным образом, то ожидаемой оценкой $\|w\|/\|y\|$ будет $^1/_2\|A^{-1}\|$. Таким образом, представляется достаточно надежным использовать небольшие значения k. Мы можем получить оценку $\|A^{-1}\|\geqslant\|\bar{x}\|/\|\mathbf{b}\|$ практически бесплатно.

3. *Использовать оценку* $||A^{-1}||$ *из пакета LINPACK.* Идея предыдущего способа усовершенствована так, что вектор **y** ищется из системы уравнений $A^T\mathbf{y}=\mathbf{e}$, где вектор **e** с компонентами ± 1 выбирается специальным образом для увеличения надежности оценки (Dongarra et al, 1979). Далее, вектор **w** ищется из системы $A\mathbf{w}=\mathbf{y}$ и $||A^{-1}||$ оценивается отношением $||w||/||y||$. Вычислительные затраты при таком подходе приблизительно соответствуют предыдущему для k=3.

В оценке Эйрда — Линча матрица C представляет собой $U^{-1}M$, где U и M определяются из треугольного разложения матрицы A. Для вычисления C**r** потребуется решить систему с другой правой частью, т. е. *около* n^2 операций. Существуют различные способы оценки $T=||CA-I||$:

а. *Вычислить матрицу CA—I и ее норму.* Пусть $S=CA-I=U^{-1}MA-I$. Тогда $US=MA-U$, где матрица MA близка к U. Таким образом, векторы-столбцы $\mathbf{s}_i$ матрицы S могут быть вычислены решением n систем уравнений

$$U\mathbf{s}_i=\mathbf{d}_i,\ i=1, 2, \ldots, n,$$

где $\mathbf{d}_i$ — i-й столбец матрицы $D=MA-U$. Общий объем вычислений для этого процесса составляет: $n^3/2$ операций на формирование MA, $n^2/2$ — на вычитание U и $n^3/2$ — на решение n систем уравнений с верхней треугольной матрицей. Чтобы вычислить нормы ∞ или 1 матрицы CA—I, потребуется еще n^2 операций. Тем самым этот способ требует дополнительно n^3 операций и приблизительно учетверяет общий объем трудозатрат на решение системы $A\mathbf{x}=\mathbf{b}$.

б. *Грубо оценить* $||CA-I||$. Матрица $D=MA-U$ должна быть нулевой матрицей, а в действительности отличается от нулевой матрицы только в силу естественного влияния ошибок округления. Тем самым она должна быть случайной матрицей с компонентами, пропорциональными уровню ε ошибок округления в вычислительной машине. Отсюда следует, что $||S||=||U^{-1}D||$ становится большой, если только $||U^{-1}||$ становится большой. Таким образом,

$$T=||S||\leqslant||U^{-1}||*c\varepsilon,$$

где c — «небольшая» функция от n (скажем, $\sqrt{n}$). Норму $||U^{-1}||$ можно оценить, применяя один из двух способов 2 или 3, рассмотренных для оценки $||A^{-1}||$. При этом будет достигнута экономия в два раза, поскольку U — верхняя треугольная матрица. Более того, T играет гораздо менее существенную роль для этой оценки ошибок, чем $||A^{-1}||$ играет для естественного и стандартного чисел обусловленности. Трудозатраты этого способа оценки T пропорциональны n^2.

в. *Положить T=1/2.* Это, конечно, крайне грубая оценка, основанная на том, что множитель 1/(1—T) имеет второстепенную

важность. Если T больше 1/2 (в этом случае A почти вырождена), то норма ||Cr|| велика и неточность в решении уже известна. Этот способ, конечно, очень дешев и легок для программирования.

Резюмируя, можно сказать, что для каждого из трех способов вычисления чисел обусловленности существует прямая зависимость между надежностью и трудозатратами. Далее, объем трудозатрат для каждого из них сравним для заданного уровня грубости в вычислении оценок.

8.В. АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ

Идея, лежащая в основе анализа чувствительности, весьма проста и имеет довольно общий характер: решить задачу с несколько разными данными и посмотреть, насколько чувствительно решение к изменению в данных. Для программы решения $Ax=b$ существуют два естественных способа применения этой идеи.

1. *Возмущение правой части.* Вычисляется решение $\bar{x}$ системы $Ax=b$, затем выбираются r_i и решаются системы

$$Ax_i=b+r_i,\ i=1, 2, \ldots, k.$$

Пусть $\bar{x}_i$— вычисленные решения этих систем. Тогда

$$\max_i \|\bar{x}-\bar{x}_i\|$$

оценивает чувствительность вычисленного решения $\bar{x}$ к возмущениям r_i. Значения r_i следует выбирать малыми по сравнению с b, а именно: порядка величины ошибок округления, если исследуется их влияние на решение, или порядка величины погрешности в исходных данных, если они присутствуют.

Этот анализ не трудоемок для нескольких возмущений, поскольку дополнительные решения вычисляются на основе разложения матрицы A, выполненного при нахождении первого решения.

2. *Возмущение матрицы A и правой части* b. Выбираются матрицы E_i и векторы r_i, имеющие малые компоненты по сравнению с соответствующими компонентами A и b, и решаются системы

$$(A+E_i)\bar{x}_i=b+r_i.$$

Опять-таки

$$\max_i \|\bar{x}-\bar{x}_i\|$$

оценивает чувствительность решения $\bar{x}$. Этот способ более трудоемок, поскольку не используется информация, полученная при решении предыдущих систем. Заметим также, что вычисление $\bar{x}_i$ из системы

$$A\bar{x}_i=b+r_i-E_iy_i$$

дает тот же результат для оценки чувствительности, однако такой способ имеет смысл только в предположении, что вектор $E_i y_i$ сравним по величине компонент с вектором r_i. Конечно, вектор y_i не известен заранее, так что полные возмущения дают бóльшую уверенность в анализе чувствительности.

Существует легкий способ выполнить анализ чувствительности решения к ошибкам округления. Для этого достаточно решить одну и ту же задачу, но программой, оттранслированной на разных компиляторах. Различные компиляторы почти наверняка дают объектные программы, выполняющие арифметические действия в другом порядке, и, следовательно, эти программы дают различные ошибки округления.

8.Г. ИТЕРАЦИОННОЕ УТОЧНЕНИЕ

Итерационное уточнение представляет собой метод уточнения уже полученного решения. Пусть $\bar{x}$ — вычисленное решение системы $Ax=b$, а $r=b-A\bar{x}$ — его невязка. Имеем

$$Ax=b, \quad A\bar{x}=b-r$$

и, положив $y=x-\bar{x}$, имеем

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

Можно решить последнюю систему для вычисления поправки $y=x-\bar{x}$ для $\bar{x}$ (обозначим через $\bar{y}$ вычисленное решение этой системы). Это легко сделать, поскольку матрица A уже факторизована. Здесь можно рассуждать следующим образом: Пусть матрица A такова, что могут быть получены только три верные цифры при решении систем с этой матрицей. Поэтому $\bar{x}+\bar{y}$ будет иметь уже шесть верных цифр: три цифры из начального решения и еще три цифры из поправки. Однако эти рассуждения ошибочны и такие вычисления вовсе не дают уточнения. Обычно $\bar{x}+\bar{y}$ обладает не большей точностью, чем $\bar{x}$. Чтобы убедиться в этом, рассмотрим частный случай, предположив, что

1. Все числа в A, x и b порядка 1.0.
2. Используется восьмизначная десятичная арифметика.
3. При решении системы $Ax=b$ теряются пять цифр, и тем самым $\bar{x}$ имеет три верные цифры.

Рассмотрим цифры в записи вычисленных чисел, обозначив знаком x верные цифры, а знаком z — неверные цифры. Выделим пробелами в последовательностях цифр те цифры, которые определяют

существо дела. После вычисления $\bar{x}$ и **r** числа примут следующий вид:

числа в A и **b**: .xxx xxxxx
числа в $\bar{x}$: .xxx zzzzz
числа в **r** : .000 xxxxx zzz

Таким образом, первые пять цифр невязки **r**, соответствующие неверным цифрам в $\bar{x}$, являются верными, а последние три цифры являются неверными в силу выбранных ограничений на арифметику. Эти три цифры не могут содержать информации, поскольку соответствующие цифры в $\bar{x}$ опущены. Когда мы вычисляем $\bar{y}$, мы теряем пять цифр; это означает, что неопределенность в цифрах вектора $\bar{y}$ сдвигается на пять позиций влево, поэтому числа в $\bar{y}$ имеют следующий вид: 000 zzzzz zzz. Эти цифры все неверные, и тем самым прибавлением $\bar{y}$ мы ничего не достигаем.

Этот подход к уточнению решения может быть использован, *если невязка* **r** *вычисляется более точно.* Рассмотрим еще раз этот пример, предположив, что для вычисления **r** используется десятизначная десятичная арифметика.

числа в A и **b**: xxx xxxxx
числа в $\bar{x}$: .xxx zzzzz
числа в **r** : .000 xxxxx xxzzz
числа в $\bar{y}$: .000 xxzzz zzz
числа в $\bar{x}+\bar{y}$: .xxx xxzzz

Будем считать, что перед вычислением **r** числа в A и **b** приводятся к виду .xxx xxxxx00, а числа в $\bar{x}$ — к виду .xxx zzzzz00. Десятизначные числа в **r** будут округлены (или усечены) до восьмизначных перед решением системы для A**y**=**r**.

На практике расширенная точность почти всегда означает удвоенную точность, как в примерах, приведенных ниже в этой главе. Второй факт состоит в том, что если поправка $\bar{y}$ не меняет $\bar{x}$ (т. е. $\bar{y}$ сравним с ошибками округления в $\bar{x}$), то $\bar{x}$ рассматривается как «правильный» ответ. Заметим, однако, что окончательная невязка (вычисленная с одинарной точностью) в действительности может быть больше первоначальной. Метод исключения Гаусса дает небольшую ошибку невязки, даже если получено не очень точное решение.

Алгоритм 8.1: Итерационное уточнение решения системы A**x**=**b**

1. Найти матрицы M и U, такие, что MA = U.
2. Найти **x** решением двух систем **v** = M**b** и U**x** = **v**.
3. Для $k = 1, 2, \ldots, k_{max}$ выполнить
   3.1. Вычислить **r** = **b** — A**x** в режиме удвоенной точности.

3.2. Найти y решением двух систем $\mathbf{v}=M\mathbf{r}$ и $U\mathbf{y}=\mathbf{v}$.
3.3. Если отношение норм $\|\mathbf{y}\|/\|\mathbf{x}\|$ удовлетворительно, то перейти на КОНЕЦ.
3.4. $\mathbf{x}=\mathbf{x}+\mathbf{y}$

Если цикл завершен, а отношение норм неудовлетворительно, то сходимость не достигнута и алгоритм потерпел неудачу.

4 Напечатать: «Неудачное итерационное уточнение, последняя оценка решения $=\mathbf{x}$».

5 КОНЕЦ

Обычно отношение $||\mathbf{y}||/||\mathbf{x}||$ считается удовлетворительным, если его величина имеет порядок ошибок округления. Поэтому, для вычислений в t-значной десятичной арифметике мы можем проверять оценку

$$\frac{\|\mathbf{y}\|}{\|\mathbf{x}\|} \leqslant 10^{-t}.$$

Машинно-независимый способ осуществления этой проверки состоит в проверке равенства

$$||\mathbf{x}||+||\mathbf{y}||=||\mathbf{x}||.$$

Число обусловленности матрицы A может быть грубо оценено подсчетом числа итераций, требуемых для сходимости процесса итерационного уточнения. Пусть t — число десятичных цифр, используемых в арифметических операциях, а s — число верных цифр, полученных при решении $A\mathbf{x}=\mathbf{b}$. Тогда, если $s\geqslant t/2$, то потребуется только одна итерация, чтобы получить правильный ответ, плюс одна дополнительная итерация, чтобы проверить, что полученный ответ правильный. Только, если $s=t$, процесс итерационного уточнения завершается сразу. Если $t/2>s\geqslant t/3$, то правильный результат получается за две итерации и требуется одна дополнительная итерация на проверку результата. В общем случае имеем

Число итераций	1	2	3	...	k
Интервал значений s	$s=t$	$s\geqslant t/2$	$t/2>s\geqslant t/3$	...	$t/(k-1)>s\geqslant t/k$

Таким образом, число обусловленности грубо оценивается числом $10^{t(1-1/k)}$.

8.Д. СРАВНЕНИЕ ПРОЦЕДУР ОЦЕНКИ ОШИБОК

Искусство проектирования программ для решения системы уравнений $A\mathbf{x}=\mathbf{b}$ включает в себя выбор комбинации методов оценки ошибок (и их многочисленных вариаций). Выбор должен обеспе-

чить и эффективность, и надежность. В этом разделе мы проведем сравнение этих методов на нескольких примерах. Прежде всего мы перечислим матрицы, которые могут вызывать затруднения при решении системы $\mathbf{Ax}=\mathbf{b}$.

8.Д.1. Некоторые матрицы, чувствительные по отношению к вычислениям

Дадим перечень 13 матриц, обладающих различным уровнем чувствительности. Обозначим элементы матрицы A через a_{ij}, а порядок A — через n. Заметим, что матрицы со случайно сгенерированными элементами не образуют подходящего множества для проверки чувствительности; почти все такие матрицы отличаются очень хорошим поведением.

1. *Матрица Гильберта:* $a_{ij}=1/(i+j-1)$.

2. $a_{ii}=i$ для $i=1, \ldots, 20$, $a_{i,\,i+1}=20$ для $i=1, \ldots, 19$, $a_{ij}=0$ для других значений i и j. Эта матрица особенно трудна для вычисления собственных значений.

3. $a_{ij}=\min[i, j]$.

4. *Pei-матрица* (с параметром α): $a_{ii}=\alpha$, $a_{ij}=1$ для $i\neq j$. Трудности имеют место для α, близких к 1, или для α, близких к $n-1$.

5.

$$\begin{pmatrix} 5 & 4 & 7 & 5 & 6 & 7 & 5 \\ 4 & 12 & 8 & 7 & 8 & 8 & 6 \\ 7 & 8 & 10 & 9 & 8 & 7 & 7 \\ 5 & 7 & 9 & 11 & 9 & 7 & 5 \\ 6 & 8 & 8 & 9 & 10 & 8 & 9 \\ 7 & 8 & 7 & 7 & 8 & 10 & 10 \\ 5 & 6 & 7 & 5 & 9 & 10 & 10 \end{pmatrix}$$

6. Матрицы с параметром θ:

$$R=\begin{pmatrix} \operatorname{ctg}\theta & \operatorname{cosec}\theta \\ -\operatorname{cosec}\theta & \operatorname{ctg}\theta \end{pmatrix}, \qquad S=\begin{pmatrix} 1-\operatorname{ctg}\theta & \operatorname{cosec}\theta \\ -\operatorname{cosec}\theta & 1+\operatorname{ctg}\theta \end{pmatrix}.$$

Собственные значения матрицы R равны $\pm i$, а матрицы $S-1\pm i$, где i — мнимая единица.

7. $a_{ii}=0.01/[(n-i+1)(i+1)]$, $a_{ij}=0$ для $i<j$, $a_{ij}=i(n-j)$ для $i>j$.

8. Матрица из п. 7, но $a_{ij}=j(n-i)$ для $i<j$.

9. Матрица

$$\begin{pmatrix} R & S & T & T \\ S & R & S & T \\ T & S & R & S \\ T & T & S & R \end{pmatrix}$$

где R и S из п. 6, а $T=\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Трудности имеют место, когда θ близко к 0 или π.

10. Матрица с параметром α:

$$a_{ii}=\alpha^{|n-2i|/2}, \quad a_{1j}=a_{11}/\alpha^{j}, \quad a_{nj}=a_{nn}/\alpha^{j},$$
$$a_{j1}=a_{1j}, \quad a_{jn}=a_{nj}, \quad a_{ij}=0 \text{ для остальных } i \text{ и } j.$$

11. $a_{ij}=e^{i*j*h}$. Трудности имеют место, когда h близко к нулю или когда h — велико (положительное или отрицательное).

12. $a_{ij}=c+\log_2(i*j)$. Трудности имеют место для всех значений c, но особенно, когда c — велико.

13.

$$\begin{pmatrix} 0.9143\times10^{-4} & 0 & 0 & 0 \\ 0.8762 & 0.7156\times10^{-4} & 0 & 0 \\ 0.7943 & 0.8143 & 0.9504\times10^{-4} & 0 \\ 0.8017 & 0.6123 & 0.7165 & 0.7123\times10^{-4} \end{pmatrix}$$

8.Д.2. Сравнение оценок ошибок на основе чисел обусловленности

Покажем на четырех примерах десять способов оценок ошибок, основанных на числах обусловленности. Первым примером является матрица Гильберта десятого порядка, а вторым примером — небольшая система с треугольной матрицей. Третий пример — 14×14-матрица относительно хорошего поведения, имеющая следующий вид:

$$\begin{pmatrix} A & B \\ C & A \end{pmatrix},$$

где A — 7×7-матрица из п. 5 подраздела 8.Д.1 и

$$b_{ij} = a_{8-ij}$$

$$C = \begin{pmatrix} \frac{1}{8} & \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 5 \\ \frac{1}{9} & \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 6 & \frac{1}{15} \\ \frac{1}{10} & \frac{1}{11} & \frac{1}{12} & \frac{1}{13} & 7 & \frac{1}{15} & \frac{1}{16} \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Последний пример представляет собой 40×40-матрицу с элементами, случайно сгенерированными так, что ее число обусловленности приблизительно 10^8. Первые ее 36 строк являются случайными векторами, а последние четыре строки — случайные линейные комбинации первых 36 строк плюс другой случайный вектор, умноженный на 10^{-5}. Конкретно, матрица A формируется следующим фрагментом программы [в предположении, что RAND (0) генерирует случайные числа на отрезке (0, 1)]:

```
      INDEP  = 36
      DEPEND=4
      EPS    = 00001
      NEQS   = INDEP + DEPEND
      DO 10 I = 1, INDEP
          DO 10 J = 1,NEQS
              A(I,J) = 2 *(RAND(0)- 5)
10    CONTINUE
      DO 20 K = 1, INDEP
20        COEF(K) = RAND(0)
      DO 40 I = INDEP+1,NEQS
          DO 40 J = 1,NEQS
              SUM = 0 0
              DO 30 K = 1,INDEP
30                SUM = SUM + COEF(K) * A(K,J)
              A(I J) = SUM + EPS*(RAND(0)- 5)
40    CONTINUE
```

Для каждой матрицы мы зададим вектор правой части **b** и приведем значение точного решения **x** системы $A\mathbf{x}=\mathbf{b}$, полученное с удвоенной точностью программой LINEQ2 (см. приложение), а также вычисленное, или пробное, решение $\bar{\mathbf{x}}$ этой системы. Наша цель состоит в оценке ошибки $\|\mathbf{x}-\bar{\mathbf{x}}\|$. Мы приведем также значения некоторых основных численных характеристик, связанных с каждой задачей; все используемые нормы являются ∞-нормой. Приводятся также временны́е характеристики программ, решающих системы $A\mathbf{x}=\mathbf{b}$.

Все вычисления проводились на вычислительной машине CDC 6500 в Университете Пэрдью, которая с одинарной точностью вы-

полняет действия с 48 двоичными разрядами (около 14,5 десятичных цифр). Даются оценки ошибок Эйрда — Линча, а также оценки, основанные на естественных стандартных числах обусловленности, причем каждая из этих оценок вычисляется тремя различными способами. Эти три способа обозначаются А, Б и В: А указывает на дешевый способ вычисления оценок ошибок, В указывает на то, что проводятся тщательные вычисления, а Б — некоторый промежуточный способ.

Конкретно, А указывает, что $||A^{-1}||$ оценивается отношением $||b||/||x||$ и используется $T=1/2$. Б указывает, что $||A^{-1}||$ для естественного числа обусловленности оценивается с использованием двух случайных векторов (альтернатива 2 из подраздела 8.Б.2), $||A^{-1}||$ для стандартного числа обусловленности оценивается по алгоритму из пакета LINPACK (альтернатива 3 из подраздела 8.Б.2), а $||CA-I||$ для оценки Эйрда — Линча оценивается с использованием двух случайных векторов (альтернатива 3b из подраздела 8.Б.2 с использованием альтернативы 2 для оценки $||U^{-1}||$). В указывает, что все величины вычисляются непосредственно невзирая на затраты.

В конце приложения кратко описывается программа BOUND. Она использует интервальную арифметику, чтобы получить границы ошибок, и оценки, полученные с ее помощью, помечаются как A/L BOUNDS.

В дополнение к оценке ошибок, мы приведем данные о затратах (в секундах) на вычисление оценки ошибок и об относительных затратах, которые представляют собой отношение времени вычисления оценки ошибок к времени решения $Ax=b$. И наконец, мы приведем данные о количестве *потерянных цифр*, которое определяется как

$$\log_{10}\left[\frac{\text{оцененная относительная ошибка}}{\text{действительная относительная ошибка}}\right]$$

Эта характеристика представляет собой число десятичных цифр, которые оценка ошибок теряет в правильном результате. Три потерянных знака показывают, что ошибка в $1000=10^3$ раз больше.

Пример 8.2.: Матрица Гильберта десятого порядка.

$$b = (1, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$$

$$x = (99\,9989793495089, -494\,995439565316, \ldots, -923884\,077853199)^T$$

ПОЛУЧЕНО LINEQ2

$$\bar{x} = (100\,005711648351, -495.057320082578, \ldots, -924185.572651327)^T$$

ПОЛУЧЕНО LINPACK

Основные численные характеристики задачи:

$$\|A\| = 2\,9290 \qquad \|\bar{x}\| = 9\,6129 * 10^{6} \qquad \|x\| = 9\,6103 * 10^{6}$$

$$\|A^{-1}\| = 1\,2074 * 10^{13} \qquad \|r\| = 5\,1223 * 10^{-9} \qquad \frac{\|x - \bar{x}\|}{\|x\|} = 2\,7700 * 10^{-4}$$

$$\|b\| = 1\,0 \qquad \|x - \bar{x}\| = 266\,21$$

Время решения системы $Ax = b$ составляет 0.024 секунды.

Таблица 8.2

Сводка данных о проверках оценок ошибок

Использованная оценка	Оценка ошибок	Затраты в сек	Потерянные цифры	Относительные затраты
Естественная-А	4.9240E—02	0.002	2.2498	0.08333
Естественная-Б	1.9230E—03	0.014	0.8415	0.58333
Естественная-В	6.4337E—03	0.419	1.3660	17.47917
Стандартная-А	1.4422E—01	0.003	2.7165	0.14583
Стандартная-Б	1.3730E+05	0.020	8.6952	0.83333
Стандартная-В	1.8115E+05	0.421	8.8155	17.54167
Эйрда—Линча-А	5.3832E—04	0.014	0.2886	0.58333
Эйрда—Линча-Б	2.8506E—04	0.044	0.0125	1.83333
Эйрда—Линча-В	2.6926E—04	0.044	—0.0123	1.83333
A/L BOUNDS	4.3471E—02	1.310	2.1957	54.58333

Пример 8.3: Пример Уилкинсона треугольной системы четвертого порядка

$b = (0\,00009143,\ 0\,87627156,\ 1\,60869504,\ 2\,13057123)^{T}$

$x = (1\,0,\ 0\,999999999979416,\ 1\,00000017635355,\ 0\,998226242846176)^{T}$

ПОЛУЧЕНО LINEQ2

$\bar{x} = (1\,0,\ 1\,0,\ 1\,0,\ 1\,0)^{T}$ ТОЧНОЕ РЕШЕНИЕ

Основные численные характеристики задачи:

$$\|A\| = 2\,1306 \qquad \|\bar{x}\| = 1\,0000 \qquad \|x\| = 1\,0000$$

$$\|A^{-1}\| = 1\,1541 * 10^{16} \qquad \|r\| = 1\,4566 * 10^{-14} \qquad \frac{\|x - \bar{x}\|}{\|x\|} = 0.0017738$$

$$\|b\| = 2.1306 \qquad \|x - \bar{x}\| = 0.0017738$$

Время решения системы $\mathbf{Ax}=\mathbf{b}$ составляет 0.005 секунды.

Таблица 8.3

Сводка данных о проверках оценок ошибок

Использованная оценка	Оценка ошибок	Затраты в сек	Потерянные цифры	Относительные затраты
Естественная-А	6.8367E—15	0	—11.4140	0
Естественная-Б	2.1110E+02	0.004	5.0756	0.80000
Естественная-В	1.6811E+02	0.020	4.9767	3.90000
Стандартная-А	6.8367E—15	0.001	—11.4140	0.10000
Стандартная-Б	1.9505E+02	0.007	5.0412	1.40000
Стандартная-В	1.6811E+02	0.020	4.9767	4.00000
Эйрда—Линча-А	3.5271E—07	0.003	—3.7015	0.60000
Эйрда—Линча-Б	3.1024E+03	0.007	9.2428	1.40000
Эйрда—Линча-В	1.7814E—07	0.007	—3.9981	1.40000
A/L BOUNDS	4.4114E—02	0.080	1.3957	16.00000

Пример 8.4: Матрица 14 порядка с хорошим поведением

b_i = СУММА a_{ij}, $1 \le j \le 14$

$\mathbf{x}$ = (0 999999999999996, 1 00000000000000, , 0 999999999999996)^T

ПОЛУЧЕНО LINEQ2

$\bar{\mathbf{x}}$ = (0 999999999999925 0 999999999999908, , 0 999999999998739)^T

ПОЛУЧЕНО LINPACK

Основные численные характеристики задачи:

$$\|\mathbf{A}\| = 100 \qquad \|\bar{\mathbf{x}}\| = 1 \qquad \|\mathbf{x}\| = 1$$

$$\|\mathbf{A}^{-1}\| = 13\,983 \qquad \|\mathbf{r}\| = 2\,2737 * 10^{-12} \qquad \frac{\|\mathbf{x}-\bar{\mathbf{x}}\|}{\|\mathbf{x}\|} = 1\,4495 * 10^{-12}$$

$$\|\mathbf{b}\| = 100 \qquad \|\mathbf{x}-\bar{\mathbf{x}}\| = 1\,4495 * 10^{-12}$$

Время решения системы $A\mathbf{x}=\mathbf{b}$ составляет 0.048 секунды.

Таблица 8.4

Сводка данных о проверках оценок ошибок

Использованная оценка	Оценка ошибок	Затраты в сек	Потерянные цифры	Относительные затраты
Естественная-А	2.2737E — 14	0.003	—1.8045	0.06250
Естественная-Б	1.6893E — 11	0.021	1.0665	0.43750
Естественная-В	3.1794E — 11	0.581	1.3411	12.09375
Стандартная-А	2.2737E — 14	0.006	—1.8045	0.13542
Стандартная-Б	9.2643E — 12	0.027	0.8056	0.56250
Стандартная-В	3.1794E — 11	0.584	1.3411	12.16667
Эйрда — Линча-А	2.5224E — 12	0.023	0.2406	0.47917
Эйрда — Линча-Б	1.2612E — 12	0.092	—0.0604	1.91667
Эйрда — Линча-В	1.2612E — 12	0.092	—0.0604	1.91667
A/L BOUNDS	2.8492E — 11	2.484	1.2935	51.75000

Пример 8.5: Случайная матрица с 36 линейно-независимыми строками и 4 возмущенными линейно-зависимыми строками (возмущение порядка 10^{-5})

$$\mathbf{b} = (1, 1, \ldots, 1)^T$$

$$\mathbf{x} = (-1.83181002028004 * 10^6, \ldots)^T \text{получено LINEQ2}$$

$$\bar{\mathbf{x}} = (-1.83181001071588 * 10^6, \ldots)^T \text{получено LINPACK}$$

Таблица 8.5

Сводка данных о проверках оценок ошибок

Использованная оценка	Оценка ошибок	Затраты в сек.	Потерянные цифры	Относительные затраты
Естественная-А	5.2926E + 00	0.024	8.5394	0.04372
Естественная-Б	1.0631E — 06	0.105	1.8423	0.19126
Естественная-В	4.0094E — 06	13.189	2.4188	24.02277
Стандартная-А	3.9191E + 02	0.048	10.4089	0.08652
Стандартная-Б	2.0694E + 02	0.153	10.1316	0.27869
Стандартная-В	1.0338E + 03	13.212	10.8302	24.06557
Эйрда — Линча-А	1.8295E — 08	0.152	0.0781	0.27687
Эйрда — Линча-Б	9.1474E — 09	0.983	—0.2230	1.79053
Эйрда — Линча-В	9.1474E — 09	0.983	—0.2230	1.79053
A/L BOUNDS	1.5030E — 06	55.929	1.9927	101.87432

Таблица 8.6

Использование анализа чувствительности для оценки ошибок в примерах 8.2—8.5[1])

Способ анализа чувствительности	Количество возмущений	Оценка ошибок для примеров				Относительные затраты для примеров			
		8.2	8.3	8.4	8.5	8.2	8.3	8.4	8.5
Возмущение одной только правой части b на 10^{-7}	1	2.7E+3	1.0E+5	4.7E—5	2.0E+3	0.29	0.6	0.25	0.12
	2	2.7E+3	1.0E+5	4.7E—5	2.0E+3	0.58	1.2	0.50	0.23
	3	2.7E+3	1.8E+5	4.7E—5	2.0E+3	0.88	1.8	0.75	0.35
	5	2.7E+3	1.8E+5	4.7E—5	2.0E+3	1.50	3.0	1.00	0.59
	10	2.7E+3	1.8E+5	4.7E—5	2.0E+3	2.90	6.0	2.00	1.17
Возмущение A и b на 10^{-7}	1	4.6E+3	5.4E+4	1.3E—5	2.9E+3	1.25	1.2	1.1	1.00
	2	3.0E+4	7.6E+4	1.3E—5	2.9E+3	2.50	2.4	2.1	2.1
	3	3.0E+4	7.6E+4	3.1E—5	2.9E+3	3.75	3.6	3.2	3.1
	5	3.0E+4	1.5E+5	3.1E—5	2.9E+3	6.25	6.0	5.3	5.2
	10	3.3E+4	1.5E+5	3.4E—5	2.9E+3	12.50	12.0	10.7	10.5

Примечание. Оценки ошибок для примера 8.5 одни и те же, потому что (1) возмущение вектора b действительно давало одни и те же результаты для всех возмущений и (2) при всех возмущениях вектора b и матрицы A имел место тот факт, что наибольшая оценка ошибок получалась именно для первого произведенного возмущения (в случае 10 возмущений оценки ошибок располагались достаточно случайным образом в пределах от 6 9E+2 до 2.9E+3).

[1]) Приводятся затраты на вычисление оценок ошибок относительно затрат на решение системы $Ax=b$; относительные затраты растут линейно с ростом количества возмущений, тогда как польза от проведения дополнительных возмущений быстро уменьшается.

Основные численные характеристики задачи:

$$\|A\| = 74\,049 \qquad \|x\| = 3\,4822 * 10^{6} \qquad \|x\| = 3\,4822 * 10^{6}$$

$$\|A^{-1}\| = 9\,1857 * 10^{6} \qquad \|r\| = 1\,5199 * 10^{-6} \qquad \frac{\|x - \bar{x}\|}{\|x\|} = 1\,5285 * 10^{-8},$$

$$\|b\| = 1 \qquad \|x - \bar{x}\| = 0\,053225$$

Время решения системы $Ax=b$ составляет 0.549 секунды.

8.Д.3. Сравнение анализа чувствительности и итерационного уточнения

Здесь рассматриваются те же самые четыре примера из предыдущих сравнений (табл. 8.6). Для каждого примера (1) возмущается правая часть на 10^{-7}, (2) возмущается матрица и правая часть на 10^{-7} и (3) применяется итерационное уточнение. Результаты используются для оценки полученной точности, а также приводится время вычислений.

Если предположить, что возмущение на 10^{-14} вместо 10^{-7} должно просто уменьшить оценки ошибок на множитель 10^{-7}, мы можем сравнить оцененные ошибки, данные в табл. 8.6, с фактическими ошибками для этих четырех примеров:

Таблица 8.7

	Примеры			
	8.2	8.3	8.4	8.5
Фактическая ошибка	2.8E—4	1.8E—3	1.4E—12	1.5E—8
Возмущение b на 10^{-14}	2.7E—4	1.8E—2	4.7E—12	2.0E—4
Возмущение A и b на 10^{-14}	3.3E—4	1.5E—2	3.4E—12	2.9E—4

Табл. 8.8 показывает эффект итерационного уточнения на точность решения системы $Ax=b$. Эти вычисления выполнены подпрограммой LINEQ1 (которая описана в конце приложения). Некоторые сравнительные данные приводятся для пакета LINPACK, которые незначительно отличаются от предыдущих данных для этого пакета, полученных для тех же самых примеров, поскольку использовался другой компилятор.

Таблица 8.8

Применение подпрограммы итерационного уточнения LINEQ1 для предыдущих примеров

	Примеры			
	8.2	8.3	8.4	8.5
LINEQ1:				
Первоначальная ошибка	$9.96*10^{1}$	$4.74*10^{-7}$	$4.90*10^{-13}$	$4.42*10^{-1}$
Окончательная ошибка	$5.96*10^{-9}$	$7.10*10^{-15}$	$7.10*10^{-15}$	$1.19*10^{-7}$
Первоначальная невязка	$3.07*10^{-9}$	$7.22*10^{-15}$	$3.91*10^{-13}$	$4.81*10^{-7}$
Окончательная невязка	$1.67*10^{-9}$	$5.40*10^{-15}$	$3.26*10^{-13}$	$5.59*10^{-7}$
Число итераций	3	2	2	3
Оценка числа обусловленности	10^{9}	10^{7}	10^{7}	10^{9}
Относительные затраты	1.00	1.25	0.48	0.26
LINPACK:				
Первоначальная ошибка	$1.17*10^{3}$	$4.65*10^{-4}$	$6.61*10^{-13}$	$3.81*10^{0}$
Первоначальная невязка	$8.38*10^{-9}$	$4.73*10^{-15}$	$3.24*10^{-12}$	$1.34*10^{-5}$

Примечание Для сравнения даны некоторые результаты для пакета LINPACK. Они получены программами, оттранслированными на компиляторе, отличном от использованного в предыдущих примерах, что привело к изменению эффекта ошибок округления

Из табл. 8.8 видны три интересных момента: (1) Величина невязки не меняется заметно, даже если достигнуто значительное улучшение точности вычисленного решения. (2) Итерационное уточнение стóит довольно дорого; затраты на уточнение превосходят на 50% затраты на обычное решение системы с 14×14-матрицей, обладающей хорошими свойствами, и на 25% — для системы с 40×40-матрицей, обладающей неплохими свойствами. (3) Оценка числа обусловленности почти бесполезна. Нет готового объяснения тому факту, что ошибки в решениях, полученных подпрограммой LINEQ1 без итерационного уточнения, соответственно лучше ошибок в решениях, полученных пакетом LINPACK; этот факт имел место для всех 10 просчитанных примеров, за исключением трех случаев, когда были достигнуты одинаковые результаты.

Задачи гл. 8

1. Пусть решается система $\mathbf{Ax}=\mathbf{b}$. Правая часть $\mathbf{b}$ имела возмущение $\delta\mathbf{b}$, так что фактически было найдено решение $\mathbf{y}$ системы $\mathbf{Ay}=(\mathbf{b}+\delta\mathbf{b})$. Найти оценку

$\|x-y\|$, выраженную через δb.

2. Рассмотреть

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

(а) Каково стандартное число обусловленности матрицы A?

(б) Пусть решается система $Ax=b$ и полученное решение $\bar{x}$ таково, что $\|b-A\bar{x}\|\leqslant 0.01$. Насколько малой может быть дана верхняя граница абсолютной ошибки $\|\bar{x}-x\|$?

(в) В предположениях пункта (б) этой задачи определить, насколько малой может быть дана верхняя граница относительной ошибки $\|\bar{x}-A^{-1}b\|/\|A^{-1}b\|$?

3. Пусть λ_1 и λ_n — наименьшее и наибольшее по абсолютной величине собственные значения $n\times n$-матрицы A.

(а) Показать, что $|\lambda_n/\lambda_1|\leqslant\|A\|\,\|A^{-1}\|$.

(б) Собственные значения матрицы A в задаче 2 равны 0.05888, 0.2007 и 84.74. Насколько хорошо отношение собственных значений оценивает стандартное число обусловленности матрицы A (использовать $\|A\|_\infty\|A^{-1}\|_\infty$)?

(в) Показать, что $|\lambda_n/\lambda_1|=1$ для ортогональной матрицы.

(г) Дать пример матрицы, у которой все собственные значения равны по абсолютной величине 1, но число обусловленности все же велико.

4. (а) Показать, что для любой матрицы P, $\|P\|<1$, имеет место оценка

$$\|(I-P)^{-1}\|\leqslant\frac{1}{1-\|P\|}.$$

(б) Применить оценку из пункта (а) этой задачи для того, чтобы показать, что

$$\|\bar{x}-x\|<\frac{\|Cr\|}{1-\|CA-I\|},$$

если C представляет собой настолько хорошую аппроксимацию матрицы, обратной к A, что $\|CA-I\|<1$. Вектор r является невязкой $Ax-A\bar{x}$.

(в) Пусть при помощи метода исключения Гаусса получено треугольное разложение LU матрицы A и вычислено решение $\bar{x}$, причем все результаты сохранены. Тогда, что следует взять в качестве аппроксимирующей матрицы C в пункте (б) этой задачи? Как вычислить $\|CA-I\|$ эффективно? Как вычислить $\|Cr\|$ эффективно?

5. Указать в процентах для линейной системы сотого порядка оценку минимального количества дополнительных вычислений, требуемых для итерационного уточнения, чтобы «проверить», что полученное решение $Ax=b$ правильно. Предположить, что операции с удвоенной точностью требуют в три раза больше времени, чем соответствующие операции с одинарной точностью.

6. Решить следующую систему, используя четырехзначную десятичную арифметику [точное решение есть вектор (2, 3)]:

$$7x_1+6.990x_2=34.97,$$
$$4x_1+\quad 4x_2=20\ 00.$$

Запомнить разложение матрицы коэффициентов и выполнить итерационное уточнение до тех пор, пока не будет достигнута сходимость. Должно потребоваться четыре итерации.

7. Составить подпрограмму на языке Фортран, которая вычисляет три числа обусловленности самым грубым способом (альтернативы 1 или 3a в подразделе

8.Б.2). Применить эти числа для вычисления оценки ошибок, основанной на ошибках округления, имеющих место на одной из конкретных вычислительных машин. Использовать эту подпрограмму для сравнения трех чисел обусловленности для матриц в подразделе 8.Д.1. *Указание.* Построить тестовые задачи с известными решениями так: выбрать решение и умножить его на матрицу, чтобы получить соответствующую правую часть.

8. Использовать подпрограмму решения линейных систем для систем из задачи 7 и сравнить оцененные ошибки с фактическими ошибками.

9. Насколько большим можно взять порядок матрицы Гильберта H_n, прежде чем будут потеряны все значащие цифры при решении системы $H_n\mathbf{x}=\mathbf{b}$ конкретной подпрограммой на конкретной машине? Получить характеристический профиль этой подпрограммы, показывающий зависимость фактической точности от порядка n.

10. Вычислить стандартное число обусловленности для матрицы H_n и получить характеристический профиль (использовать результаты задачи 9), показывающий зависимость фактической точности от стандартного числа обусловленности.

11. Повторить решение задачи 10 для естественного числа обусловленности.

12. Повторить решение задачи 10 для числа обусловленности Эйрда—Линча.

13. Изучить эффективность анализа чувствительности при помощи характеристического профиля, показывающего зависимость отношения оцененной ошибки к истинной ошибке от возмущений в правой части. Использовать H_6 в качестве матрицы тестовой системы с правой частью $(1, 1, 1\ 1, 1, 1)^T$ и возмущения порядка 10^{-6}.

14. Повторить решение задачи 13 при условии, что возмущения имеют место как в правой части, так и в матрице.

15. Спроектировать и выполнить вычислительный эксперимент для проверки правильности того, что альтернатива 2 в подразделе 8.Б.2 дает ожидаемое значение оценки $\frac{1}{2}\|A^{-1}\|$.

Глава 9

ОБУСЛОВЛЕННОСТЬ И ОБРАТНЫЙ АНАЛИЗ ОШИБОК

Мы провели детальный анализ того, как установить правильность полученного решения линейной системы уравнений. К сожалению, можно получить правильные ответы для вычислительной задачи, которые все же дают совершенно неверное решение для исходной задачи. Это кажущееся противоречие приводит иногда к серьезным недоразумениям — и нелегко разобраться, в чем тут дело. Цель настоящей главы состоит в том, чтобы объяснить эту ситуацию и представить некоторую точку зрения, которая в этом смысле весьма полезна.

9.А. ОБУСЛОВЛЕННОСТЬ ЗАДАЧ И ВЫЧИСЛЕНИЙ

Трудность понимания природы вычислительных ошибок впервые была осознана для линейных систем, а это та область, где такие трудности возникают часто. Однако вычислительные ошибки существуют повсюду в численных расчетах, и мы кратко изложим теорию обусловленности в ее общем контексте.

Рассмотрим, что означает иметь задачу и ее ответ. «Данные» процесса решения задачи содержат *всю* информацию, которая определяет задачу. Бóльшая часть данных состоит из численных значений, хотя данные включают в себя также математическую модель со всеми ее формулами и предположениями. Поэтому в задаче $\mathbf{Ax}=\mathbf{b}$ данные представляют собой не только числа в матрице A и в векторе b, но также и выбор переменных и уравнений для включения в линейную систему. В простом дифференциальном уравнении

$$3.1\,\frac{d^2u}{dt^2}+\left(1.-\frac{t^2}{2}\right)u=\sin(4.6t),\quad u(0.)=0.2,$$
$$u(6\ 2)=4.7,$$

девять коэффициентов 3.1, 1., 1/2, 1. (коэффициент у синуса), 4.6, 0.0, 0.2, 6.2 и 4.7 являются численными данными. В равной степени значимые «данные» представляют собой предположение о том, что в уравнении отсутствует член du/dt, возможное допущение, что $1.-(t^2/2)$ суть упрощенное представление cos (t), предположение о том, что осциллирующий член правой части точно представлен

функцией синуса, и т. д. Все эти числа, предположения и формулы воздействуют на решение исходной задачи и в своей совокупности образуют то, что называют термином «данные».

Ответом задачи может быть единственное число, множество чисел (для линейной системы), функция (для дифференциального уравнения) или их сложная комбинация. На рис. 9.1 показано геометрическое представление процесса решения задачи. Мы имеем

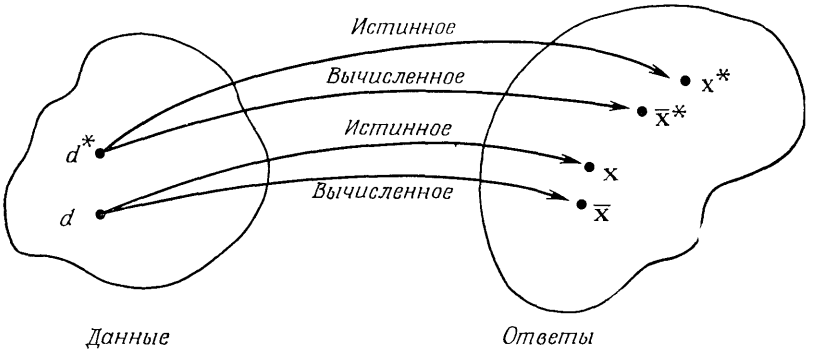


Рис. 9.1 Геометрическое представление влияния неопределенности исходных данных ($\mathbf{d}$ вместо $\mathbf{d}^*$) и вычислительных ошибок ($\bar{\mathbf{x}}$ вместо $\mathbf{x}$ и $\bar{\mathbf{x}}^*$ вместо $\mathbf{x}^*$) при решении задачи.

«истинные данные» $\mathbf{d}^*$ и «истинный ответ» $\mathbf{x}^*$. Мы имеем также «возмущенные данные» $\mathbf{d}$ и соответствующий «истинный ответ для возмущенных данных» $\mathbf{x}$. В численных расчетах мы почти никогда не получим истинного ответа, поэтому мы должны также рассмотреть «вычисленный ответ для точных данных» $\bar{\mathbf{x}}^*$ и «вычисленный ответ для возмущенных данных» $\bar{\mathbf{x}}$.

На самом деле процесс заканчивается вычислением $\bar{\mathbf{x}}$ вместо $\mathbf{x}^*$, и поэтому желательно оценить $\|\mathbf{x}^* - \bar{\mathbf{x}}\|$. К сожалению, часто это совершенно невозможно сделать конструктивным способом, и в этом состоит трудность в понимании существа ошибок. *Теория обусловленности* представляет собой попытку систематического изучения этого вопроса. Говорят, что задача, модель или вычисление *плохо обусловлены*, если они чувствительны к ошибкам или к неопределенности исходных данных. Обусловленность является качественным свойством, хотя мы будем стараться оценивать ее количественно.

Прежде всего мы оговорим различие между *плохо обусловленной задачей* и *плохо обусловленными вычислениями*. Все плохо обусловленные вычисления являются результатом применения численно неустойчивых алгоритмов. Четыре возможные комбинации «пло-

хой» и «хорошей» обусловленности показаны на рис. 9.2. Ключ к пониманию существа дела состоит в следующем:

> «*Если задача плохо обусловлена, то никакие усилия, потраченные на организацию изощренных вычислений, не могут дать правильных ответов, исключая случайности.*»

Поэтому (см. рис. 9.2 (в)) можно добиться того, чтобы $\bar{\mathbf{x}}$ было ближе к $\mathbf{x}$, а $\bar{\mathbf{x}}^*$ — ближе к $\mathbf{x}^*$, но вычисления не могут повлиять на расстояние между $\mathbf{x}$ и $\mathbf{x}^*$.

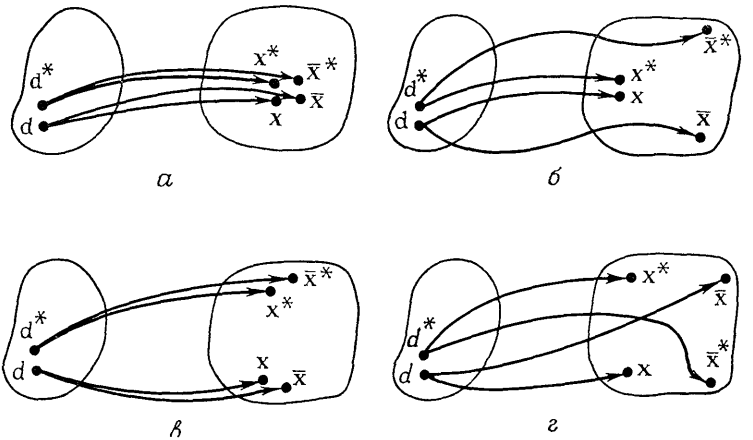


Рис. 9.2. Четыре комбинации плохой и хорошей обусловленности задачи и вычислений при решении этой задачи.
(а) Хорошо обусловленная задача.
Хорошо обусловленные вычисления.
(б) Хорошо обусловленная задача
Плохо обусловленные вычисления.
(в) Плохо обусловленная задача.
Хорошо обусловленные вычисления.
(г) Плохо обусловленная задача.
Плохо обусловленные вычисления.

Эта ситуация хорошо иллюстрируется примером гл. 8, содержащим матрицу Гильберта. Когда в статистической регрессии (как это часто бывает) используется плохо обусловленная модель (задача), решающий задачу обычно осознает, что вычисленные результаты бессмысленны. Первая реакция состоит в предположении, что в библиотечной программе имеется ошибка. Поэтому пользователь может составить свою собственную программу решения линейных уравнений. Это обычно приводит к совершенно отличным, но одинаково бессмысленным результатам. Тогда могут быть предприняты попытки использования арифметики двойной или тройной точности или интервальной арифметики. Это приводит к более точно опреде-

ленным неверным результатам, но все же неверным. Затем может быть решено, что вычислительный алгоритм является источником беспокойств.

Могут быть предприняты попытки решить систему методами, использующими элементарные отражения, или элементарные вращения, или итерации Гаусса — Зейделя, или итерации Якоби, или SOR-итерации[1]), или обобщенные обращения, или что-либо другое. Результатами будут новые множества бессмысленных ответов. Существо дела состоит в том, что *истинное решение* поставленной задачи не имеет отношения к реальности. Единственный способ получить осмысленные результаты — переформулировать математическую задачу для последующего решения так, чтобы ни матрица Гильберта, ни любая другая отдаленно похожая на нее матрица не входила когда-либо в модель или в процесс решения задачи.

В качестве отступления покажем, как делать такого рода переформулировки. В рассматриваемом примере причина трудностей заключена в том обстоятельстве, что полиномы записываются через степени x, т. е. 1, x, x^2, x^3, ... Эти функции почти линейно зависимы, поэтому их векторы значений почти линейно зависимы и матрица коэффициентов системы нормальных уравнений почти вырождена. Если использовать хорошее представление полиномов, скажем их запись через полиномы Чебышева $T_0(x)$, $T_1(x)$, $T_2(x)$, $T_3(x)$, . . . или полиномы Лежандра $P_0(x)$, $P_1(x)$, $P_2(x)$, $P_3(x)$, . . ., то эта задача может быть решена точно. Тем самым, решение задачи возможно только в предположениях, что полиномиальная модель экспериментальных данных в приемлемой степени достоверна и что в процессе вычислений приняты меры предосторожности: в любом случае никогда не использовать степени x.

Обусловленность связана с процессом (решения задачи) или преобразованием (исходных данных в ответ). Она может также весьма существенно зависеть от конкретных данных задачи, выделяющих последнюю из класса задач. Исходя из изложенного, дадим следующее определение:

> *Обусловленность задачи: обусловленность задачи «решить систему* $\mathbf{Ax}=\mathbf{b}$» *представляет собой обусловленность преобразования* $\mathbf{A}^{-1}$ *(процесса решения задачи) в окрестности точки* $\mathbf{b}$.

Определение дано для частного случая линейных уравнений, но та же самая идея применима и в общем случае.

Отметим, что из всех возможных недостатков программ для численных расчетов главным недостатком является выполнение плохо обусловленных вычислений для хорошо обусловленных задач (см. рис. 9.2(б)). Программа, которая использует метод исключения Гаусса без выбора ведущего элемента, может обладать таким не-

[1]) SOR — сокращение для метода последовательной верхней релаксации. — *Прим. перев.*

достатком. Для любой программы допустимо выполнение плохо обусловленных вычислений для плохо обусловленных задач. В конце концов решение этих задач — в любом случае почти наверняка безнадежное дело, и получить одно множество бессмысленных ответов, вероятно, все равно, что получить другое такое множество. Конечно, для программы весьма желательно обнаруживать, что ее вычисления плохо обусловлены, и сообщать об этом факте.

9.Б. ЕЩЕ РАЗ О ЧИСЛАХ ОБУСЛОВЛЕННОСТИ

В гл. 8 мы определили числа обусловленности как меру обусловленности задачи или вычислений. Рассмотрим здесь в деталях этот подход. Мы хотим найти *одно* число, которое измеряет обусловленность, даже несмотря на то что процесс решения задачи часто весьма сложен, и очень оптимистично надеяться, что только одно число может точно отразить чувствительность этого процесса ко всем видам ошибок и неопределенностей.

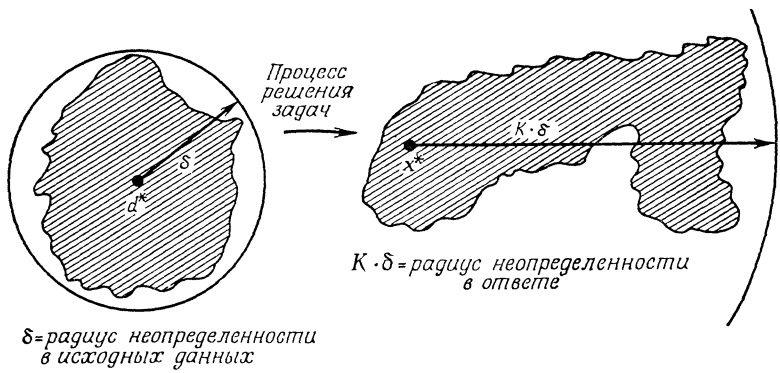


Рис. 9.3. Модель определения числа обусловленности K для процесса решения задачи

Понятие числа обусловленности основано на модели, показанной на рис. 9.3. Из него мы видим, что число обусловленности K представляет собой множитель, на который неопределенность δ умножается в процессе решения задачи. Вообще говоря, этот множитель K зависит от δ, что препятствует простоте изложения. Поэтому в последующих обсуждениях будем *предполагать*, что *как только значение* δ *становится малым, множитель K приближается к константе, и мы берем эту константу в качестве числа обусловленности.*

Ситуация усложняется далее тем обстоятельством, что иногда мы интересуемся абсолютными ошибками или неопределенностями, хотя обычно интерес представляют относительные ошибки. Приведенные выше определения применимы к обоим случаям, и мы будем

говорить соответственно об *абсолютной обусловленности* или *относительной обусловленности*.

Эти идеи могут быть изложены точным математическим образом (см. Rice, 1966a в списке литературы в конце книги), однако мы представим здесь более «интуитивное» их изложение.

Пример 9.1: Вычисление функции в окрестности точки. Пусть требуется вычислить заданную функцию [скажем, $f(x)=x^2+\cos(x^3+1)$] в окрестности точки x_0. Исходные данные состоят из одного

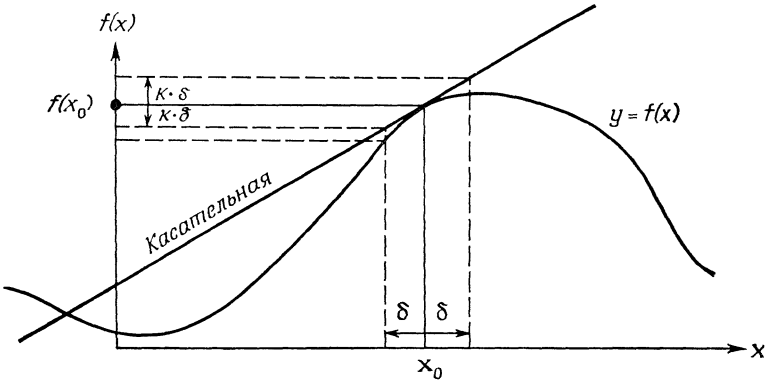


Рис. 9.4. Определение обусловленности вычисления $f(x)$ в окрестности точки x_0.

числа x_0, принадлежащего пространству данных из интервала $(-\infty, \infty)$. Ответ также состоит из одного числа $f(x_0)$ из интервала $(-\infty, \infty)$. Ситуация показана на рис. 9.4. Вблизи точки x_0 функцию $f(x)$ можно аппроксимировать ее касательной в точке x_0 и использовать касательную для оценки обусловленности вычисления функции. На рисунке значение δ взято довольно большим, поэтому существует значительное различие между истинным числом обусловленности и числом обусловленности, полученным с использованием касательной. Напомним, однако, что мы ограничиваем себя случаем, когда δ произвольно мало.

Диапазон значений, который может принимать $f(x)$ при изменении x в интервале $[x_0-\delta, x_0+\delta]$, имеет вид

$$[f(x_0-\delta),\ f(x_0+\delta)] \sim [f(x_0)-f'(x_0)\delta,\ f(x_0)+f'(x_0)\delta],$$

и поэтому абсолютная обусловленность вычисления функции $f(x)$ в окрестности точки x_0 представляет собой $|f'(x_0)|$.

Чтобы найти относительную обусловленность, необходимо рассмотреть, что происходит для $x \in [x_0(1-\delta),\ x_0(1+\delta)]$. Полученные значения функции лежат в интервале

$$[f(x_0(1-\delta)),\ f(x_0(1+\delta))] \sim [f(x_0)-f'(x_0)\,\delta x_0,\ f(x_0)+f'(x_0)\,\delta x_0].$$

Величина этого интервала относительно $f(x_0)$ равна

$$\left|\frac{f'(x_0)\ x_0}{f(x_0)}\right|\delta,$$

и поэтому относительное число обусловленности равно

$$|x_0|\ \frac{|f'(x_0)|}{|f(x_0)|}.$$

Пример 9.2: Решение $\mathbf{Ax}=\mathbf{b}$. Несмотря на то что мы уже нашли числа обусловленности для этой задачи, выведем их снова из наших общих рассуждений. Мы увидим, что это не совсем так просто, как можно было бы ожидать. Мы хотим найти результат возмущения матрицы A матрицей δA и вектора $\mathbf{b}$ вектором $\delta\mathbf{b}$. Имеем

$$A\mathbf{x}^* = \mathbf{b},$$

$$(A+\delta A)\,\bar{\mathbf{x}} = \mathbf{b}+\delta\mathbf{b}.$$

Мы видим, что

$$\begin{aligned}\mathbf{x}^*-\bar{\mathbf{x}} &= A^{-1}\mathbf{b}-(A+\delta A)^{-1}(\mathbf{b}+\delta\mathbf{b})\\ &= [A^{-1}-(A+\delta A)^{-1}]\,\mathbf{b}-(A+\delta A)^{-1}\,\delta\mathbf{b}.\end{aligned}$$

Предположим теперь, что δA и $\delta\mathbf{b}$ настолько малы, что любой член, содержащий $(\delta A)^2$ или $\delta A\delta\mathbf{b}$, может быть отброшен. Тогда обычные преобразования линейной алгебры приводят к выражению:

$$\mathbf{x}^*-\bar{\mathbf{x}}\sim A^{-1}\delta AA^{-1}\mathbf{b}-A^{-1}\delta\mathbf{b}.$$

Теперь мы имеем дело с *двумя* различными возмущениями, но требуем наличия *одного* фактора, определяющего число обусловленности. Существует возможность ввести векторные числа обусловленности, однако мы не будем этого делать и взамен *предположим* некоторое соотношение между величинами δA и $\delta\mathbf{b}$. Наиболее простым таким предположением является предположение о том, что *возмущения в обеих частях системы имеют одну и ту же величину* δ. Это означает, что

$$\|\delta A\mathbf{x}^*\|\sim\|\delta\mathbf{b}\|,$$

откуда имеем

$$\begin{aligned}\|\mathbf{x}^*-\bar{\mathbf{x}}\| &\sim \|A^{-1}\delta AA^{-1}\mathbf{b}\|+\|A^{-1}\delta\mathbf{b}\|\\ &= \|A^{-1}\delta A\mathbf{x}^*\|+\|A^{-1}\delta\mathbf{b}\|\\ &\leqslant \|A^{-1}\|\,\|\delta A\mathbf{x}^*\|+\|A^{-1}\delta\mathbf{b}\|\\ &\sim 2\|A^{-1}\|\,\|\delta\mathbf{b}\| = 2\|A^{-1}\|\,\delta.\end{aligned}$$

Поскольку возмущение в исходных данных имеет величину 2δ (одно δ для каждого из $\mathbf{b}$ и $A\mathbf{x}$), мы видим, что абсолютная обусловленность равна $\|A^{-1}\|$.

Относительная обусловленность определяется из

$$\frac{\|\mathbf{x}^*-\bar{\mathbf{x}}\|}{\|\mathbf{x}^*\|}\leqslant\frac{2\|A^{-1}\|\,\|\delta\mathbf{b}\|}{\|\mathbf{x}^*\|}\leqslant\frac{\|A^{-1}\|\,\|\mathbf{b}\|}{\|\mathbf{x}^*\|}*\frac{2\|\delta\mathbf{b}\|}{\|\mathbf{b}\|},$$

что дает $\|A^{-1}\|\,\|\mathbf{b}\|/\|x^*\|$, а это то же самое, что и *естественное число обусловленности*, найденное ранее. *Стандартное число обусловленности* находится подстановкой оценки $1/\|\mathbf{x}\| \leqslant \|A\|/\|\mathbf{b}\|$, которая выполняется для всех $\mathbf{x}$.

9.В. СОСТАВНАЯ ФОРМУЛА ОЦЕНКИ ОШИБОК И АЛГОРИТМЫ

Теперь мы проведем еще одно исследование задачи решения линейных систем. Будем различать три линейные системы:

Исходная задача: $A\mathbf{x}=\mathbf{b}$.
Машинная задача: $A\,(I+P)\mathbf{x}'=(I+D)\mathbf{b}$.
Решенная задача: $A\,(I+P)\bar{\mathbf{x}}=(I+D)\mathbf{b}+\mathbf{r}$.

Исходная задача — это задача из «реального мира», которую мы на самом деле и хотим решить в точной арифметике. *Машинная задача* — это задача, которая фактически находится внутри вычислительной машины; существуют матрицы P и D возмущений исходной задачи (D — диагональная матрица). Эти возмущения могут возникнуть при записи чисел a_{ij} и b_i в память машины или в силу неопределенностей, либо присущих этим числам, либо возникающих из-за ошибок эксперимента. Последние неопределенности весьма велики для многих реальных задач по сравнению с машинными ошибками округления. *Решенная задача* порождает дальнейшие возмущения из-за различного рода несоответствий в машинной арифметике или в алгоритме решения машинной задачи. Вектор невязки $\mathbf{r}$ представляет общий эффект этих несоответствий.

Оценим $\bar{\mathbf{x}}-\mathbf{x}$. Имеем

$$\mathbf{x}=A^{-1}\mathbf{b},$$
$$\mathbf{x}'=(I+P)^{-1}A^{-1}(I+D)\,\mathbf{b},$$
$$\bar{\mathbf{x}}=(I+P)^{-1}A^{-1}(I+D)\,\mathbf{b}+(I+P)^{-1}A^{-1}\mathbf{r}.$$

Как обычно, предположим, что P, D и $\mathbf{r}$ относительно малы. Тем самым можно использовать оценку

$$(I+P)^{-1}\sim I-P$$

и получить

$$\bar{\mathbf{x}}=A^{-1}\mathbf{b}-PA^{-1}\mathbf{b}+A^{-1}D\mathbf{b}+A^{-1}\mathbf{r}-PA^{-1}D\mathbf{b}-PA^{-1}\mathbf{r}$$
$$\sim A^{-1}\mathbf{b}-PA^{-1}\mathbf{b}+A^{-1}D\mathbf{b}+A^{-1}\mathbf{r}.$$

Вычитание $\mathbf{x}=A^{-1}\mathbf{b}$ из обеих частей дает

$$\bar{\mathbf{x}}-\mathbf{x}\sim -PA^{-1}\mathbf{b}+A^{-1}D\mathbf{b}+A^{-1}\mathbf{r}$$
$$=-P\mathbf{x}+A^{-1}D\mathbf{b}+A^{-1}\mathbf{r}$$

и
$$\frac{\|\bar{x}-x\|}{\|x\|} \leqslant \frac{\|Px\|}{\|x\|} + \frac{\|A^{-1}Db\|}{\|x\|} + \frac{\|A^{-1}r\|}{\|x\|}$$
$$\leqslant \|P\| + \frac{\|A^{-1}Db\|}{\|x\|} + \frac{\|A^{-1}r\|}{\|x\|}.$$

Эта *составная формула оценки ошибок* является расширением оценки обусловленности Эйрда — Линча $||\mathbf{Cr}||/||\mathbf{x}||$; добавлены два члена для учета эффекта $||P||$ от возмущения матрицы и $||A^{-1}Db||/||x||$ от возмущения правой части. Вспомним из гл. 8, что оценка Эйрда — Линча недооценивала действительную ошибку во всех рассмотренных примерах. Если исследовать эти и подобные примеры более детально, то можно увидеть, что причина такой недооценки состоит в игнорировании ошибок возмущения из-за перехода к 48-разрядным двоичным числам и причислении их к фактически измеренным ошибкам. Естественное и стандартное числа обусловленности также игнорируют эти возмущения, однако они настолько сильно переоценивают ошибки, что это игнорирование не проявилось в примерах. Можно построить примеры, когда они слишком недооценивают фактическую ошибку.

Возможно проведение вычислений по составной формуле оценки ошибок. Норма P относится к исходным данным: либо известна неопределенность в задании матрицы A, либо предполагается, что p_{ij} имеют порядок машинных ошибок округления. Можно оценить $||A^{-1}Db||$ и $||A^{-1}r||$ довольно эффективно, как это обсуждалось в гл. 8, без вычисления A^{-1}. Следующий пример демонстрирует рабочие характеристики составной формулы оценки ошибок для четырех примеров, данных в гл. 8.

Пример 9.3: Составная формула оценки ошибок, примененная к примерам 8.2—8.5. Вычисления проводились на CDC 6500, выполняющей арифметические операции с 14.5 десятичными знаками. Поэтому мы возьмем EPSA=EPSB=10^{-14} в оценке

$$\frac{\|x-\bar{x}\|}{\|x\|} \sim \frac{2\|Cr\|}{\|x\|} + N * EPSA + \frac{\|A^{-1}d\|\,EPSB}{\|x\|},$$

где **d** — вектор с компонентами $d_i = v * b_i$ и **v** — случайная переменная, равномерно распределенная между 1 и —1. Были сгенерированы два вектора **d** и было использовано наибольшее значение $||A^{-1}d||$. Для облегчения сравнения мы воспроизведем данные для трех чисел обусловленности: естественного, стандартного и Эйрда — Линча (см. табл. 9.0).

Эти сравнения показывают, что составная формула оценки ошибок является привлекательной; она всегда дает верхнюю границу и из всех четырех оценок наиболее дешева для вычислений. Конечно, более быстрые способы оценки других чисел обусловленности — быстрее, но ненамного. Сводки двух экспериментов над большим числом матриц приведены в табл. 9.1 и 9.2.

Таблица 9.0

Формула оценки ошибок	Оценка ошибок	Потерянные цифры	Относительные затраты
Пример 8.2: фактическая ошибка = 2.770E — 4, время решения = = 0.024 сек			
Естественная	6.434E — 3	1.366	17.5
Стандартная	1.812E + 5	8.816	17.5
Эйрда — Линча	2.693E — 4	—0.012	1.83
Составная	5.383E — 4	0.289	1.08
Пример 8.3: фактическая ошибка = 1.774E — 3, время решения = = 0.005 сек			
Естественная	1.681E + 2	4.977	3.9
Стандартная	1.681E + 2	4.977	4.0
Эйрда — Линча	1.781E — 7	—3.998	1.4
Составная	8.721E — 3	0.692	1.6
Пример 8 4: фактическая ошибка = 1.450E — 12, время решения = = 0.048 сек			
Естественная	3.179E — 11	1.34	12.1
Стандартная	3.179E — 11	1.34	12.2
Эйрда — Линча	1.261E — 12	—0.06	1.9
Составная	5.258E — 12	0.56	0.88
Пример 8.5: фактическая ошибка = 1.529E — 8, время решения = = 0.549 сек			
Естественная	4.009E — 6	2.42	24.0
Стандартная	1.034E + 3	10.83	24.1
Эйрда — Линча	9.147E — 9	—0.22	1.8
Составная	1.830E — 8	0.078	0.43

Приведенный ниже алгоритм 9.1 является одним из способов включения составной формулы оценки ошибок в процесс решения методом исключения Гаусса системы Ax=b. На вход алгоритму 9.1 подаются A, b, EPSA и EPSB; выходными данными являются x и ERROR.

Таблица 9.1

Сводка характеристик оценок ошибок для 52 линейных систем с обусловленностью, изменяющейся в пределах от умеренной до крайне плохой

Оценка ошибок	Количество потерянных цифр		
	Максимальное	Минимальное	Среднее
Естественная	16.3	0.82	7.2
Стандартная	28.5	3.1	14.6
Эйрда—Линча	0.0	—4.0	—0.31
BOUNDS	12.8	0.26	5.3
Составная	12.7	—0.20	1.2

Таблица 9.2

Сводка характеристик оценок ошибок для 20 случайных систем (хорошо обусловленных), имеющих порядок 10 и 25

Оценка ошибок	Количество потерянных цифр					
	Максимальное		Минимальное		Среднее	
	Порядок 10	Порядок 25	Порядок 10	Порядок 25	Порядок 10	Порядок 25
Естественная	1.66	2.03	0.47	1.11	1.05	1.47
Стандартная	3.45	4.38	1.59	2.62	2.57	3.43
Эйрда—Линча	0.13	0.02	—0.49	—0.29	—0.06	—0.04
BOUNDS	1.70	1.88	0.68	0.89	1.15	1.33
Составная	1.09	0.96	0.25	0.49	0.67	0.68

Алгоритм 9.1: Метод исключения Гаусса с составной формулой оценки ошибок

1. Разложить матрицу A на множители исключением Гаусса с частичным выбором ведущего элемента так, чтобы $MA = U$.
2. Вычислить $\mathbf{x}$ из систем $\mathbf{v} = M\mathbf{b}$, $U\mathbf{x} = \mathbf{v}$.
3. а. Вычислить невязку $\mathbf{r}$ в режиме удвоенной точности и привести ее усечением к одинарной точности.
   б. Вычислить $\mathbf{y}$ из систем $\mathbf{v} = M\mathbf{r}$, $U\mathbf{y} = \mathbf{v}$.
   в. $\mathbf{x} = \mathbf{x} + \mathbf{y}$.
4. а. Дважды выполнить следующее:
   Сгенерировать случайное число α и положить $c_i = \alpha b_i$, $i = 1, 2, \ldots, N$.
   Вычислить $\mathbf{z}$ из систем $\mathbf{v} = M\mathbf{c}$, $U\mathbf{z} = \mathbf{v}$.
   б. $ERR1 = 2\|\mathbf{y}\|/\|\mathbf{x}\|$,

ERR2 = N * EPSA,
ERR3 = 1.3 * max [‖z‖]/‖x‖ * EPSB,
5. ERROR = ERR1 + ERR2 + ERR3.

Отметим, что этот алгоритм выполняет первую итерацию итерационного уточнения. Поскольку для оценки ошибок необходимо знать ||y||, представляется разумным на шаге 3в прибавить **y** к **x**. Это сделает формулу оценки ошибок значительно более умеренной, поскольку ERROR на самом деле оценивает ошибку решения, полученного на шаге 2, а не ошибку более точного решения, фактически полученного алгоритмом.

Важно подчеркнуть, что составная формула оценки ошибок основывается на предположении, что возмущение матрицы A имеет вид A(I+P), где $|P_{ij}| \leqslant$ EPSA. Это возмущение не то же самое, что возмущение матрицы A в виде A+E, где $e_{ij} = \alpha a_{ij}$ и α — случайное число, $|\alpha| \leqslant$ EPSA. Мы видим, что $P = A^{-1}E$, и поэтому ||P|| может быть гораздо больше, чем ||E||, если A плохо обусловлена. Этот эффект может наблюдаться на практике, и составная формула оценки ошибок *не* является надежной оценкой ошибок при возмущениях EPSA в индивидуальных элементах матрицы A. Если такого рода возмущения имеют место, следует использовать модифицированную составную формулу оценки:

$$\frac{\|x - \bar{x}\|}{\|x\|} \leqslant \frac{\|A^{-1}Ex\|}{\|x\|} + \frac{\|A^{-1}Db\|}{\|x\|} + \frac{\|A^{-1}r\|}{\|x\|}.$$

Эта формула является более дорогостоящей для вычислений, так как теперь необходимо оценить $\|A^{-1}Ex\|$. Однако, поскольку E — случайная матрица, можно оценить $\|A^{-1}Ex\|$ посредством решения $Az = e$ для нескольких случайных векторов. Это не вполне надежно, но тем не менее имеется высокая вероятность того, что будет получена верхняя граница ошибки. Такой подход реализован а алгоритме 9.2.

Алгоритм 9.2: Метод исключения Гаусса с модифицированной составной формулой оценки ошибок

1. Разложить матрицу A на множители исключением Гаусса с частичным выбором ведущего элемента так, чтобы MA = U.
2. Вычислить **x** из систем **v** = M**b**, U**x** = **v**.
3. а. Вычислить невязку **r** в режиме удвоенной точности и привести ее усечением к одинарной точности.
   б. Вычислить **y** из систем **v** = M**r**, U**y** = **v**.
   в. **x** = **x** + **y**.
4. а. Дважды выполнить следующее:
   Сгенерировать случайные числа α, β и положить $c_i = \alpha b_i$, $d_i = \beta x_i$ для i = 1, 2, ..., N.
   Вычислить **z** и **w** из систем **v** = M**c**, U**z** = **v**, **u** = M**d**, U**w** = **u**.

6. $ERR1 = 2\|\mathbf{y}\|/\|\mathbf{x}\|,$
   $ERR2 = 1.3 * \max[\|\mathbf{w}\|]/\|\mathbf{x}\| * EPSA,$
   $ERR3 = 1\ 3 * \max[\|\mathbf{z}\|]/\|\mathbf{x}\| * EPSB.$
5. $ERROR = ERR1 + ERR2 + ERR3$

Важно подчеркнуть, что данные в этом разделе оценки ошибок имеют следующую интуитивную интерпретацию

Неопределенность решения $\sim$ неопределенность модели + неопределенность исходных данных + ошибки вычислений.

$$\frac{\|\mathbf{x}-\bar{\mathbf{x}}\|}{\|\mathbf{x}\|} \sim \frac{\|A^{-1}Ex\|}{\|\mathbf{x}\|} + \frac{\|A^{-1}Db\|}{\|\mathbf{x}\|} + \frac{\|A^{-1}\mathbf{r}\|}{\|\mathbf{x}\|}.$$

Здесь матрица интерпретируется как модель состояния, а правая часть — как исходные данные или вынуждающая функция модели. Разумное использование этих оценок требует предварительных суждений об относительных размерах этих неопределенностей. Численный анализ концентрируется на ошибках вычислений, поскольку они могут регулироваться в процессе вычислений. Во многих реальных ситуациях ошибки вычислений пренебрежимо малы по сравнению с неопределенностями в модели или в исходных данных. Невероятные ответы, полученные в результате вычислений, обычно являются только отражением высокой чувствительности модели (плохой обусловленности) и не указывают на то, что вычисления выполнены плохо.

Существует распространенное мнение, что стандартное число обусловленности обеспечивает хорошую оценку неопределенности для тех ситуаций, в которых возмущены как A, так и **b**. Достоверность этого мнения существенно зависит от предположений об относительных размерах неопределенностей и сущности возмущений. Пусть матрица E «пропорциональна» матрице A в том смысле, что $|e_{ij}| \sim |a_{ij}| * EPSA$. Тогда невольно напрашивается оценка $||A^{-1}Ex||/||x||$ через $||A^{-1}||\ ||A||\ EPSA$, которая дает верхнюю границу

$$\frac{\|\mathbf{x}-\bar{\mathbf{x}}\|}{\|\mathbf{x}\|} \leqslant \|A^{-1}\|\|A\|EPSA + \|A^{-1}\|EPSB + \frac{\|A^{-1}\mathbf{r}\|}{\|\mathbf{x}\|}.$$

Первый член часто доминирует, и в этих случаях можно было бы заключить, что $||A^{-1}||\ ||A||\ EPSA$ представляет собой надежную оценку неопределенности при решении $A\mathbf{x}=\mathbf{b}$. Однако эксперименты показывают, что $||A^{-1}||\ ||A||\ EPSA$ — *не* полностью надежная оценка эффекта такого рода возмущений в матрице A. Причина этого следующая. Оценка

$$||A^{-1}Ex|| \leqslant ||A^{-1}||\ ||Ex||,$$

вообще говоря, не является тонкой оценкой. Заметим, что решение **x** фиксировано для конкретной задачи $A\mathbf{x}=\mathbf{b}$, и равенство

$$||A^{-1}Ex|| = ||A^{-1}||\ ||Ex||$$

требует, чтобы вектор Ex был специальным вектором (собственным вектором матрицы A^{-1}, соответствующим наибольшему собственному значению) Это, конечно, может быть достигнуто для некоторой матрицы E, но, вообще говоря, это не может быть достигнуто для случая, когда ||Ex||=||x|| EPSA Равным образом, если вектор Ex обладает требуемым свойством, то ||Ex||=||x|| EPSA не является тонкой оценкой

Последние задачи в конце этой главы иллюстрируют то обстоятельство, что различные предположения относительно сущности матрицы E отражаются на точности этой оценки. Рассматривается простая верхняя треугольная матрица третьего порядка при предположениях

$$|e_{ij}| \leqslant |a_{ij}| * \text{EPSA}; \tag{9.1}$$

$$\begin{aligned} |e_{ij}| &\leqslant \max_{i,\,j} |a_{ij}| * \text{EPSA}, && \text{если } |a_{ij}| > 0; \\ &= 0, && \text{если } |a_{ij}| = 0; \end{aligned} \tag{9.2}$$

$$\begin{aligned} |e_{ij}| &\leqslant \|A\| * \text{EPSA}, && \text{если } |a_{ij}| > 0; \\ &= 0, && \text{если } |a_{ij}| = 0; \end{aligned} \tag{9.3}$$

$$\|E\| \leqslant \|A\| * \text{EPSA}. \tag{9.4}$$

Нехватка точности в рассмотренной выше оценке характеризуется множителями 18.87, 2.75, 1.83 и 1 соответственно для этих четырех предположений.

9.Г. ОБРАТНЫЙ АНАЛИЗ ОШИБОК

Первой и наиболее естественной попыткой изучения воздействия ошибок на точность численных расчетов является *прямой анализ ошибок*. При таком анализе начинают с исходной задачи и исследуют шаг за шагом воздействие вычислительных ошибок (ошибок округления) и неопределенностей исходных данных. Если обладать достаточным искусством, можно получить оценку ошибки в окончательном результате. Этот процесс иллюстрируется примером 9.4.

Пример 9.4: Прямой анализ ошибок при вычислении выражения $ab^2+\sin$ (*3.1a—b*). Рассмотрим фрагмент программы

```
      READ 10  A, B
10    FORMAT (2F8 6)
      ANS = A * B ** 2 + SIN(3 1 * A   B)
      PRINT 20, ANS
20    FORMAT (F15 9)
```

Предположим, что вычисления проводятся на машине с семью десятичными цифрами, так что ошибка округления ε равна 10^{-7}. Предположим также, что значения A и B приблизительно равны 1. Предполагается, что исходные данные специфицируются оператором FORMAT с меткой 10, и поэтому неопределенность δ в A и B мы по-

ложим равной 10^{-6}. Операторы ввода/вывода языка Фортран также вносят ошибки округления, но мы их будем здесь игнорировать.

В действительности вычисления проводятся следующим образом:

$$B**2 = B^2 + 2\delta B + \delta^2 + \varepsilon \tag{9.5}$$

$$A * B**2 = AB^2 + 2\delta AB + \delta^2 A + \varepsilon A + \delta B^2 + 2\delta^2 B + \delta^2 + \delta\varepsilon + \varepsilon \tag{9.6}$$

$$3.1 * A = 3.1A + 3.1\delta + \varepsilon \tag{9.7}$$

$$3.1 * A - B = 3.1A + 3.1\delta + \varepsilon - B - \delta + \varepsilon \tag{9.8}$$

$$SIN\,(3.1 * A - B) = SIN\,(3.1A + 3.1\delta + \varepsilon - B - \delta + \varepsilon) + \varepsilon \tag{9.9}$$

$$ANS = AB^2 + 2\delta AB + \delta^2 A + \varepsilon A + \delta B^2 + 2\delta^2 B + \delta^2 + \delta\varepsilon + \varepsilon + SIN\,(3.1A + 3.1\delta + \varepsilon - B - \delta + \varepsilon) + \varepsilon + \varepsilon \tag{9.10}$$

Теперь в этих выражениях имеется большая путаница, и очевидно, что этот подход не может быть выполнен до конца в деталях для любых сложных вычислений. Поэтому мы должны выполнить аппроксимации и упрощения. Таковыми прежде всего являются: (1) пренебрежение любыми членами, включающими в себя произведения двух ошибок; (2) аппроксимация функций их касательной (один член ряда Тейлора); (3) комбинирование членов, содержащих ε и δ, в предположении, что ε и δ — неизменны. Повторив вычисления, получим

$$B ** 2 = B^2 + 2\delta B + \varepsilon \tag{9.5a}$$

$$A * B ** 2 = AB^2 + 2\delta AB + \varepsilon A + \delta B^2 + \varepsilon. \tag{9.6a}$$

$$3.1 * A = 3.1A + 3.1\delta + \varepsilon \tag{9.7a}$$

$$3.1 * A - B = 3.1A + 3.1\delta + \varepsilon - B - \delta + \varepsilon \tag{9.8a}$$

$$SIN\,(3.1 * A - B) = SIN\,(3.1\,A - B) + \varepsilon + COS(3\,1A - B)*(3.1\delta + \varepsilon - B - \delta + \varepsilon) \tag{9.9a}$$

$$ANS = AB^2 + SIN\,(3.1A - B) + \varepsilon(A + 3 + 2\,COS\,(3.1A - B)) + \delta(2AB + B^2 + 4.1) \tag{9.10a}$$

Это выражение немного, но не слишком, проще. При этом эффект ошибок существенно преувеличен, поскольку мы предположили, что ошибки комбинируются так, чтобы дать наихудший из всех возможных результатов. Это означает, например, что сумма $\varepsilon+\varepsilon+\varepsilon$ (сумма трех ошибок округления) вряд ли равна $3*10^{-7}$, так как (1) фактические ошибки меньше, чем 10^{-7}, и (2) некоторые ошибки

положительны, а некоторые — отрицательны, так что следует ожидать их взаимного уничтожения.

В одной из наиболее знаменитых статей, посвященных численным расчетам, рассматривался прямой анализ ошибок для исключения Гаусса (von Neumann, Goldstine, 1947). Эта статья приводила к заключению, что эффект накопления ошибок округления делает невозможным решение больших (скажем, 100×100) систем уравнений. Вычислительные машины работали в то время с числами, содержавшими от 10 до 12 десятичных цифр. В ретроспективе мы видим: эта статья доказывает также, что прямой анализ ошибок вряд ли дает много полезной информации.

Многое в теории обусловленности и использовании чисел обусловленности выдержано в духе прямого анализа ошибок, и хотя их менее утомительно вычислять, мы видели, что эти числа часто значительно переоценивали эффект вычислительных ошибок и неопределенностей в исходных данных.

Основная цель анализа ошибок состоит в оценке $\|\bar{\mathbf{x}}-\mathbf{x}^*\|$, где $\mathbf{x}^*$ — истинное решение, а $\bar{\mathbf{x}}$ — вычисленное решение. К сожалению, практические оценки $\|\bar{\mathbf{x}}-\mathbf{x}^*\|$ трудно либо невозможно получить. Приведем список вопросов, которые можно рассматривать при анализе эффекта ошибок и неопределенностей данных на ответы решаемых задач:

1. Какова наихудшая ошибка, которая может иметь место?
2. Какие ошибки на самом деле имели место?
3. Какие ожидаются ошибки?
4. Насколько чувствительна задача к ошибкам?
5. Насколько близка фактически решенная задача к задаче, которую мы хотели решить?

При помощи прямого анализа ошибок и априорных границ ошибок можно попытаться ответить на первый вопрос. Сплошь и рядом эти ответы безнадежно пессимистичны и полезны только для того, чтобы показать, что задача ведет себя очень хорошо. Казалось бы, большая часть усилий в гл. 8 направлена на второй вопрос, однако в действительности рассматривался четвертый вопрос. Истинное решение $\mathbf{x}^*$ фактически почти никогда не известно, поэтому на второй вопрос почти никогда нельзя дать ответ.

Программа должна использовать оценки уровня неопределенностей исходных данных и чувствительности задачи так, чтобы делать суждение относительно достоверности ответа; это означает, что необходимо отвечать на третий вопрос настолько хорошо, насколько это возможно. Вспомним, что хорошая программа вполне может вычислить «правильный» ответ для неверной задачи и сообщить, что вычисления прошли успешно. Немного известно относительно ожидаемых ошибок сверх того факта, что они часто *гораздо* меньше ошиб-

бок, вычисляемых процедурами оценки ошибок. Мы приходим к выводу, что первые четыре вопроса либо не имеют ответов, либо ответы на них дают мало информации о действительной величине ошибки $||\bar{x}-x^*||$.

На пятый вопрос может быть дан удовлетворительный ответ, и его рассмотрение приводит к действительному проникновению в суть вопроса о достоверности многих численных расчетов. На пятый вопрос дает ответ *обратный анализ ошибок*, и он может быть выполнен для линейных систем (и для многих других задач) с относительной

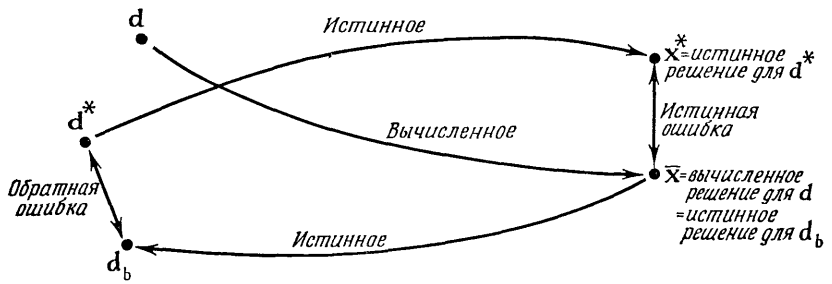


Рис. 9.5. Графическая иллюстрация обратного анализа ошибок. Истинные исходные данные и истинное решение обозначены как $\mathbf{d}^*$ и $\mathbf{x}^*$. Фактически использованные исходные данные обозначены через $\mathbf{d}$, а вычисленное на основе этих данных решение обозначено через $\bar{\mathbf{x}}$. Данные $\mathbf{d}_b$ представляют собой набор исходных данных, соответствующих истинному решению $\bar{\mathbf{x}}$.

легкостью (Wilkinson, 1963). Однако *этот анализ не дает какой-либо информации относительно ошибки* $||\bar{x}-x^*||$, *и требуется некоторое время, чтобы настроиться на эту точку зрения*. Идея обратного анализа ошибок графически представлена на рис. 9.5. Существует много наборов исходных данных, для которых истинным решением является $\bar{x}$; требуется найти набор данных d_b, которому соответствует малая обратная ошибка.

Пример 9.5: *Решение уравнения*

$$f(x)=2.23\,x^2\cos(x/4)+2.41x^4-5.06e^{-6.4x}+0.832=0.$$

Пусть предпринята попытка решить это уравнение и $x=0.256$ выбрано в качестве приближенного ответа. Если подставить 0.256 в уравнение $f(x)=0$, то получим невязку

$$f(0.256)=0.005086.$$

Это означает, что 0.256 является точным решением уравнений:

$$2.23x^2\cos(x/4)+2.41x^4-5.06e^{-6.4x}+0.82691=0,$$
$$2.23x^2\cos(x/4)+1.22582x^4-5.06e^{-6.4x}+0.832=0,$$
$$2.23x^2\cos(x/4)+2.41x^4-5.0338e^{-6.4x}+0.832=0.$$

Все эти уравнения имеют коэффициенты, измененные более чем на три знака (подразумеваемая точность) по сравнению с коэффициентами исходного уравнения. Поэтому вряд ли мы примем 0.256 в качестве решения, если только не предположить, что неопределенность в коэффициентах исходной задачи выше, чем это указано заданием значащих цифр в коэффициентах.

Предположим теперь, что в качестве другого приближения к ответу получено значение $\bar{x}=0.2553$. Тогда невязка будет равна

$$f(0.2553)=-0.0002356,$$

и поэтому 0.2553 является точным решением уравнения

$$2.23x^2\cos(x/4)+2.41x^4-5.06e^{-6.4x}+0.8322356=0.$$

Поскольку вероятно, что четвертая цифра члена 0.832 неизвестна, мы можем также предположить, что эта цифра есть 2, и тогда мы имеем точное решение уравнения, которое неотличимо от исходного уравнения.

Если мы продолжим поиск решения и найдем $\bar{x}=0.2553295$, которое дает невязку — 0.0000106, то мы, конечно, должны принять это $\bar{x}$ в качестве точного решения. В этом примере, как читатель, вероятно, уже догадывается, процесс решения протекает хорошо и все полученные ответы достаточно близки к «истинному» ответу 0.2553309476. Однако это не всегда имеет место, как видно из примера 9.6.

Пример 9.6: *Решение уравнения*

$$f(x)=x^5-7.5x^4+22.5x^3-33.75x^2+25.3125x-7.59375=0.$$

Пусть получено 1.33 в качестве приближенного ответа. Невязка равна

$$f(1.33)=-0.000142.$$

Все вычисления в этом примере выполняются с семью десятичными цифрами. Это означает, что 1.33 является точным решением уравнений

$$x^5-7.4995557x^4+22.5x^3-33.75x^2+25.3125x-7.59375=0,$$
$$x^5-7.5x^4+22.500334x^3-33.75x^2+25.3125x-7.59375=0,$$
$$x^5-7.5x^4+22.5x^3-33.75x^2+25.3125x-7.593608=0.$$

Первые два уравнения кажутся незначительными изменениями исходного уравнения, и поэтому мы могли бы принять 1.33 в качестве ответа. Третье же уравнение имеет значительное изменение в свободном члене.

Улучшенное приближение к решению равно 1.4026, оно дает невязку

$$f(1.4026)=-0.000032.$$

Таким образом, 1.4026 представляет собой точное решение уравнения

$$x^5-7.5x^4+22.500088x^3-33.75x^2+25.3125x-7.59375=0.$$

Это уравнение является очень небольшим возмущением исходного уравнения, и его решение должно быть принято в качестве правильного решения исходного уравнения.

Однако мы могли бы предположить, что «точный» ответ должен иметь меньшую невязку. Действительно, 1.582 имеет меньшую невязку:

$$f(1.582)=-0.000008.$$

Эта невязка является нулем с точностью до восьми знаков относительно коэффициента 33.75, и поэтому невязка настолько мала, насколько мы могли бы рассчитывать при использовании арифметики с семью десятичными знаками. Тем самым число 1.582 должно быть «точным» решением.

Однако нам следует быть настороже, поскольку три приближенных решения заметно отличаются друг от друга, но каждое из них дает небольшую невязку. Дальнейшие вычисления показывают, что

$$\begin{aligned} f(1.4) &= -0.00001, \\ f(1.462) &= -0.00001, \\ f(1.5002) &= 0.000007, \\ f(1.55) &= 0.000006, \\ f(1.6) &= 0.00001. \end{aligned}$$

На самом деле любое значение решения в пределах от 1.4 до 1.6 дает малую невязку, включая десятки значений, для которых невязка точно равна нулю (с учетом того, что вычисления проводятся с семью десятичными цифрами).

Дальнейшие исследования показывают, что функция $f(x)$ точно равна $(x-1.5)^5$, так что истинное решение должно быть равно 1.5. Однако, что же можно сказать относительно других значений x, которые также точно обращают $f(x)$ в нуль? Они, безусловно, должны быть истинными решениями этого уравнения. Этот пример представляет собой типичную плохо обусловленную задачу; существует много значительно отличающихся друг от друга «ответов», которые дают малые невязки, и каждый из этих ответов является точным решением задачи, *очень* близкой к исходной задаче. Все эти задачи оказываются хорошими в пределах неопределенности исходной задачи, и бессмысленно рассматривать любой результат в качестве точного ответа.

Подобная ситуация показана на рис. 9.6, содержащем график вычисленных значений полинома шестой степени:

$$P(x)=x^6-6x^5+15x^4-20x^3+15x^2-6x+1.$$

Вычисления проводились с 14 десятичными цифрами. Мы видим, что существует *интервал решений*, обращающих полином в нуль, и неопределенность в любом решении составляет 0.01.

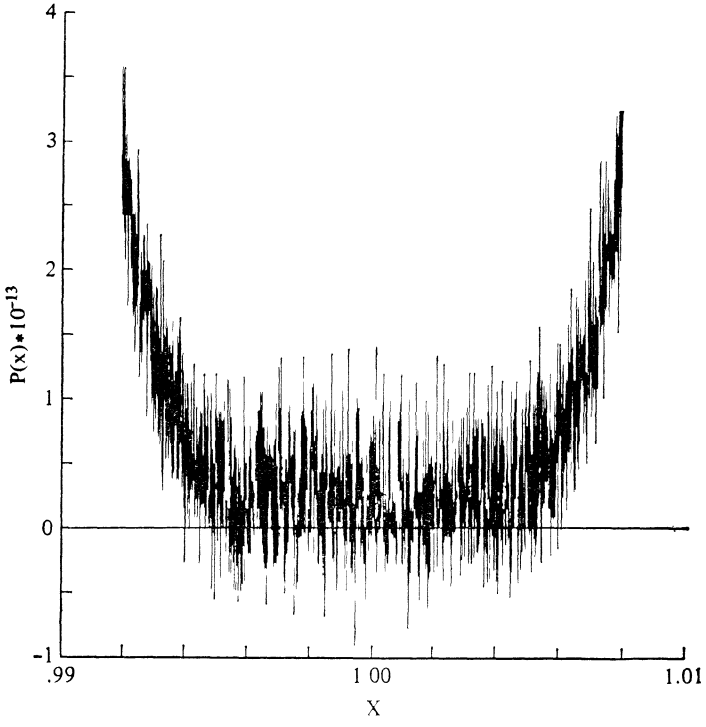


Рис. 9.6. График значений полинома шестой степени, вычисленных с использованием операций с 14 десятичными цифрами.

Цель обратного анализа ошибок состоит в том, чтобы оставить в стороне вопрос, получен ли «точный» ответ, поскольку это понятие для большинства реальных задач не является хорошо определенным. В действительности требуется найти ответ, который представляет собой истинное математическое решение задачи, лежащей в пределах области неопределенности исходной задачи. Любой результат, который удовлетворяет этому требованию, должен быть приемлем в качестве ответа к поставленной задаче, по крайней мере с позиции «философии» обратного анализа ошибок.

Применение обратного анализа ошибок к методу исключения Гаусса — не слишком трудная задача по крайней мере в распространенных случаях.

Пример 9.7: Обратный анализ ошибок для линейной системы третьего порядка. Рассмотрим метод исключения Гаусса для систе-

мы

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1,$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2,$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3,$$

в которой все a_{ij} приблизительно равны 1 и не требуется проводить выбор ведущего элемента. Обозначим через ε величину типичной ошибки округления. Первые два множителя в этом методе вычисляются следующим образом:

$$m_{21} = a_{21}/a_{11} + \varepsilon = (a_{21} + \varepsilon_1)/a_{11}, \qquad m_{31} = a_{31}/a_{11} + \varepsilon = (a_{31} + \varepsilon_2)/a_{11}.$$

Поскольку $a_{11} \sim 1$, то ε_1 и ε_2 близки по величине к ε, и поэтому m_{21} и m_{31} являются *точными множителями* для системы, в которой a_{21} и a_{31} возмущены на порядок ε.

Рассмотрим далее вычисление подматрицы второго порядка, чтобы завершить шаг исключения:

$$a_{22} = a_{22} - m_{21}a_{12} + \varepsilon_3 = (a_{22} + \varepsilon_3) - m_{21}a_{12},$$
$$a_{23} = (a_{23} + \varepsilon_4) - m_{21}a_{13},$$
$$a_{32} = (a_{32} + \varepsilon_5) - m_{31}a_{12},$$
$$a_{33} = (a_{33} + \varepsilon_6) - m_{31}a_{13}.$$

Каждая из ошибок округлений ε_3, ε_4, ε_5 и ε_6 имеет порядок ε, поскольку все числа имеют порядок 1, и поэтому новые вычисленные значения образуют *точную подматрицу второго порядка* для системы, в которой a_{ij}, $i=2, 3$ и $j=1, 2, 3$, возмущены на порядок ε.

На матричном языке это означает, что результат первого шага исключения Гаусса представляет собой точный результат для матрицы $A+E_1$, где E_1 имеет вид

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

Рассуждения повторяются для второго шага исключения Гаусса для подматрицы второго порядка. Результат второго шага представляет собой точный результат для матрицы $(A+E_1)+E_2$, где E_2 имеет вид

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

Первый столбец и первая строка матрицы E_2 являются нулевыми, поскольку соответствующие коэффициенты системы не участвуют в вычислениях. Остальная часть E_2 представляет собой просто E_1, сдвинутую вниз и вправо. Мы приходим к выводу, что общая мат-

рица возмущения $E=E_1+E_2$ имеет вид

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 2 \end{pmatrix}\varepsilon.$$

Применение множителей к правой части дает результат, подобный тому, который получен для последнего столбца матрицы системы. Это означает, что этот результат является точным результатом для вектора $\mathbf{b}+\mathbf{e}_1$, где $\mathbf{e}_1$ имеет вид $(0, 1, 2)^T\varepsilon$. Вычисления при обратной подстановке имеют вид

$$\begin{aligned} x_3 &= b_3/a_{33}+\varepsilon = (b_3+\varepsilon)/a_{33}, \\ x_2 &= (b_2-a_{23}x_3+\varepsilon)/a_{22}+\varepsilon = ((b_2+2\varepsilon)-a_{23}x_3)/a_{22}, \\ x_1 &= (b_2-a_{13}x_3-a_{12}x_2+2\varepsilon)/a_{22}+\varepsilon = \\ &= ((b_1+3\varepsilon)-a_{13}x_3-a_{12}x_2)/a_{11}. \end{aligned}$$

Таким образом, вычисленные значения x_1 представляют собой точные значения для правой части $\mathbf{b}+\mathbf{e}_1$, возмущенной вектором $\mathbf{e}_2=(3, 2, 1)^T\varepsilon$.

Мы приходим к заключению, что в этих предположениях вычисленный методом исключения Гаусса результат $\bar{\mathbf{x}}$ является точным решением системы

$$(A+E)\bar{\mathbf{x}}=\mathbf{b}+\mathbf{e},$$

где E имеет приведенный выше вид, а вектор $\mathbf{e}$ имеет порядок $(3, 3, 3)^T\varepsilon$.

Распространенной является ситуация, в которой все элементы матрицы A приблизительно равны по порядку 1 и *все* полученные промежуточные элементы не больше 1. Если применяется метод исключения Гаусса с выбором ведущего элемента для решения системы $A\mathbf{x}=\mathbf{b}$, то вычисленный результат $\bar{\mathbf{x}}$ представляет собой точное решение системы

$$(A+E)\,\bar{\mathbf{x}}=\mathbf{b}+\mathbf{e}.$$

Пусть ε — величина ошибки округления одной арифметической операции (сложения, вычитания или умножения). Тогда *в этих предположениях* E и $\mathbf{e}$ (по абсолютной величине) не больше, чем

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2 & 2 & 2 & \dots & 2 \\ 1 & 2 & 3 & 3 & 3 & & 3 \\ 1 & 2 & 3 & 4 & 4 & & 4 \\ 1 & 2 & 3 & 4 & 5 & & 5 \\ \vdots & & & & & & \vdots \\ 1 & 2 & 3 & 4 & 5 & \dots & N-1 \end{pmatrix}\varepsilon \quad \textit{и} \quad \begin{pmatrix} N \\ N \\ \vdots \\ \vdots \\ \vdots \\ N \\ N \end{pmatrix}\varepsilon$$

В большинстве случаев матрица A+E является хорошей в пределах неопределенности исходной задачи, и вычисленное решение должно быть признано правильным.

Заметим, что этот анализ неприменим ко всем задачам в силу принятых предположений. Вспомним из подраздела 5.В.2 трудности масштабирования матриц, которые показывают, что не всегда можно удовлетворить предположению относительно размеров элементов матрицы A. Более того, нельзя гарантировать, что размеры элементов матрицы A не будут каким-либо образом расти в процессе исключения Гаусса, хотя это и происходит очень редко (см. рассуждение в конце разд. 5.Б).

Идея обратного анализа ошибок может быть выражена в терминах *неопределенных линейных систем*. Определим абсолютное значение матрицы или вектора как матрицу или вектор абсолютных значений элементов, т. е

$$|A| = \begin{pmatrix} |a_{11}| & |a_{12}| & \dots & |a_{1n}| \\ |a_{21}| & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ |a_{m1}| & \dots & \dots & |a_{mn}| \end{pmatrix}, \quad |x| = (|x_1|, |x_2|, \dots, |x_n|)^T.$$

Говорят, что $|x| \leqslant |y|$, если $|x_i| \leqslant |y_i|$ для всех i, и $|A| \leqslant |B|$, если $|a_{ij}| \leqslant |b_{ij}|$ для всех i и j. Пусть даны матрица A^* и вектор $\mathbf{b}^*$ вместе с матрицей возмущений δA и вектором возмущений $\delta \mathbf{b}$. Тогда *неопределенная система* есть множество уравнений $Ax=\mathbf{b}$, где

$$|A^* - A| \leqslant \delta A,$$
$$|\mathbf{b}^* - \mathbf{b}| \leqslant \delta \mathbf{b}.$$

Вектор $\bar{\mathbf{x}}$ называется *приемлемым решением неопределенной системы*, если он является решением одного из множеств линейных уравнений, образующих неопределенную систему. Следующий результат относительно неопределенных систем установлен в работе (Oettle, Prager, 1964).

Теорема 9.1. *Вектор* $\bar{\mathbf{x}}$ *является приемлемым решением тогда и только тогда, когда*

$$|\mathbf{b}^* - A^*\bar{\mathbf{x}}| \leqslant \delta \mathbf{b} + \delta A |\bar{\mathbf{x}}|.$$

Доказательство. Пусть $\bar{\mathbf{x}}$ — приемлемое решение. Это означает, что $A\bar{\mathbf{x}}=\mathbf{b}$, где $|A-A^*| \leqslant \delta A$ и $|\mathbf{b}-\mathbf{b}^*| \leqslant \delta \mathbf{b}$ Тогда имеем

$$\begin{aligned} |\mathbf{b}^* - A^*\bar{\mathbf{x}}| &= |\mathbf{b} + (\mathbf{b}^* - \mathbf{b}) - A\bar{\mathbf{x}} - (A^* - A)\bar{\mathbf{x}}| \\ &\leqslant |\mathbf{b} - A\bar{\mathbf{x}}| + |\mathbf{b}^* - \mathbf{b}| + |(A^* - A)\bar{\mathbf{x}}| \\ &\leqslant \delta \mathbf{b} + \delta A |\bar{\mathbf{x}}|, \end{aligned}$$

что устанавливает первую часть теоремы

Пусть теперь $\bar{\mathbf{x}}$ удовлетворяет неравенству теоремы. Обозначим $\mathbf{r}=\mathbf{b}^*-A^*\bar{\mathbf{x}}$, $s=\delta\mathbf{b}+\delta A|\bar{\mathbf{x}}|$ и

$$t_i=\frac{r_i}{s_i}, \text{ если } s_i\neq 0,$$
$$t_i=0, \text{ если } s_i=0.$$

Вектор $\mathbf{t}$, очевидно, удовлетворяет неравенству $|\mathbf{t}|\leqslant\mathbf{1}$. Заметим, что если $s_i=0$, то из неравенства $|\mathbf{r}|\leqslant|\mathbf{s}|$ вытекает $\mathbf{r}_i=0$. Поэтому имеем

$$r_i=s_it_i=\left(\delta b_i+\sum_{j=1}^{n}\delta a_{ij}|\bar{x}_j|\right)t_i$$
$$=\delta b_it_i+\sum_{j=1}^{n}\delta a_{ij}\operatorname{sgn}(\bar{x}_j)\,t_i\bar{x}_j.$$

Обозначим

$$\varepsilon_i=-\delta b_it_i,$$
$$e_{ij}=\delta a_{ij}\operatorname{sgn}(\bar{x}_j)\,t_i.$$

Из определения $\mathbf{r}$ предыдущее соотношение примет вид

$$r_i=b_i^*-\sum_{j=1}^{n}a_{ij}^*\bar{x}_j=-\varepsilon_i+\sum_{j=1}^{n}e_{ij}\bar{x}_j,$$

что эквивалентно

$$\sum_{j=1}^{n}(a_{ij}^*+e_{ij})\,\bar{x}_j=b_i^*+\varepsilon_i.$$

Таким образом, имеем $A\bar{\mathbf{x}}=\mathbf{b}$, где $|A^*-A|=|E|\leqslant\delta A$ и $|\mathbf{b}-\mathbf{b}^*|\leqslant\delta\mathbf{b}$. Это — условие приемлемости решения $\bar{\mathbf{x}}$, что и заключает доказательство.

Пример 9.8: Анализ неопределенной системы с треугольной матрицей Уилкинсона (см. пример 8.3). Пусть $A^*\mathbf{x}=\mathbf{b}^*$ — машинная задача; это означает, что числа a_{ij} и b_i преобразованы в 48-разрядные двоичные числа. Пусть ε — машинная ошибка округления, которая в условиях этих вычислений равна $0.5*10^{-14}$. Тогда можно ожидать, что отклонения исходной задачи от машинной задачи составляют

$$\delta A=\begin{pmatrix}\varepsilon & 0 & 0 & 0\\ \varepsilon & \varepsilon & 0 & 0\\ \varepsilon & \varepsilon & \varepsilon & 0\\ \varepsilon & \varepsilon & \varepsilon & \varepsilon\end{pmatrix}\qquad \delta\mathbf{b}=(\varepsilon,\varepsilon,\varepsilon,\varepsilon)^T$$

Теперь зададим вопрос: является ли $\bar{\mathbf{x}}=(1, 1, 1, 1)^T$ приемлемым решением для машинной задачи? Норма невязки равна $1.5*10^{-14}$,

поэтому имеем

$$|\mathbf{b}^*-\mathrm{A}^*\bar{\mathbf{x}}|\leqslant(1.5*10^{-14},\ 1.5*10^{-14},\ 1.5*10^{-14},\ 1.5*10^{-14})^{\mathrm{T}}=\mathbf{r}.$$

В соответствии с теоремой 9.1 потребуем соблюдения неравенства

$$\bar{\mathbf{r}}\leqslant\delta\mathbf{b}+\delta\mathrm{A}|\bar{\mathbf{x}}|,$$

или

$$\bar{\mathbf{r}}\leq(\varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon)^{\mathrm{T}}+\begin{pmatrix}\varepsilon&0&0&0\\ \varepsilon&\varepsilon&0&0\\ \varepsilon&\varepsilon&\varepsilon&0\\ \varepsilon&\varepsilon&\varepsilon&\varepsilon\end{pmatrix}\begin{pmatrix}1\\1\\1\\1\end{pmatrix}=(\varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon)^{\mathrm{T}}+(\varepsilon,\ 2\varepsilon,\ 3\varepsilon,\ 4\varepsilon)^{\mathrm{T}}$$

$$=(2\varepsilon,\ 3\varepsilon,\ 4\varepsilon,\ 5\varepsilon)^{\mathrm{T}}=(1*10^{-14},\ 1.5*10^{-14},\ 2*10^{-14},\ 2.5*10^{-14})^{\mathrm{T}}$$

Это неравенство *не* удовлетворяется, поскольку

$$\bar{\mathbf{r}}_1=1.5*10^{-14}>2\varepsilon=1*10^{-14}.$$

Конечно, нам следовало бы применить теорему к фактическому значению невязки $\mathbf{r}$, а не к $\bar{\mathbf{r}}$, и, действительно, дальнейшие вычисления показывают, что r_1 точно равно нулю. Таким образом, $\bar{\mathbf{x}}$ является приемлемым решением, поскольку

$$(0,\ 1.5*10^{-14},\ 1.5*10^{-14},\ 1.5*10^{-14})^{\mathrm{T}}$$
$$\leqslant(1*10^{-14},\ 1.5*10^{-14},\ 2*10^{-14},\ 2.5*10^{-14})^{\mathrm{T}}.$$

Пример 9.9: Анализ неопределенной системы с матрицей Гильберта десятого порядка, решенной пакетом LINPACK (см. пример 8.2). Теперь используем теорему 9.1 для того, чтобы приблизительно ответить на вопрос: каковы наименьшие возмущения $\delta\mathrm{A}$ и $\delta\mathbf{b}$ для которых решение, полученное пакетом LINPACK, будет приемлемо? Иными словами, предположив, что

$$\delta\mathrm{A}=\varepsilon_{\mathrm{A}}(a_{ij}),$$
$$\delta\mathbf{b}=\varepsilon_{\mathrm{b}}(b_i)^{\mathrm{T}},$$

мы хотим оценить такие значения ε_{A} и ε_{b}, для которых

$$|\mathbf{b}^*-\mathrm{A}^*\bar{\mathbf{x}}|\leqslant\delta\mathbf{b}+\delta\mathrm{A}|\bar{\mathbf{x}}|.$$

Заметим, что согласно обратному анализу ошибок $\bar{\mathbf{x}}$ является решением системы $(\mathrm{A}+\mathrm{E})\bar{\mathbf{x}}=\mathbf{b}$, если $|e_{ij}|\leqslant9\varepsilon$. Вместе с $\varepsilon=0.5*10^{-14}$ и $|a_{ij}|=1/19$ это говорит о том, что $\varepsilon_{\mathrm{b}}=0$ и $\varepsilon_{\mathrm{A}}=9.5*10^{-14}$ — подходящий выбор.

Ограничим невязку $\mathbf{r}$ постоянным вектором

$$\bar{\mathbf{r}}=(\|\mathbf{r}\|,\ \|\mathbf{r}\|,\ \ldots,\ \|\mathbf{r}\|)^{\mathrm{T}}$$
$$=(2.2*10^{-12},\ 2.2*10^{-12},\ \ldots,\ 2.2*10^{-12})^{\mathrm{T}}.$$

Нам надо найти такие значения ε_A и ε_b, что

$$|\bar{r}| \leq (\varepsilon_b, 0, 0, \ldots, 0)^T + \begin{pmatrix} \varepsilon_A & \varepsilon_A/2 & \ldots \\ \varepsilon_A/2 & & \vdots \\ \vdots & \ldots & \varepsilon_A/19 \end{pmatrix} \begin{pmatrix} 100.005711648351 \\ \vdots \\ 924185.572651327 \end{pmatrix}$$

Это соотношение есть совокупность 10 обычных неравенств:

$$2.2 * 10^{-12} \leqslant \varepsilon_b + \varepsilon_A\ (1,\ 1/2,\ \ldots,\ 1/9)^T\ |\bar{x}|,$$

$$2.2 * 10^{-12} \leqslant \varepsilon_A\ (\text{строка } j \text{ матрицы } A)^T\ |\bar{x}|,\quad j = 2, 3, \ldots, 10.$$

Скалярные произведения (строка j матрицы A)$^T|\bar{x}|$, j=2, 3, ..., 10, равны

$$4.8 * 10^6,\ 4.2 * 10^6,\ 3.7 * 10^6,\ 3.3 * 10^6,\ 3.0 * 10^6,$$
$$2.8 * 10^6,\ 2.5 * 10^6,\ 2.4 * 10^6,\ 2.2 * 10^6,\ 2.1 * 10^6.$$

Эти значения велики, поскольку абсолютные значения исключают возможность взаимного уничтожения слагаемых в скалярных произведениях. Только $a_{1,10}\bar{x}_{10}$ всегда не меньше 48 600, а это означает, что ε_A не может быть больше, чем $2.2*10^{-12}/48\,600 = 4.5*10^{-17}$, для того чтобы выполнялись неравенства

$$2.2*10^{-12} \leqslant \varepsilon_A |a_{1,10}|\ |\bar{x}_{10}| \leqslant \varepsilon_A\ (\text{строка } j \text{ матрицы } A)^T|\bar{x}|.$$

Последнее неравенство определяет ε_A и ε_b так:

$$\varepsilon_A = \frac{2.2 * 10^{-12}}{2.1 * 10^6} \sim 10^{-18},$$

$$\varepsilon_b = 0.$$

Поэтому возмущение A на величину 10^{-18} достаточно, чтобы удовлетворить условиям теоремы 9.1.

Таким образом, мы приходим к заключению, что решение, полученное пакетом LINPACK, является приемлемым для возмущений, *гораздо меньших*, чем это предполагается обычным обратным анализом ошибок. Вывод, полученный на основе рассмотрения этого примера, может быть перефразирован так: существует возмущение этой задачи порядка 10^{-18}, такое, что решение, вычисленное пакетом LINPACK, является точным решением возмущенной задачи.

Задачи гл. 9

1. Описать различие между плохо обусловленной задачей и плохо обусловленными вычислениями.

2. Пусть имеется приближенное решение системы

$$x + 2y = 3,$$
$$2x + y = 3,$$

дающее невязку $(0.01,\ 0.023)^T$. Выполнить обратный анализ ошибок для этой задачи и приближенного решения $(1.012,\ 0\ 999)^T$. Не изменять правую часть и не использовать общую формулу примера 9 7, применить идею обратного анализа ошибок непосредственно.

3. Для каждой из линейных систем

$$\text{(i)}\quad \begin{aligned} 3x+7y&=1,\\ 0.04x+8y&=24, \end{aligned} \qquad \text{(ii)}\quad \begin{aligned} x+\ y&=401\\ 0.01x-2y&=2, \end{aligned} \qquad \text{(iii)}\quad \begin{aligned} 7.5x+10.5y&=18,\\ 7x+\ 10y&=17 \end{aligned}$$

выполнить следующее:

(а) Вычислить стандартное и естественное числа обусловленности.

(б) Решить системы, используя вычисления с тремя десятичными цифрами, и оценить точность вычисленного решения с помощью естественного и стандартного чисел обусловленности.

(в) Вычислить оценку ошибок Эйрда—Линча для решения, полученного в (б).

(г) Переставить два уравнения и повторить (б) без выбора ведущего элемента.

(д) Сравнить оценки ошибок с фактическими ошибками.

4. Рассмотреть линейную систему с параметром θ:

$$\begin{aligned} x\operatorname{ctg}\theta+y\operatorname{cosec}\theta&=b_1,\\ -x\operatorname{cosec}\theta-y\operatorname{ctg}\theta&=b_2. \end{aligned}$$

(а) Вычислить стандартное число обусловленности матрицы системы.

(б) Показать, что собственные значения матрицы системы по величине близки к 1.

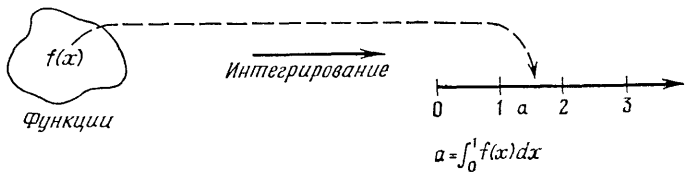
(в) Какие значения параметра θ должны вызвать трудности при решении этой системы? Какие соответствующие значения b_1 и b_2 должны сделать вычисления особенно плохо обусловленными? Дать набор особенных значений θ, b_1 и b_2.

5. Рассмотреть функцию $f(x)$ для $0\leqslant x\leqslant 1$. Один из способов измерения «величины» $f(x)$ состоит в вычислении:

$$\max_{0\leqslant x\leqslant 1}|f(x)|.$$

(а) Что означает: «$g(x)$ лежит внутри сферы радиуса 10^{-3} и с центром $f(x)$»?

(б) Что означает: «$g(x)$ лежит внутри относительной сферы радиуса 10^{-3} и с центром $f(x)$»?



Процесс интегрирования, т. е. вычисления $a=\int_0^1 f(x)dx$, переносит нас из пространства функций в пространство вещественных чисел.

(в) Какова абсолютная обусловленность этого процесса?

(г) Какова относительная обусловленность этого процесса?

(д) Заменить операцию «интегрирования» операцией «**дифференцирования**», т. е. рассмотреть

$$a(x)=\frac{df(x)}{dx}$$

и ответить на вопросы (в) и (г).

Заметим, что теперь этот процесс ставит функции в соответствие функциям, и мы должны измерять их величины так, как это было указано в начале задачи.

6. Различные виды ошибок или неопределенностей могут возникнуть при решении системы $Ax=b$, и мы разработали набор схем (как обратных, так и прямых) оценки влияния этих неопределенностей.

(а) Перечислить три такие схемы, или оценки ошибок, изложенные в этой книге.

(б) Описать на двух-трех примерах их цели или использование.

(в) Описать на двух-трех примерах их сильные и слабые стороны.

(г) Что можно сказать на основе этих оценок относительно конкретной задачи

$$A x=\begin{pmatrix}0.932 & 0.443\\ 1.237 & 0.587\end{pmatrix} x=\begin{pmatrix}0.699\\ 3.242\end{pmatrix},$$

если используются вычисления с шестью десятичными цифрами? *Указание.* В некоторых случаях одно или более из следующих чисел могут быть полезны для формулирования результатов:

$$\|A\| \sim 2, \qquad \|A^{-1}\| \sim 2400,$$

$$x_{\text{истинное}} = (1131.083869,\ -2378.038749)^T,$$

$$x_{\text{вычисленное}} = (1148.01,\ -2413.66)^T,$$

$$\text{невязка} = (0.006,\ -0.028)^T.$$

7. Рассмотреть следующую задачу решения $Ax=b$ при использовании вычислений с шестью десятичными цифрами:

$$\begin{pmatrix}0.932165 & 0.443126 & 0.417632\\ 0.712345 & 0.915312 & 0.887652\\ 0.632165 & 0.514217 & 0.493909\end{pmatrix} x=\begin{pmatrix}0.876132\\ 0.815327\\ 0.012345\end{pmatrix}, \quad \|A^{-1}\|_\infty \sim 1.4E+5.$$

Вычисленное решение

$$\bar{x}=(0.495702E+3,\ -0.236728E+5,\ 0.240135E+5)^T$$

имеет невязку

$$r=(0.02721797,\ 0.07109741,\ 0.01931627)^T.$$

Выполнить следующий анализ ошибок для этой задачи:

(а) Оценить $\dfrac{\|x-\bar{x}\|}{\|x\|}$ при помощи стандартного числа обусловленности.

(б) Оценить $\dfrac{\|x-\bar{x}\|}{\|x\|}$ при помощи естественного числа обусловленности.

(в) Получить обратную оценку ошибок, используя общие оценки, полученные в примере 9.7.

8. Рассмотреть вычисления с четырьмя десятичными цифрами (все числа представлены в виде $0.xxxx\ 10^{-p}$). Предположить, что арифметическое устройство допускает ошибку самое большее на 1/2 разряда в четвертой значащей цифре для любой арифметической операции. Например, $\zeta_{\text{вычисл}}=x+y$ отличается от $\zeta_{\text{истин}}$ самое большее на 1/2 разряда в четвертой значащей цифре. Необходимо решить систему $Ax=b$ методом исключения Гаусса, где

$$A=\begin{pmatrix}2.xxx & 1.xxx & 0\\ 1.xxx & 2.xxx & -1.xxx\\ 0 & 1.xxx & 2.xxx\end{pmatrix}.$$

Символом x обозначены цифры, которые известны, но не должны быть использованы в этой задаче.

(а) Выполнить обратный анализ ошибок *(для этой конкретной матрицы)* для процесса треугольного разложения A; не применять общую формулу из примера 9.7.

(б) Что можно сказать об этом способе получения информации об ошибках?

9. Спроектировать и выполнить небольшой вычислительный эксперимент для проверки предположения о том, что обусловленность относительного возмущения матрицы равна 1. Конкретно, взять две матрицы (одна из них плохо обусловлена, другая — хорошо обусловлена) шестого порядка и применить к ним возмущения I+P, где $\|P\|$ мала по сравнению с 1 и велика по сравнению с ошибками округления используемой вычислительной машины. Вычислить

$$\frac{\|x^* - \bar{x}\|}{\|x\|}$$

и сравнить с $\|P\|$. Повторить это для нескольких P.

10. Выполнить прямой анализ ошибок для метода исключения Гаусса при решении системы третьего порядка из примера 9 7; использовать те же самые предположения относительно вычислений.

11. Пусть A — матрица Гильберта порядка N, где порядок N выбран так, чтобы получить приблизительно одну верную цифру при решении системы $Ax=b$ на вашей вычислительной машине. Взять EPSA$=10*\varepsilon$, где ε — ошибка округления этой машины. Выполнить возмущение матрицы A двумя способами: A (I+ +P) с $|p_{ij}| \leqslant$ EPSA и A+E с $|e_{ij}| \leqslant$ EPSA. Взять $b=(1, 1, \ldots, 1)^T$ и решить систему $Ax=b$ без и с этими возмущениями.

(а) Сравнить изменения в вычисленных решениях, вызванные двумя различными способами возмущения матрицы системы.

(б) Сравнить изменения в вычисленном решении с оценками ошибок, полученными соответствующими составными формулами оценок.

12. Применить теорему 9.1 к решению, полученному в задаче 3, часть (б), чтобы получить такое минимальное возмущение задачи, при котором вычисленное решение станет точным решением. Сравнить полученный результат с результатом применения обратного анализа ошибок (пример 9.7) к этой задаче

13. Пусть $|e_{ij}| \leqslant |a_{ij}|$ для всех i, j и $\|x\|_\infty \leqslant 1$.

Показать, что

$$\max_E \|Ex\|$$

достигается при $|e_{ij}| = |a_{ij}|$. *Указание.* Компонентами Ex являются линейные функции от переменных e_{ij}.

14. Рассмотреть вектор $b=(2.5, 1.5, 0.05)^1$ и матрицу

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0.1 \end{pmatrix}.$$

Пусть x — решение системы $Ax=b$ и E — матрица с элементами $|e_{ij}| \leqslant |a_{ij}|$ EPSA. Показать, что $\|Ex\| \leqslant 2.5$. *Указание.* Использовать задачу 13.

Вычислить обе части неравенства

$$\|A^{-1}Ex\| \leqslant \|A^{-1}\| \, \|A\| \, \|x\| \text{ EPSA}$$

и показать, что независимо от выбора E правая часть неравенства по крайней мере в 18 раз превосходит левую часть.

15. Рассмотреть A, **x** и **b** из задачи 14 и изменить ограничения на E так, чтобы $||E|| \leqslant ||A||$ EPSA. Показать, что всегда существует такая матрица E, что

$$||A^{-1}\mathbf{E}\mathbf{x}|| = ||A^{-1}||\,||A||\,||\mathbf{x}||\ \text{EPSA}.$$

Указание. Этот результат имеет место для любых A и **x**. Найти такой вектор **y**, $||\mathbf{y}|| = 1$, что $||A^{-1}\mathbf{y}|| = ||A^{-1}||$, и затем построить такую матрицу вращения R, $||R|| = 1$, что $R\mathbf{x} = ||\mathbf{x}||\mathbf{y}$. Построить E на основе R.

16. Решить задачу 14 при следующем предположении относительно E:

$$|e_{ij}| \leqslant \max_{i,j} |a_{ij}| * \text{EPSA}, \quad \text{если } |a_{ij}| > 0,$$
$$= 0, \quad \text{если } a_{ij} = 0.$$

Показать, что множитель равен 2.75.

17. Решить задачу 14 при следующем предположении относительно E:

$$|e_{ij}| \leqslant \|A\| * \text{EPSA}, \quad \text{если } |a_{ij}| > 0,$$
$$= 0, \quad \text{если } a_{ij} = 0.$$

Показать, что множитель равен 1.8333.

Глава 10

ИТЕРАЦИОННЫЕ МЕТОДЫ

Алгоритм исключения Гаусса решения системы $\mathbf{Ax}=\mathbf{b}$ представляет собой конечный, или *прямой метод*. В конце вычислений получается точный ответ, если оставить без внимания ошибки округления. Итерационные методы решения $\mathbf{Ax}=\mathbf{b}$ являются бесконечными методами и вычисляют только приближенные ответы. Они легко формулируются и широко используются. Однако их следует использовать только при определенных обстоятельствах, и поэтому здесь мы очертим основной круг идей и дадим рекомендации относительно того, когда они могли бы быть уместны. Если предполагается использование итерационных методов, то рекомендуется более тщательное изучение задачи и конкретного метода. Итерационные методы привлекательны для разреженных матриц, поскольку они требуют гораздо меньше оперативной памяти, чем прямые методы, и могут быть использованы, несмотря на то что требуют больше времени на исполнение. Уравнения в частных производных часто приводят к большим разреженным линейным системам, для решения которых заманчиво применение итерационных методов.

10.А. ВВЕДЕНИЕ В ИТЕРАЦИОННЫЕ МЕТОДЫ

Мы начнем с двух старых и простых методов; первый из них — *метод Якоби* — проиллюстрируем на следующей линейной системе:

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

Можно переписать эту систему в виде

$$\begin{aligned} x_1&=\frac{1}{3}\,[7-4x_2+x_3],\\ x_2&=\frac{1}{6}\,[-2-2x_1-3x_2],\\ x_3&=\frac{1}{4}\,[4+x_1-x_2], \end{aligned}$$

или в других обозначениях $\mathbf{x}=B\mathbf{x}+\mathbf{g}$, где

$$B=\begin{pmatrix} 0 & -\frac{4}{3} & \frac{1}{3} \\ -\frac{2}{6} & 0 & -\frac{3}{6} \\ \frac{1}{4} & -\frac{1}{4} & 0 \end{pmatrix}$$

и $\mathbf{g}=(7/3, -2/6, 4/4)^T$. Тем самым мы решили j-е уравнение явно относительно j-го неизвестного и перенесли все остальное в правую часть. Метод Якоби состоит теперь в определении начального приближения к решению, скажем $x^{(0)}$, и в применении итерации

$$\mathbf{x}^{(K+1)}=B\mathbf{x}^{(K)}+\mathbf{g}, \qquad K=0, 1, 2, \ldots .$$

Этот метод зависит от правильной нумерации уравнений и неизвестных.

Вторым старым методом является *метод Гаусса — Зейделя.* (Отметим, между прочим, что этот метод не был известен Зейделю и презирался Гауссом как бесполезный; таковы капризы исторической точности в науке.) Перепишем ту же самую систему в виде

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

В j-м уравнении мы перенесли в правую часть все члены, содержащие x_k для $k>j$. Эта запись может быть представлена в виде $(L+D)\mathbf{x}=U\mathbf{x}+\mathbf{g}$, где

$$L+D=\begin{pmatrix} 3 & 0 & 0 \\ 2 & 6 & 0 \\ -1 & 1 & 4 \end{pmatrix}, \qquad U=\begin{pmatrix} 0 & 4 & -1 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \mathbf{g}=\mathbf{b}.$$

Здесь D означает диагональ матрицы A, U — ее верхнюю треугольную часть и L — ее нижнюю треугольную часть. Затем мы применим итерацию

$$(L+D)\mathbf{x}^{(K+1)}=-U\mathbf{x}^{(K)}+\mathbf{g}, \qquad K=0, 1, 2, \ldots$$

после выбора начального приближения $\mathbf{x}^{(0)}$. Заметим, что эти вычисления почти такие же, как и в методе Якоби. Различие состоит в том, что новые значения x_i используются здесь сразу же по мере получения, в то время как в методе Якоби они не используются до следующей итерации.

10.А.1. Основной анализ сходимости

Обычно итерационные методы для исследования сходимости представляются в одной из следующих двух форм:

$$\mathbf{x}^{(K+1)}=\mathbf{x}^{(K)}+H^{(K)}[\mathbf{b}-A\mathbf{x}^{(K)}] \text{ (используется невязка),}$$

или $$\mathbf{x}^{(K+1)}=B^{(K)}\mathbf{x}^{(K)}+\mathbf{g}^{(K)},$$

где $H^{(K)}$, $B^{(K)}$ и $\mathbf{g}^{(K)}$ могут изменяться вместе с изменением K. Итерация называется *стационарной*, если $H^{(K)}$, $B^{(K)}$ и $\mathbf{g}^{(K)}$ постоянны, т. е. не зависят от K. Как метод Якоби, так и метод Гаусса—Зейделя являются стационарными. Любой метод, имеющий какую-либо практическую пользу, должен при $K\to\infty$ иметь вид

$$\mathbf{x}^*=\mathbf{x}^*+H^{(K)}[\mathbf{b}-A\mathbf{x}^*],$$
$$\mathbf{x}^*=B^{(K)}\mathbf{x}^*+\mathbf{g}^{(K)},$$

где $\mathbf{x}^*$ — истинное решение $A\mathbf{x}=\mathbf{b}$. Если ввести $\boldsymbol{\varepsilon}_K=\mathbf{x}^*-\mathbf{x}^{(K)}$, то после небольших вычислений получим

$$\boldsymbol{\varepsilon}_{K+1}=[I-H^{(K)}A]\,\boldsymbol{\varepsilon}_K,$$

или $$\boldsymbol{\varepsilon}_{K+1}=B^{(K)}\boldsymbol{\varepsilon}_K.$$

Вспомним, что $\rho(A)$ обозначает спектральный радиус матрицы A, который представляет собой модуль наибольшего (по абсолютной величине) собственного значения матрицы A. Мы сформулируем без доказательства основную теорему, которая показывает, что матрицы $I-HA$ и B определяют сходимость итерационного метода.

Теорема 10.1. *Необходимым и достаточным условием сходимости стационарной итерации является*

$$\rho(I-HA)<1 \text{ } \textit{или} \text{ } \rho(B)<1.$$

Подобный результат имеет место и в случае нестационарной итерации, для которой эти условия должны выполняться при всех достаточно больших значениях K. Заметим, что эти итерационные методы относятся к классу сжатых *отображений* $x_{m+1}=f(x_m)$, и теорема 10.1 представляет собой просто обобщение того факта, что для сходимости таких итераций требуется выполнение условия $|f'(x)|<1$.

Можно использовать любые доступные способы оценки $\rho(B)$ и пытаться определить, является ли какой-либо конкретный метод сходящимся. Напомним, что $\rho(B)\leqslant\|B\|$ для любой нормы, и простейшими нормами для вычислений оказываются 1- и ∞-нормы. Для *процесса Якоби* имеем:

$$\mathbf{x}^{(K+1)}=-D^{-1}(A-D)\mathbf{x}^{(K)}+D^{-1}\mathbf{b},$$

т. е. $B=-D^{-1}(A-D)=I-D^{-1}A$. Использование для оценок 1- и ∞-норм приводит к критерию сходимости, основанному на *диа-*

гональном преобладании:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}| \quad \text{или} \quad |a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^{N} |a_{ij}|.$$

Если матрица A имеет диагональное преобладание, то $\|B\|<1$, и итерационный метод Якоби сходится. Для *процесса Гаусса — Зейделя* имеем

$$\begin{aligned} \mathbf{x}^{(K+1)} &= -(D+L)^{-1}U\mathbf{x}^{(K)} + (D+L)^{-1}\mathbf{b} \\ &= [I-(D+L)^{-1}A]\,\mathbf{x}^{(K)} + (D+L)^{-1}\mathbf{b}, \end{aligned}$$

т. е. $B=I-(D+L)^{-1}A$. Выражая этот процесс через невязку $\mathbf{r}=\mathbf{b}-A\mathbf{x}$, получим

$$\mathbf{x}^{(K+1)}=\mathbf{x}^{(K)}-(D+L)^{-1}[\mathbf{b}-A\mathbf{x}^{(K)}].$$

Поэтому $H=(D+L)^{-1}$.

Сходимость имеет место, когда матрица B или $I-HA$ «малы». Для метода Якоби это означает, что матрица $D^{-1}A-I$ мала или что D^{-1} является достаточно хорошим приближением к A^{-1}. Для метода Гаусса — Зейделя матрица $(D+L)^{-1}A-I$ должна быть мала, или $(D+L)^{-1}$ должна хорошо аппроксимировать A^{-1}. Эти условия могут быть сформулированы по-другому так: метод Якоби предполагает, что матрица A «почти диагональная», а метод Гаусса — Зейделя предполагает, что матрица A «почти нижняя треугольная».

10.A.2. Три распространенных заблуждения

Существуют три широко утвердившихся, но совершенно ошибочных убеждения относительно итерационных методов. Первые два заблуждения не очень опасны, но третье заблуждение — уже серьезно. Ниже приводятся три утверждения, которые устраняют эти ошибки.

1. *Диагональное преобладание не требуется для сходимости как метода Якоби, так и метода Гаусса — Зейделя.* Достаточно рассмотреть только матрицу

$$A=\begin{pmatrix} 8 & 2 & 1 \\ 10 & 4 & 1 \\ 50 & 25 & 2 \end{pmatrix},$$

чтобы убедиться, что метод Гаусса — Зейделя может сходиться очень быстро без диагонального преобладания. Матрица A должна быть «почти диагональной» для сходимости метода Якоби, однако это не совсем то же самое, что диагональное преобладание. Заметим, что *столбцовое диагональное преобладание*

$$|a_{ii}| > \sum_{j \neq i} |a_{ji}|$$

также достаточно для сходимости, и легко построить примеры матриц, обладающих одним из этих свойств, но не обладающих другим. С несколько бóльшими усилиями можно найти примеры матриц, не обладающих ни одним из видов диагонального преобладания, но для которых метод Якоби сходится.

2. *Неверно, что если метод Якоби сходится, то метод Гаусса — Зейделя сходится еще лучше.* Существуют примеры, для которых метод Якоби сходится, а Гаусса — Зейделя — расходится. Источником этого заблуждения является то обстоятельство, что *если A — симметричная положительно определенная матрица*, то метод Гаусса—Зейделя сходится в два раза быстрее метода Якоби.

3. *Итерационные методы для плохо обусловленных задач нисколько не лучше, чем метод исключения Гаусса.* Часто считают, что источник неприятностей при применении исключения Гаусса для плохо обусловленных задач состоит в устойчивом накоплении тысяч или миллионов ошибок в процессе вычислений. Это не так: неточность часто возникает почти в полном объеме из-за единственного арифметического шага в исключении. В любом случае итерационные методы, примененные к плохо обусловленным задачам, дают такую же чепуху, что и прямые методы.

10.Б. КОГДА СЛЕДУЕТ ПРИМЕНЯТЬ ИТЕРАЦИОННЫЕ МЕТОДЫ

Итерационные методы используются прежде всего тогда, когда матрица A (и, следовательно, матрицы B и H) разрежена и имеет большой порядок. Если B или H разрежены, скажем, имеют p ненулевых элементов в одной строке, то одна итерация потребует pn операций для матрицы порядка n. Таким образом, K итераций требуют pKn операций, что меньше общего числа операций метода исключения Гаусса для полных матриц, если выполнено условие

$$K < \frac{n^2}{3p}.$$

Если A — ленточная матрица с шириной ленты d, которая мала по сравнению с n, то число операций метода Гаусса составляет около d^2n, в то время как K итераций потребуют около $2dnK$ операций. Поэтому, для того чтобы итерационный метод давал уменьшение числа арифметических операций для ленточной матрицы, необходимо выполнение условия

$$K < \frac{d}{2}.$$

Для матриц, возникающих при решении уравнений в частных производных, ширина ленты обычно приблизительно равна $\sqrt{n}$, однако лента сама по себе разрежена так, что p равно 4 или 6 (или около того). При таких условиях затраты метода Гаусса составля-

ют n^2 операций, тогда как затраты на K итераций возможно составят $6Kn$ операций. Поэтому для применения итерационных методов должно выполняться условие $K<n/6$. Однако более важным условием применения итерационных методов могут оказаться ограничения на размеры требуемой оперативной памяти, которые составляют n^2 машинных слов для метода Гаусса и pn — для итерационного метода. Для такого рода задач n может лежать в пределах от 100 до 1 000 000, причем $n=100\,000$ не является необычным случаем.

10.Б.1. SOR и методы подавления компонент

Если предположить, что итерационные методы могут быть полезны, то следует изучить два дополнительных метода. Первый из них — метод *последовательной верхней релаксации*, или, для краткости, SOR.

Идея этого метода состоит в том, что берется приращение, полученное в результате шага итерации Гаусса — Зейделя, умножается на некоторую величину q и прибавляется к текущему значению решения. Символически это можно представить так:

$$\mathbf{x}^{(K+1)}=\mathbf{x}^{(K)}+Q\,[\mathbf{x}_{GS}^{(K+1)}-\mathbf{x}^{(K)}],$$

где Q — диагональная матрица параметров релаксации, а $\mathbf{x}_{GS}^{(K+1)}$ — результат итерации Гаусса — Зейделя. SOR имеет небольшое отличие, состоящее в том, что новые значения **x** используются сразу же, как только они вычислены:

$$\mathbf{x}^{(K+1)}=\mathbf{x}^{(K)}+QD^{-1}\,[\mathbf{b}-(D+U)\,\mathbf{x}^{(K)}-L\mathbf{x}^{(K+1)}].$$

Матрица, определяющая сходимость метода, имеет вид

$$(I+QD^{-1}L)^{-1}[I-QD^{-1}\,(D+U)],$$

или

$$(D+QL)^{-1}\,(D-DQ-QU).$$

Знаменитый результат Дэвида Янга (David Young) состоит в установлении соотношения между скоростью сходимости метода SOR и скоростью сходимости метода Якоби для некоторого класса «блочно трехдиагональных матриц». Как в теории, так и на практике затруднительным является правильный выбор параметров релаксации (верхней релаксации) в матрице Q.

Для случая стандартной конечно-разностной аппроксимации уравнения Лапласа $U_{xx}+U_{yy}=0$ в прямоугольнике получается ленточная матрица с шириной ленты $\sqrt{n}$ и пятью ненулевыми элементами в строке, причем второй, третий и четвертый элементы образуют трехдиагональные блоки. Можно показать, что требуется около $\sqrt{n}$ итераций, чтобы уменьшить ошибку решения линейных уравнений до того же самого размера, который имеет ошибка в аппроксимации уравнения Лапласа.

Поэтому затраты метода SOR составляют порядка $n^{3/2}$ операций, тогда как затраты метода Гаусса — порядка n^2 операций *при условии, что мы знаем оптимальные множители верхней релаксации* (в матрице Q). Эти множители известны для данного конкретного случая, однако для других случаев оптимальными могут быть совсем другие множители релаксации. Имеет место экономия не только в числе операций, но также и в использованной памяти. Полное изложение такого типа методов приводится в книге Янга (1971); более простое их изложение представлено в работах Форсайта и Вазова (Forsythe, Wasow, 1960), Фокса (Fox, 1965, гл. 8) и Варги (Varga, 1962, гл. 4) (см. список литературы в конце книги).

Второй метод, или класс методов, представляет собой метод *подавления компонент* (это название не общепринято). Эти методы технически сложны, но зато в некоторых случаях очень мощны. Идея этих методов состоит в построении итераций вида:

$$\mathbf{x}^{(K+1)}=\mathbf{x}^{(K)}+f(A)\,[\mathbf{b}-A\mathbf{x}^{(K)}],$$

где f (A) — функция (обычно полином) от матрицы A. Положив снова $\varepsilon_K=\mathbf{x}^*-\mathbf{x}^{(K)}$, получим

$$\varepsilon_{K+1}=-[I-Af(A)]\,\varepsilon_K=g(A)\varepsilon_K.$$

Если g (A) обладает хорошими свойствами (например, является полиномом или аналитической функцией), то g (A)* (собственный вектор)=g (собственное значение). Теперь предположим, что нам известны некоторые сведения относительно собственных значений матрицы A, скажем, что все они содержатся в некоторой области Z. Можно выбрать такую функцию g (A), что она мала в Z, причем вне этой области g (A) может быть любой, за исключением одной точки: *обязательно g (0)=1, иначе функция f (A) будет содержать* A^{-1} и итерации станут практически неприменимы.

Наиболее широко известный специальный случай такого рода методов представляет собой *полуитерационный метод Чебышева* для положительно определенных матриц. В этом случае областью Z является интервал [a, T] вещественной оси и «наилучшей» функцией оказывается полином Чебышева, связанный с [a, T]. Возможно организовать так вычисления, что этот полином от A никогда явно не формируется, а общий объем затрат оказывается приемлемым.

Рассмотренный подход включает в себя много других специальных методов, и хорошее их изложение содержится в гл. 9 книги Фаддеева и Фаддеевой (1963), в которой они называются *универсальными методами.*

Задачи гл. 10

1. Составить программу, реализующую методом Гаусса—Зейделя решение системы

$$2x_1-x_2=1,$$

$$-x_{i-1}+2x_i-x_{i+1}=0, \qquad i=2, 3, \ldots, n-1,$$
$$-x_{n+1}+2x_n=1$$

для $n=20$. Вычисления начать для $x^{(0)}=0$, а итерации закончить, когда

$$\frac{\|x^{(k)}-x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty}<10^{-6}.$$

Точное решение системы: $x_i=1$ для всех i. Какова достигнутая точность?

2. Сравнить время вычислительной машины, потребовавшееся для решения задачи 1 методом Гаусса—Зейделя, с тем временем, которое потребуется для метода исключения Гаусса. Метод Гаусса—Зейделя для задачи 1 сходится приблизительно за 466 итераций.

3. Применить 100 итераций по методу Якоби для задачи 1 и найти число итераций по методу Гаусса—Зейделя, которое потребуется для достижения той же точности.

4. Пусть A — верхняя треугольная матрица порядка N с $a_{ii}=0$ для всех i. Показать, что $A^N=0$. Использовать этот факт, чтобы показать, что как метод Якоби, так и метод Гаусса—Зейделя сходятся за конечное число итераций всякий раз, когда A — верхняя треугольная и невырожденная матрица. Сколько итераций потребуется методу Гаусса—Зейделя для сходимости, если A — нижняя треугольная матрица?

5. Рассмотреть следующую систему линейных уравнений, у которой матрица разбита на четыре подматрицы, а каждый из векторов решения и правой части разбит на два вектора:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

Определим следующим образом *блочный метод Якоби*. Каждая итерация состоит из двух шагов.

Шаг 1. Взять текущее приближение к **y** и решить систему $Ax=\bar{c}_1-By$ относительно **x**.

Шаг 2. Использовать вектор **x**, полученный на первом шаге, и решить систему $Dy=\bar{c}_2-Cx$ относительно **y**. Полученное значение **y** взять в качестве следующего приближения.

(а) Сформулировать этот метод в матричных обозначениях.

(б) Дать пример системы пятого порядка (A — матрица третьего порядка D — матрица второго порядка), для которой следует ожидать быстрой сходимости.

(в) Выписать матрицу, которая определяет сходимость этого метода.

6. Пусть для решения системы $Ax=b$ использовался метод Гаусса—Зейделя и на последней итерации $x^{198}-x^{197}<10^{-10}$ для каждой компоненты. Матрица A имеет порядок 500 и в каждой строке — самое большее 50 ненулевых элементов, причем все a_{ij} и b_i по величине порядка 1. Выполнить обратный анализ ошибок для этого случая.

7. Итерационный процесс решения $Ax=b$ строится следующим образом. Матрица A разбивается на три части T — трехдиагональная часть, U — лежащая выше T часть и L — лежащая ниже T часть. Результат умножения подматриц L и U на текущее приближение **x** переносится в правую часть системы и затем решается получившаяся трехдиагональная система для лучшего определения **x**. После этого вычисленное приближение умножается на множитель верхней релаксации w. Окончательный результат принимается за следующее приближение к **x**.

(а) Дать формулировку этого итерационного метода в матричных обозначениях.

(б) Найти матрицу, которая определяет сходимость метода.

(в) Определить число операций на каждой итерации этого метода.

8. Пусть требуется найти решение следующей системы методом Гаусса—Зейделя:

$$\begin{aligned} 2x+\ y-\frac{1}{2}z+\ 8w&=4,\\ 6x+\ 2y+8z+14w&=12,\\ x-6y+10z+\ 9w&=-1,\\ 2x+11y+3z-\ 4w&=4. \end{aligned}$$

(а) Описать, что прежде всего следует сделать.

(б) Выполнить несколько итераций (взять в качестве начального приближения нулевой вектор) и приблизительно оценить, сколько итераций следует сделать, чтобы получить шесть верных цифр в решении.

9. Применить методы Якоби и Гаусса—Зейделя к системе $A\mathbf{x}=(8, 8, 29)^T$ с начальным приближением $(-1, 1, 0)^T$ и с матрицей A из подраздела 10.А.2. Сколько потребуется итераций для этих методов, чтобы достигнуть точности 10^{-4}?

10. Рассмотреть следующую систему, которая «почти» разбита на две части:

$$\sum_{j=1}^{n} a_{ij}x_j+\sum_{j=1}^{m} e_{ij}y_j=c_i, \qquad i=1, 2, \ldots, n,$$

$$\sum_{j=1}^{n} h_{ij}x_i+\sum_{j=1}^{m} b_{ij}y_i=c_{i+n}, \qquad i=1, 2, \ldots, m.$$

Матрицы $E=(e_{ij})$ и $H=(h_{ij})$ «малы» по сравнению с матрицами A и B. Матрица A гораздо меньше, чем B, а B обладает существенным диагональным преобладанием. Можно применить следующую итерационную схему:

Шаг 1. Взять текущее приближение к **y** и вычислить **x** из первой части системы.

Шаг 2. Взять полученный вектор **x** и решить вторую часть системы методом Гаусса—Зейделя для вычисления уточненного приближения вектора **y**.

(а) Представить исходную линейную систему в матричной форме

(б) Выразить итерационные шаги 1 и 2 в матричной форме.

(в) Представить эту итерационную схему в матричной форме как стационарный процесс.

(г) Выписать матрицу, которая определяет сходимость этой схемы.

Глава 11

ЛИНЕЙНАЯ ЗАДАЧА НАИМЕНЬШИХ КВАДРАТОВ И РЕГРЕССИЯ

Линейная задача наименьших квадратов, или статистическая регрессия, относится к оптимизационным задачам. Предположим, нам нужно выбрать «наилучший» набор параметров, или коэффициентов, «наилучший» — в смысле наименьших квадратов. Критерий наименьших квадратов не является, как это многие полагают, особенно естественным. Его применение на практике не объясняется также наличием тщательно разработанной для этого критерия теорией. Доводы в пользу «принципа наименьших квадратов», основанные на «нормально распределенных случайных ошибках», подозрительны, поскольку очень редко известно много, если вообще что-либо известно, относительно распределения ошибок в реальной задаче. Однако факт остается фактом: критерий наименьших квадратов для выбора «наилучших» параметров вполне адекватен в широком многообразии ситуаций. Одной из сильных сторон критерия наименьших квадратов является то, что получающиеся оптимизационные задачи можно решать, непосредственно применяя методы и программное обеспечение матричного исчисления. В этой главе мы покажем, как это делается.

Математический механизм для наименьших квадратов существенно включает в себя ортогональность, и читатель может повторить соответствующие основные понятия из гл. 2.

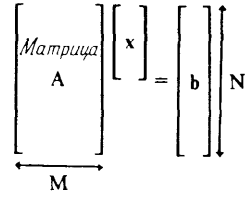
11.А. ЗАДАЧА НАИМЕНЬШИХ КВАДРАТОВ

Опишем три до некоторой степени различных случая, которые приводят к одной и той же математической задаче.

11.А.1. Переопределенные системы уравнений

Пусть даны N линейных уравнений с $M<N$ переменными. Это означает, что существует больше условий (уравнений), которые должны удовлетворяться, чем степеней свободы (переменных), и маловероятно, что одновременно могут быть удовлетворены все уравнения. Наглядно рассматриваемую линейную систему можно

изобразить следующим образом:



Поскольку уравнения не могут быть удовлетворены точно, мы можем попытаться удовлетворить их как можно лучше, т е. попытаться сделать величину вектора невязки **r** с компонентами

$$r_j = b_j - \sum_{i=1}^{M} a_{ij} x_i, \qquad j = 1, 2, \ldots, N,$$

настолько малой, насколько это возможно. Критерий наименьших квадратов основывается на использовании евклидовой (или квадратичной) нормы для вычисления величины вектора **r**; т. е. минимизируется

$$\sqrt{\sum_{j=1}^{N} r_j^2} = \| \mathbf{r} \|_2.$$

11.А.2. Функциональное приближение наименьшими квадратами

При реализации компилятора с языка Фортран возникает задача вычисления стандартных функций SIN(T), TAN(T) и др. Очевидно, при реализации должны использоваться арифметические операции, а это означает, например, что может быть применено следующее представление

$$\sin(t) \sim \text{полином степени } M-1 \text{ в точке } t.$$

Можно ввести функцию невязки r (t):

$$r(t) = \sin(t) - \sum_{i=1}^{M} x_i t^{i-1} \quad \text{для} \quad t \in [0, \pi/2].$$

Ясно, что мы не можем сделать r (t) равной нулю на всем отрезке. Поэтому попытаемся сделать невязку настолько малой, насколько это возможно в том смысле, чтобы выражение

$$\sqrt{\int_0^{\pi/2} r(t)^2 \, dt}$$

было минимальным.

Тем самым задача наименьших квадратов состоит в выборе таких коэффициентов x_i полинома, чтобы выполнялось указанное условие

минимальности. Может быть затруднительным (или невозможным) вычислять интегралы, если вместо sin (t) рассматривать arctg (t) или функцию Бесселя, поэтому вместо этого можно выбирать x_i из условия, чтобы выражение

$$\sqrt{\sum_{j=1}^{101} r(t_j)^2}$$

было минимальным. Здесь t_j — узлы равномерной сетки на отрезке $[0, \pi/2]$

Для реальных стандартных функций Фортрана используется другой, более естественный, но более трудно достижимый критерий: *коэффициенты x_i выбираются так, чтобы max|r (t)| был минимален.*

11.А.3. Регрессия и математическое моделирование

Рассмотрим процесс исследования какого-нибудь явления, которое включает в себя одну независимую и одну зависимую переменные. Примерами могут служить: вес коров как функция от возраста, стоимость ремонта автомобилей как функция от числа пройденных миль, доход в течение всей жизни как функция от полученного образования. В каждом случае мы знаем (или предполагаем или надеемся), что существуют несколько факторов, которые влияют на рассматриваемое соотношение. Пусть t — независимая переменная и пусть $f_i(t)$ выражают зависимости, согласно которым эти факторы изменяются (т. е. количество зерна или сена, идущего на корм коровам, как функция от времени, объем эксплуатационных расходов или интенсивность использования автомобилей на ферме как функция от числа пройденных миль; вид обучения или местожительство как функция от числа лет обучения) Наконец, пусть d (t) — зависимая переменная; введем линейную математическую модель

$$d(t) = \sum_{i=1}^{M} x_i f_i(t).$$

Теоретически должны существовать значения коэффициентов, которые превращают эту модель в точную

Предположим теперь, что нам не удалось найти такие значения коэффициентов Сделаем большое число измерений (наблюдений) и получим значения d_j для значений t_j, $j=1, 2, \ldots, N$, независимой переменной t Если $N=M$, то мы просто решаем линейную систему

$$d_j = d_i(t_j) = \sum_{i=1}^{M} x_i f_i(t_j), \qquad j = 1, 2, \ldots, N = M,$$

для определения неизвестных коэффициентов x_i. Однако жизнь

не так проста, и если мы возьмем N бо́льшим, чем M, то обнаружим, что коэффициенты x_i не могут быть выбраны так, чтобы превратить модель в точную для всех наблюдений. Расхождения могут быть из-за ошибок наблюдений, неполноты модели или случайностей, свойственных явлению (или комбинации всех трех факторов).

Для того чтобы получить «наилучшую» модель, мы решаем сделать невязку

$$r_j = d_j - \sum_{i=1}^{M} x_i f_i(t_j), \qquad j = 1, 2, \ldots, N,$$

настолько малой, насколько это возможно. Если выбрать критерий наименьших квадратов, то минимизируется выражение

$$\sqrt{\sum_{j=1}^{N} r_j^2}.$$

Таким образом, мы имеем три задачи, которые приводят к минимизации квадратного корня из суммы квадратов невязок. Легко видеть, что квадратный корень можно минимизировать при помощи непосредственной минимизации подкоренного выражения, и ниже мы попытаемся это сделать.

11.Б. ПОДХОД С ИСПОЛЬЗОВАНИЕМ НОРМАЛЬНЫХ УРАВНЕНИЙ

Мы использовали различные обозначения для каждого из этих примеров, однако последние два примера могут быть выражены в обозначениях первого примера. Положим $a_{ij} = t_j^{i-1}$ для задачи полиноминальной аппроксимации и $a_{ij} = f_i(t_j)$ для задачи регрессии. Таким образом, мы хотим достигнуть минимума суммы:

$$E = E(x_1, x_2, \ldots, x_M) = \sum_{j=1}^{N} \left[b_j - \sum_{i=1}^{M} x_i a_{ij} \right]^2.$$

Применяя дифференциальное исчисление для этой минимизационной задачи, положим производные от E по x_i равными нулю. Для $k = 1, 2, \ldots, M$ имеем

$$0 = \frac{\partial E}{\partial x_k} = \frac{\partial \sum [b_j - \sum x_i a_{ij}]^2}{\partial x_k} = 2 \sum [b_j - \sum x_i a_{ij}] a_{kj}.$$

Эти равенства для $k = 1, 2, \ldots, M$ могут быть записаны в виде:

$$\sum_{j=1}^{N} b_j a_{kj} = \sum_{i=1}^{N} \sum_{i=1}^{M} x_i a_{ij} a_{kj} = \sum_{i=1}^{M} x_i \sum_{i=1}^{N} a_{ij} a_{kj}.$$

Последние равенства образуют M линейных уравнений с M неизвестными x_i, которые могут быть выражены в матричных обозначе-

ниях так:

$$A^TAx=A^Tb.$$

Это — *нормальные уравнения.*

Мы можем следующим образом оценить затраты на формирование и решение нормальных уравнений. Чтобы вычислить один элемент матрицы A^TA, требуется N операций (одна операция=одно сложение+одно умножение), а всего должно быть найдено $M^2/2$ элементов (A^TA — симметричная матрица, поэтому нужно вычислять только половину из общего числа элементов). Поскольку нормальные уравнения симметричны, можно применить метод Холесского для их решения, который требует порядка $M^3/6$ операций. Таким образом, общие затраты (для больших значений M и N) составляют $M^2N/2+M^3/6$ операций.

Мы подошли к обсуждению двух других способов решения задачи наименьших квадратов. Подход с использованием нормальных уравнений настолько прост, что до тех пор, пока не возникают какие-либо трудности в его применении, не следует рассматривать другие подходы. Основная проблема здесь состоит в том, что распространенные выборы моделей приводят к плохо обусловленным матрицам A^TA, настолько плохим, что решения систем представляют собой случайные числа, не имеющие отношения к исходной задаче. Вспомним рассуждения в примере 8.1, связанные с матрицей Гильберта. Конечно, такая ситуация не всегда имеет место, однако наблюдается достаточно часто, чтобы считать этот подход ненадежным. Заметим, что число обусловленности матрицы A^TA является числом обусловленности матрицы A *в квадрате.*

Слабость этого подхода состоит также в том, что он настолько прост, что многие пользователи не видят истинного источника трудностей: почти линейная зависимость базисных векторов (столбцов матрицы A). Так, модель атлетических, или физических, характеристик школьников может содержать в качестве переменных:

возраст, вес, рост и силу.

Очевидно, эти характеристики сильно коррелируют и потому почти линейно зависимы. Модель экономии горючего для автомобилей может содержать в качестве переменных:

вес, скорость, мощность двигателя в лошадиных силах, объем двигателя, сопротивление ветра, трение вращения и давление в шинах.

Здесь опять имеется значительная корреляция между некоторыми из этих факторов (например, скорость, сопротивление ветра, трение вращения).

И наконец, не так уж редко применяется потенциально ката-

строфический подход, примером которого может послужить следующее рассуждение:

> «Я не знаю, какие переменные на самом деле влияют на интересующую меня величину; поэтому возьму все, что придет в голову, и пусть машина разбирается».

Однако далеко не всегда такие надежды оправдываются, в результате можно получить чепуху или обнаружить, что в действительности небольшие случайные вариации в исходных данных определяют, какие переменные будут выбраны в качестве значимых.

11.В. ПОДХОД С ИСПОЛЬЗОВАНИЕМ ОРТОГОНАЛИЗАЦИИ ГРАМА — ШМИДТА

Если векторы a_i, $i=1, 2, \ldots, M$ (столбцы матрицы A), ортогональны, то задача наименьших квадратов легко решается. Действительно, матрица A^TA будет диагональной, поскольку строки A^T представляют собой столбцы A, и ненулевые скалярные произведения $a_i^Ta_i$ будут располагаться на диагонали матрицы A^TA. Поэтому мы можем дать явное выражение для решения, определяющего наилучшие коэффициенты для этого случая:

$$x_i = \frac{b^Ta_i}{a_i^Ta_i}.$$

Итак, мы можем попытаться сделать столбцы матрицы A ортогональными вместо того, чтобы формировать и решать нормальные

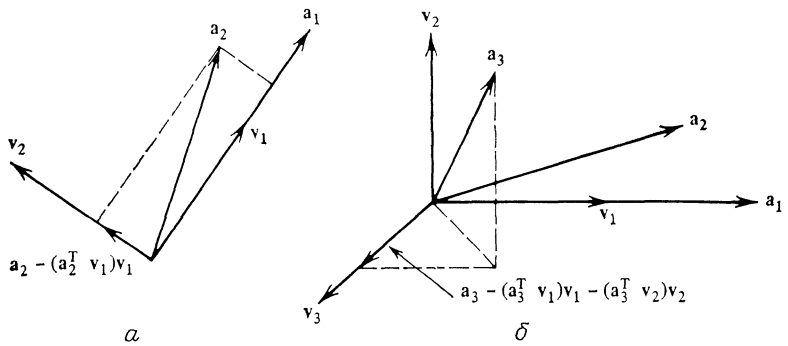


Рис. 11.1. Геометрическое представление процесса Грама—Шмидта для случаев (а) двух и (б) трех векторов.

уравнения. Классическим методом для этого является *ортогонализация Грама — Шмидта*. В явном виде задача состоит в следующем: для данных векторов $a_1, a_2, \ldots, a_M$ найти эквивалентный набор векторов $v_1, v_2, \ldots, v_M$, которые являются ортогональными. Нор-

мирование векторов $\mathbf{v}_i$ так, чтобы их длины равнялись 1, упрощает вычисления и последующее использование этих векторов. Идея совсем проста. Возьмем $\mathbf{a}_1$ и разделим его на евклидову длину $\|\mathbf{a}_1\| = (\mathbf{a}_1^T\mathbf{a}_1)^{1/2}$, чтобы получить $\mathbf{v}_1$. Теперь вычтем компоненту вектора $\mathbf{a}_2$ в направлении $\mathbf{v}_1$. Остаток будет ортогонален $\mathbf{v}_1$ (см. диаграмму на рис. 11.1). Вектор-остаток нормируется делением на его длину, в результате чего получим вектор $\mathbf{v}_2$. В алгебраической записи имеем

$$\mathbf{v}_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|},$$

$$\mathbf{v}_2 = \frac{\mathbf{a}_2 - (\mathbf{a}_2^T\mathbf{v}_1)\,\mathbf{v}_1}{\|\mathbf{a}_2 - (\mathbf{a}_2^T\mathbf{v}_1)\,\mathbf{v}_1\|}.$$

Процесс продолжается вычитанием компонент вектора $\mathbf{a}_3$ в направлениях $\mathbf{v}_1$ и $\mathbf{v}_2$ и получением вектора-остатка. Он нормируется, чтобы получить вектор $\mathbf{v}_3$ (см. рис. 11.1). В алгебраической записи имеем

$$\mathbf{v}_3' = \mathbf{a}_3 - (\mathbf{a}_3^T\mathbf{v}_1)\,\mathbf{v}_1 - (\mathbf{a}_3^T\mathbf{v}_2)\,\mathbf{v}_2,$$
$$\mathbf{v}_3 = \mathbf{v}_3'/\|\mathbf{v}_3'\|.$$

Очевидно, что в общем виде шаг этого процесса выглядит так:

$$\mathbf{v}_k' = \mathbf{a}_k - \sum_{j=1}^{k-1} (\mathbf{a}_k^T\mathbf{v}_j)\,\mathbf{v}_j,$$
$$\mathbf{v}_k = \mathbf{v}_k'/\|\mathbf{v}_k'\|.$$

Алгоритм 11.1: *Ортогонализация Грама—Шмидта*

For k = 1 to M do

$$\mathbf{v}_k' = \mathbf{a}_k - \sum_{j=1}^{k-1} (\mathbf{a}_k^T\mathbf{v}_j)\,\mathbf{v}_j$$

$$\mathbf{v}_k = \mathbf{v}_k'/\|\mathbf{v}_k\|$$

конец k-цикла.

Компоненты y_i ортогональных векторов $\mathbf{v}_k$ равны $\mathbf{v}_i^T\mathbf{b}$. Чтобы вычислить $\mathbf{x}_i$ (если нужно), используется обратная подстановка в треугольной системе, относящейся к $\mathbf{a}_k$ и $\mathbf{v}_k$ (см. задачу 18 в конце этой главы).

Можно следующим образом оценить объем затрат на ортогонализацию Грама — Шмидта. Существуют $M^2/2$ членов вида $(\mathbf{a}_k^T\mathbf{v}_j)\mathbf{v}_j$, которые необходимо вычислять. Вычисление каждого из них требует N операций (сложение+умножение) для скалярного произведения и N операций (сложение+умножение) для вычитания векторов. Затраты на вычисление x_i составляют M скалярных произведений N-векторов или MN операций, что немного по сравнению с затратами на ортогонализацию. Таким образом, общие затраты равны M^2N операций. Если N приблизительно равно M, то эти за-

траты почти на 50% превышают затраты на решение нормальных уравнений. Если N очень велико по сравнению с M, что не является необычным случаем, то затраты на ортогонализацию Грама—Шмидта приблизительно в два раза превосходят затраты на использование нормальных уравнений.

Существуют три преимущества в применении этого метода по сравнению с использованием нормальных уравнений:

1. В этом процессе число обусловленности задачи не возводится в квадрат. Это помогает в определенных (относительно немногочисленных) случаях.

2. Когда вместо ответа получается чепуха, вы знаете где и по какой причине она возникает. Предположим, что все $\mathbf{a}_i$ по порядку близки к 1 (если это не так, то их следовало бы сделать такими). Тогда, если один из $\mathbf{v}'_k$ имеет порядок ошибок округления, то вся информация о нем оказывается утерянной. Можно проверить, когда это происходит, и сообщить пользователю, что первые k векторов линейно зависимы в пределах ошибок округлений.

3. Можно ввести другой допуск (оценку ошибки наблюдений или модель неопределенности) и просто перескочить через векторы $\mathbf{v}'_k$, длина которых меньше этого допуска. Вы можете сообщить пользователю, что эти векторы (столбцы матрицы A) опущены в модели (матрицы A), поскольку они линейно зависимы с ранее обработанными векторами.

Существует современная версия метода Грама—Шмидта, называемая *модифицированной ортогонализацией Грама—Шмидта* (Rice, 1966), которая представляет собой просто перестановку порядка выполнения вычислений. В этом смысле такая модификация является аналогом модификации Краута метода исключения Гаусса. Алгоритм модификации имеет вид

Алгоритм 11.2: *Модифицированная ортогонализация Грама—Шмидта*

For $k = 1$ to M do

$\mathbf{a}_k = \mathbf{a}_k / \| \mathbf{a}_k \|$

$\mathbf{a}_j = \mathbf{a}_j - (\mathbf{a}_j^T \mathbf{a}_k)\, \mathbf{a}_k$ для $j = k+1, k+2, \ldots, M$

конец k-цикла.

Преимущества этого метода перед классическим методом Грама—Шмидта:

1. Он требует меньше оперативной памяти. Заметим, что формулы не используют других имен для новых векторов $\mathbf{v}'_k$ — они вычисляются и помещаются в то же место памяти, которое занимают исходные векторы.

2. Его легче запрограммировать.

3. Метод численно устойчив; это означает, что ответы настолько точны, насколько этого можно ожидать от обычных вычислительных методов (Bjorck, 1967).

4. Можно использовать стратегию «выбора ведущего элемента», а именно, выбрать наибольший из оставшихся векторов в качестве следующего обрабатываемого вектора. Эта стратегия выгодна в небольшом числе случаев, и могут быть построены примеры, когда такой выбор ведущего вектора так же полезен, как и выбор ведущего элемента в методе Гаусса.

11.Г. ПОДХОД С ИСПОЛЬЗОВАНИЕМ ОРТОГОНАЛЬНОГО РАЗЛОЖЕНИЯ МАТРИЦЫ

Как уже было показано, если можно найти такие матрицы M и U (нижняя и верхняя треугольные), что $MA=U$, то решение системы $A\mathbf{x}=\mathbf{b}$ не вызывает затруднений. Существует другое привлекательное разложение матрицы A, а именно, можно найти ортогональную матрицу Q и верхнюю треугольную матрицу R, такие, что

$$QA=R.$$

Тогда, чтобы решить систему $A\mathbf{x}=\mathbf{b}$, умножим обе части этого разложения на матрицу Q и получим

$$QA\mathbf{x}=Q\mathbf{b}=R\mathbf{x},$$

откуда вектор $\mathbf{x}$ может быть получен обратной подстановкой из вектора $Q\mathbf{b}$. Заметим, что такое свойство матрицы Q, как $Q^T=Q^{-1}$, не играет существенной роли для процесса решения, хотя в качестве разложения матрицы A мы имеем $A=Q^TR$.

Привлекательность ортогонального разложения объясняется тем обстоятельством, что умножение матрицы A или вектора $\mathbf{b}$ на матрицу Q не увеличивает никаких ошибок округления или неопределенностей в задании A и $\mathbf{b}$. Этот процесс имеет число обусловленности, равное 1 — наилучшее, на которое можно рассчитывать. Чтобы убедиться в этом, заметим, что $||Q\mathbf{x}||=||\mathbf{x}||$ для любого вектора $\mathbf{x}$, поскольку $Q^TQ=$:

$$||Q\mathbf{x}||^2=(Q\mathbf{x})^TQ\mathbf{x}=\mathbf{x}^TQ^TQ\mathbf{x}=\mathbf{x}^T\mathbf{x}=||\mathbf{x}||^2.$$

Вспомним рассуждения по поводу выбора ведущего элемента в методе исключения Гаусса как средства, противодействующего увеличению ошибок; такой выбор не нужен для ортогонального разложения.

Существуют две различные схемы ортогонального разложения: плоские вращения и элементарные отражения (преобразования Хаусхолдера). Остановимся кратко на этих схемах.

11.Г.1. Ортогональное разложение при помощи плоских вращений

Можно вращать координатную систему в любой плоскости, например, так, как может быть повернуто обычное трехмерное евклидово пространство. Как и в обычном трехмерном пространстве, объекты, находящиеся на оси вращения, остаются без изменения. Геометрически ясно, что такое преобразование не меняет размеры векторов. Наша цель заключается в том, чтобы найти такие вращения, которые обращают в нуль определенные элементы матрицы A, с тем чтобы получить верхнюю треугольную матрицу R.

Элементарное плоское вращение (преобразование Гивенса) представляется матрицей:

$$R_{ij} = \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & & & & \\ & & & & c & \cdots & s & & \\ & & & & \vdots & 1 & \vdots & & \\ & & & & \vdots & & 1 & & \\ & & & & -s & \cdots & c & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \leftarrow \text{строка } i \\ \\ \\ \leftarrow \text{строка } j \\ \\ \\ \end{matrix}$$

где $c^2+s^2=1$ и $c=\cos(\theta)$, $s=\sin(\theta)$ (θ — угол вращения координатной плоскости (i, j)). Непосредственной проверкой можно убедиться, что матрица $R_{ij}A$ имеет только четыре элемента, отличные от элементов матрицы A, а именно элементы с индексами (i, i), (i, j), (j, i) и (j, j).

Рассмотрим детально матрицу $R_{ij}A$ и выберем значения c и s так, чтобы обратить в нуль элементы с индексами (j, 1). Имеем

$$\text{элемент (j, 1) матрицы } R_{ij}A = -sa_{11} + ca_{j1}.$$

Мы можем подобрать угол θ так, чтобы этот элемент равнялся нулю:

$$\frac{s}{c} = \operatorname{tg}\theta = \frac{a_{j1}}{a_{11}}.$$

Таким образом, можно применить матрицы R_{1j} к матрице A для $j=2, 3, \ldots, N$ со значениями θ, обращающими каждый раз элементы a_{j1} в нули В результате матрица A примет вид, аналогичный тому, который она имеет после применения метода исключения Гаусса к первому столбцу.

Существуют различные способы явного вычисления c и s (в действительности нет необходимости знать значение угла θ как такового). Наиболее очевидный способ

$$r = \pm\sqrt{a_{11}^2 + a_{j1}^2},$$
$$c = a_{11}/r, \quad s = a_{j1}/r,$$

который требует четырех операций умножения и деления, двух операций сложения и одной операции извлечения квадратного корня. Это количество операций существенно больше числа операций на один шаг исключения Гаусса. Хитроумная, но сложная для описания схема, была найдена Джентлменом (Gentleman, 1973), которая требует только двух операций умножения и деления, двух сложений и не требует извлечения квадратного корня. Тем самым последняя схема требует всего лишь в два раза больше операций по сравнению с исключением Гаусса. Эта схема реализована в пакете BLAS (см. приложение).

11.Г.2. Ортогональное разложение с использованием элементарных отражений

Рассмотрим отражение пространства относительно некоторой плоскости, как это показано на рис. 11.2. Идея состоит в том, чтобы выбрать такую плоскость, относительно которой вектор $\mathbf{a}=(a_1, a_2, \ldots, a_N)^T$ отражался бы на первую координатную ось, т. е. $\mathrm{Ha}=(-\sigma, 0, 0, \ldots, 0)^T$, где H обозначает операцию отражения. Это линейное преобразование применяется затем к вектору $\mathbf{a}_1$, являющемуся первым столбцом матрицы A.

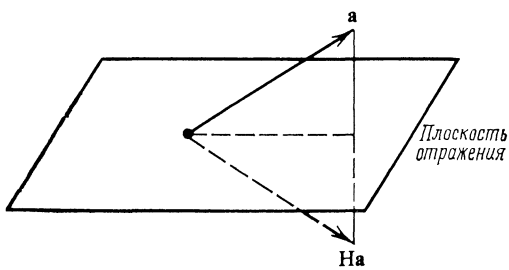


Рис. 11.2. Отражение вектора относительно плоскости

Матрица H, которая представляет это элементарное отражение (*преобразование Хаусхолдера*), легко находится следующим образом Пусть $\mathbf{e}_1=(1, 0, 0, \ldots, 0)^T$ суть первый координатный вектор, $\sigma=\pm 1$. Положим

$$\mathbf{u}=\mathbf{a}_1+\sigma\mathbf{e}_1,$$
$$c={}^1/_2\ \|\mathbf{u}\|^2,$$
$$H=I-\frac{\mathbf{u}\mathbf{u}^T}{c}.$$

Заметим, что $\mathbf{u}\mathbf{u}^T$ является полной матрицей, тогда как произведение $\mathbf{u}^T\mathbf{u}$ представляет собой одно число. Предоставим читателю убедиться в том, что H — ортогональная матрица ($H^TH=I$) и $\mathrm{Ha}_1=-\sigma\mathbf{e}_1$.

Таким образом, матрица HA имеет своим первым столбцом столбец того же вида, что и матрица A после применения исключения Гаусса. Можно применить матрицу H к другим столбцам матрицы A, после чего, как и в исключении Гаусса, применяется та же самая схема для оставшейся матрицы порядка (N—1)×(N—1). Заметим, что матрица H никогда не формируется в явном виде; результирующая матрица HA вычисляется следующим образом:

$$\mathbf{u} = \mathbf{a}_1 + \sigma \mathbf{e}_1,$$
$$c = \frac{1}{2}\|\mathbf{u}\|^2,$$
$$\mathbf{a}_1 = -\sigma \mathbf{e}_1,$$
For $j = 2, 3, \ldots, N$ do
$$d_j = \mathbf{u}^T \mathbf{a}_j / c$$
$$\mathbf{a}_j = \mathbf{a}_j - d_j \mathbf{u}$$
конец j-цикла.

Тщательный подсчет числа операций показывает, что эти вычисления также требуют в два раза больше арифметических операций по сравнению с соответствующим этапом исключения Гаусса. Тонкая реализация преобразований Хаусхолдера включена в пакет BLAS и в программы Лоусона и Хэнсона решения задач наименьших квадратов (см. приложение).

Оценка затрат каждого из этих методов дает около $M^2N - M^3/3$ операций (операция=одно сложение+одно умножение). Если N приблизительно равно M, то эта оценка приблизительно равна оценке затрат на использование нормальных уравнений. Если N велико по сравнению с M, то ортогональное разложение требует приблизительно тех же затрат, что и ортогонализация Грама—Шмидта, и в два раза больше затрат по сравнению с процессом, использующим нормальные уравнения.

Преимущества последнего метода:

1. Он численно устойчив; т. е. ответы настолько точны, насколько этого было бы естественно ожидать от численного метода.

2. Можно провести *анализ сингулярных значений* (эта тема в данной книге не обсуждается) для матрицы R, чтобы получить детальную информацию относительно сущности линейных зависимостей, которые могут существовать в исходной задаче.

11.Д. КНИГА ЛОУСОНА И ХЭНСОНА: РЕШЕНИЕ ЗАДАЧ НАИМЕНЬШИХ КВАДРАТОВ

Пока мы лишь вкратце ознакомились с существом метода наименьших квадратов. Существует много книг, излагающих теорию наименьших квадратов в деталях, однако есть одна книга, целиком

посвященная теории и практике решения задач наименьших квадратов: книга Лоусона и Хэнсона (Lawson, Hanson, 1974). Этот труд основательно покрывает все вопросы, которые возникают при практическом решении задач наименьших квадратов. В нем рассмотрены такие практические вопросы, как применение различной терминологии для одной и той же теории в численных расчетах, статистике и электротехнике; такие трудные задачи, как управление очень большими наборами данных (скажем, N=10 000 000, M=20) на реальных вычислительных системах; такие сложные теоретические вопросы, как анализ ошибок для методов разложений матриц.

Авторы показывают, что на каждый «хитрый прием» решения задачи (такой, как применение многочисленных наборов данных для одной и той же модели) с помощью одного из рассмотренных выше подходов всегда существует эквивалентный прием для двух других подходов. Они приходят к заключению, что подход с использованием ортогонального разложения на основе элементарных отражений оказывается в большинстве случаев лучшим, и анализируют его применение на большом числе примеров, включая нелинейную задачу наименьших квадратов. Всем, кто проводит серьезные вычисления с использованием метода наименьших квадратов, следует ознакомиться с этой книгой. Программы, представленные в конце книги Лоусона и Хэнсона, рассматриваются в приложении.

Задачи гл. 11

1. Дать определение *невязки* решения системы $\mathbf{Ax}=\mathbf{b}$ в смысле наименьших квадратов.

2. Пусть $\mathbf{b}=(0.01,\ 0{,}1)^{\mathrm{T}}$, $\bar{\mathbf{b}}=(0.0101,\ 0,\ 1)^{\mathrm{T}}$ и

$$A=\begin{pmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{pmatrix}.$$

(а) Вычислить решения $\mathbf{x}$ и $\bar{\mathbf{x}}$ систем $A\bar{x}=\mathbf{b}$ и $\mathbf{Ax}=\bar{\mathbf{b}}$ в смысле наименьших квадратов

(б) Какова относительная разность между $\mathbf{b}$ и $\bar{\mathbf{b}}$? Между $\mathbf{x}$ и $\bar{\mathbf{x}}$? Объяснить, почему эти разности могут так отличаться друг от друга.

(в) Дать пример векторов $\mathbf{b}$ и $\bar{\mathbf{b}}$, для которых относительная разность между векторами $\mathbf{x}$ и $\bar{\mathbf{x}}$ в 10^5 раз превосходит относительную разность между $\mathbf{b}$ и $\bar{\mathbf{b}}$.

3. Рассмотреть

$$A=\begin{pmatrix}1.000 & 1.050\\ 1.000 & 1.000\\ 1.000 & 1.000\end{pmatrix},\qquad A+E=\begin{pmatrix}1.000 & 1.051\\ 1.000 & 1.001\\ 1.000 & 1.000\end{pmatrix},$$

$$\mathbf{b}=(2\ 050,\ 2.000,\ 2\ 000)\ ,\quad \bar{\mathbf{b}}=(2.050,\ 1.500,\ 2.500)^{\mathrm{T}}.$$

(а) Показать, что $(1.000,\ 1.000)^{\mathrm{T}}$ суть решение обеих систем $\mathbf{Ax}=\mathbf{b}$ и $\mathbf{Ax}=\bar{\mathbf{b}}$ в смысле наименьших квадратов.

(б) Показать, что решения в смысле наименьших квадратов $\mathbf{x}_E$ и $\bar{\mathbf{x}}_E$ систем $(A+E)\mathbf{x}=\mathbf{b}$ и $(A+E)\mathbf{x}=\bar{\mathbf{b}}$ таковы: $\mathbf{x}_E=(1.0097, 0.9898)^T$ и $\bar{\mathbf{x}}_E=(1.3088, 0.6958)^T$.

(в) Вычислить относительные разности между A и A+E, между $\mathbf{x}$ и $\mathbf{x}_E$, между $\bar{\mathbf{x}}$ и $\bar{\mathbf{x}}_E$.

(г) Технически сложно объяснить, «почему» такие большие отличия в этих разностях могут иметь место. Однако какой вывод можно сделать из этого примера?

4. Рассмотреть

$$A=\begin{pmatrix}1.000 & 1.000\\ 1.000 & 1.040\\ 1.000 & 1.000\end{pmatrix}, \qquad A+E=\begin{pmatrix}1.000 & 1.001\\ 1.000 & 1.041\\ 1.000 & 1.000\end{pmatrix},$$

$$\mathbf{b}=(2.000, 2.040, 2.000)^T, \quad \bar{\mathbf{b}}=(1.500, 2.040, 2.500)^T.$$

(а) Показать, что $(1.000, 1.000)^T$ суть решение в смысле наименьших квадратов обеих систем $A\mathbf{x}=\mathbf{b}$ и $A\mathbf{x}=\bar{\mathbf{b}}$.

(б) Вычислить методом наименьших квадратов решения $\mathbf{x}_E$ и $\bar{\mathbf{x}}_E$ систем $(A+E)\mathbf{x}_E=\mathbf{b}$ и $(A+E)\bar{\mathbf{x}}_E=\bar{\mathbf{b}}$.

(в) Вычислить относительные разности между A и A+E, между $\mathbf{x}$ и $\mathbf{x}_E$, между $\bar{\mathbf{x}}$ и $\bar{\mathbf{x}}_E$. Сравнить по величине эти разности и провести рассуждения относительно чувствительности решения в смысле наименьших квадратов к возмущениям в матрице A.

5. При ортогонализации Грама—Шмидта векторы нормируются на каждом шаге. Описать модификацию процесса Грама—Шмидта, для которого векторы не нормируются до конца алгоритма. Объяснить, почему эта модификация менее эффективна.

6. Дать список параметров библиотечной подпрограммы хорошей организации на языке Фортран решения линейной системы $A\mathbf{x}=\mathbf{b}$ методом наименьших квадратов. Явно сформулировать алгоритмические и программистские цели ее организации, а также отметить, какие из них наиболее важны. Определить все переменные, которые присутствуют в списке параметров. Описать пример приложения, для которого предлагаемая организация программы имеет слабые стороны по сравнению с разумной альтернативой (кратко описать эту альтернативу).

7. Провести детальный подсчет числа операций для ортогонализации Грама—Шмидта системы m векторов длины n. Какие шаги этого процесса соответствуют обратной подстановке в методе исключения Гаусса?

8. Показать, как ортогонализация Грама—Шмидта может быть применена для решения системы $A\mathbf{x}=\mathbf{b}$ посредством ортогонализации столбцов матрицы A и последующего использования этого метода для ортогональной матрицы.

9. В этой книге мы изучили следующие четыре типа задач линейной алгебры: (1) решение системы $A\mathbf{x}=\mathbf{b}$; (2) вычисление A^{-1}; (3) решение $A\mathbf{x}=\mathbf{b}$ в смысле наименьших квадратов; (4) вычисление det (A). Провести сравнительный анализ следующих вопросов, касающихся этих типов задач:

(а) Сложность теории матриц, лежащей в основе вычислений.

(б) Реальный объем вычислений, требуемых для решения этих задач (т. е. количество машинного времени для матриц сравнимых размеров).

(в) Предположительный объем предварительного анализа, который следует сделать, чтобы составить библиотечную подпрограмму решения каждой из этих задач для широкого класса матриц A.

10. Составить подпрограмму на Фортране, выполняющую ортогонализацию Грама—Шмидта, и применить ее к решению задач наименьших квадратов для следующих конкретных примеров:

(а) $a_i=\left(\frac{1}{i+1}, \frac{1}{i+2}, \ldots, \frac{1}{i+n}\right)^T$ для $i=1, 2, \ldots, 10$ и $n=20$,
$b_j=1$ для всех j;

(б) $a_i=(\sin(i/n), \sin(2i/n), \sin(3i/n), \ldots, \sin(i))^T$ для $i=1, 2, \ldots, n$ и $n=10$,
$b_1=1$ и $b_j=0$ для $j>1$;

(в) a_i=случайный вектор длины n для $i=1, 2, \ldots, N$,
b_i=случайное число.

11. Повторить решение задачи 10 для модифицированной ортогонализации Грама—Шмидта.

12. Сравнить размеры и сложность составления подпрограмм на Фортране для задач 10 и 11. Сравнить вычисленные значения для каждой из трех задач наименьших квадратов.

13. Модифицировать подпрограмму для задачи 10 (или 11), включив в нее проверку на линейную зависимость векторов. Сделать допуск (оценивающий степень неопределенности модели) входной переменной. Если этот допуск равен нулю, то подпрограмма должна переприсвоить ему значение величины, соответствующей уровню ошибок округления используемой вычислительной машины и умноженной на небольшую константу. Какое влияние имеет этот допуск на задачу 10.а? Провести несколько прогонов подпрограммы с различными допусками и объяснить полученные результаты.

14. Описать алгоритм решения задачи наименьших квадратов, использующего элементарные вращения. Уровень детализации описания должен быть таким же, как и для алгоритма 5.1.

15. Реализовать алгоритм решения задачи 14 в виде подпрограммы на Фортране и применить ее к решению задачи 10.

16. Повторить решение задачи 14 для элементарных отражений.

17. Реализовать алгоритм решения задачи 16 в виде подпрограммы на Фортране и применить ее к решению задачи 10.

18. Пусть векторы a_k ортогонализированы при помощи алгоритма 11.1. Дать алгоритм вычисления коэффициентом x_i при помощи величин b^Tv_k. *Указание*: запомнить и использовать информацию, генерируемую алгоритмом 11.1.

Глава 12

УЧЕБНЫЕ ПРОЕКТЫ

Настоящая глава содержит учебные вычислительные проекты, обладающие следующими особенностями:

1. Проекты включают в себя анализ некоторых математических задач и/или разработку алгоритмов.
2. Проекты требуют значительных усилий при программировании и использовании вычислительных машин.
3. Проекты требуют сбора и анализа экспериментальных данных, полученных в результате вычислений.

Каждый проект предназначен для группы из трех-четырех студентов и рассчитан на 4—6 недель. Некоторые темы в определенной степени «открыты», и студенты могут хорошо поработать, погружаясь все глубже в исследуемые проблемы. Формулировка заданий на выполнение такого рода проектов должна предупреждать студентов о такой возможности и точно устанавливать то, что от них ожидается получение результатов в сроки, соразмерные с периодом времени, отведенного для выполнения этих заданий. Конечно, проекты могут быть выполнены одним или двумя студентами в течение пропорционально более длительного периода времени.

Учебные проекты полезны не только для изучения матричного исчисления и его математического обеспечения, но так же как опыт в реализации коллективных программных проектов и как практика в проведении научного анализа. Для последующего практического применения каждое задание должно включать в себя подготовку итогового отчета, который оценивается как научный документ. При оценке отчета следует учитывать следующее:

Анализ: правильность математического анализа задачи; правильность разработанного алгоритма.

Математическое обеспечение: правильность; стиль программирования и документация; эффективность.

Организация вычислений: соответствие экспериментам; надлежащий набор данных, точные измерения, должные выводы и должный анализ данных.

Отчет: стиль языка изложения и научного представления материала; логичность и аргументированность сделанных выводов.

ТЕМА 1: ИЗУЧЕНИЕ МЕТОДА ИСКЛЮЧЕНИЯ ГАУССА

Составить и отладить программу, применяющую исключение Гаусса с частичным выбором ведущего элемента для решения системы $Ax=b$ и вычисления A^{-1}. Спроектировать подпрограмму так, чтобы выбор режима ее работы управлялся простым переключателем. Для этой подпрограммы выполнить следующие задания:

А. Провести подсчет числа операций, выполняемых подпрограммой.

Б. Представить требования на размер оперативной памяти для размещения переменных через размеры параметров задачи. В эти требования не включать длину самой подпрограммы.

В. Дать требования на размер оперативной памяти для размещения подпрограммы.

Г. Сравнить скорость исполнения двух других подобных подпрограмм. Выбрать подпрограммы из местной библиотеки, из библиотеки IMSL (например, LEQT1F и LINV1F) или из пакета LINPACK (например, SGECO, SGESL и SGEDI). Спроектировать и провести эксперимент, который охватывает диапазон порядков матриц в пределах от 2 до 30. Представить результаты вместе с составным характеристическим профилем.

Д. Оценить точность вычислений подпрограмм для хорошо обусловленных матриц, имеющих порядок в пределах от 2 до 30. Использовать случайные матрицы, выбрать точное решение $\mathbf{x}^*$ и образовать правые части на основе $\mathbf{x}^*$. Провести анализ точности вычисленного решения $\mathbf{x}$, как функцию от порядка матрицы.

Е. Повторить задание Д для плохо обусловленных матриц (см. гл. 8 для примеров), имеющих порядок в пределах от 4 до 12 или 15. Исследовать использованные ведущие элементы для выявления особенностей в поведении. Сравнить их с ведущими элементами для хорошо обусловленных систем. Исследовать числа, получаемые при обратной подстановке:

$$x_j = \frac{b_j - A_j^T \mathbf{s}}{a_{jj}},$$

где вектор $\mathbf{s}$ представляет собой часть вектора решения $(0, \ldots, 0, x_{i+1}, \ldots, x_n)^T$, а A_j суть строка j матрицы A. Показать некоторые из полученных результатов при помощи характеристических профилей, включающих в себя числа обусловленности.

Ж. Повторить задание Д для какой-либо другой сопоставимой библиотечной подпрограммы и затем сравнить рабочие характеристики подпрограмм.

З. Повторить задание Е для какой-либо другой сопоставимой библиотечной подпрограммы и затем сравнить рабочие характеристики подпрограмм.

ТЕМА 2: ВЛИЯНИЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ НА ПРОГРАММЫ РЕШЕНИЯ ЛИНЕЙНЫХ УРАВНЕНИЙ

Эта тема включает в себя семь подпрограмм, из числа которых первые четыре должны быть составлены.

1. Подпрограмма на Фортране для
   а. Решения системы $Ax=b$ методом исключения Гаусса с частичным выбором ведущего элемента при возможности использования нескольких правых частей.
   б. Вычисления A^{-1}.
2. Подпрограмма на Паскале того же назначения.
3. Подпрограмма на Бейсике, Алголе или ПЛ/1 (на выбор) того же назначения.
4. Подпрограмма на Фортране с удвоенной точностью того же назначения.
5. Аналогичная подпрограмма из библиотеки вычислительного центра.
6. Аналогичная подпрограмма из какой-либо стандартной библиотеки (например, LEQT1F из библиотеки IMSL).
7. Аналогичная подпрограмма из пакета LINPACK (например, SGEFA и SGESL).

Для того чтобы сделать первые четыре подпрограммы как можно более идентичными, алгоритм следует представить в некоторой независимой от языков программирования форме (например, в виде блок-схемы) и затем составить каждую подпрограмму на основе этой спецификации. Эти четыре подпрограммы должны быть хорошо структурированы, обладать высоким качеством и хорошей документацией.

Выполнить следующие задания:

А. Оценить время, требуемое на спецификацию алгоритма, его кодирование и отладку каждой подпрограммы. Дать длину каждой подпрограммы в операторах (как с комментариями, так и без них).

Б. Объяснить замысел выбранного интерфейса между подпрограммами и пользователем.

В. Измерить время компиляции каждой подпрограммы (если возможно).

Г. Организовать и провести эксперимент для оценки требуемой памяти и скорости исполнения всех семи подпрограмм. Использовать полные матрицы с порядками в пределах от 3 до 40. Представить результаты в виде характеристических профилей. Объяснить наблюдаемые различия.

ТЕМА 3: ИЗУЧЕНИЕ ПОДПРОГРАММ РЕШЕНИЯ СИСТЕМ С ЛЕНТОЧНЫМИ МАТРИЦАМИ

Эта тема включает в себя четыре подпрограммы, две из которых должны быть составлены.

1. Подпрограмма решения системы Ax=b, где A — ленточная матрица порядка n с шириной ленты k. Применить метод исключения Гаусса без выбора ведущего элемента и экономную схему размещения в памяти ленточной матрицы.
2. Аналогичная подпрограмма, которая использует частичный выбор ведущего элемента.
3. Две аналогичные библиотечные подпрограммы, например SGBCO из пакета LINPACK или LEQT1B из библиотеки IMSL или какие-либо другие подпрограммы из местной библиотеки вычислительного центра.

Выполнить следующие задания:

А. Показать, что подпрограмма 2 с частичным выбором ведущего элемента требует в два раза больше памяти, чем подпрограмма 1.

Б. Объяснить замысел интерфейса между подпрограммами 1 и 2 и пользователем, включая выбранную схему размещения в памяти ленточной матрицы.

В. Качественно сравнить трудность программирования метода исключения Гаусса для полных матриц с трудностью его программирования в условиях подпрограммы 1: сравнить полученные результаты с трудностью составления подпрограммы 2.

Г. Сравнить скорость выполнения этих четырех подпрограмм на наборе задач, содержащих до 8000 элементов в лентах матриц. Показать их рабочие характеристики через размеры параметров n и k. Согласуются ли наблюдаемые результаты с тем поведением подпрограмм, которого следует ожидать исходя из подсчета числа операций?

Д. Повторить задание Г для исследования требований на размеры памяти вместо скорости.

Е. Выбрать некоторые плохо обусловленные ленточные системы и провести эксперимент по вашему плану, чтобы показать значимость выбора ведущего элемента при решении этих систем. Выявить также невырожденную ленточную матрицу, для которой подпрограмма 1 терпит неудачу.

ТЕМА 4: ВЛИЯНИЕ ЯЗЫКОВ И СТИЛЯ ПРОГРАММИРОВАНИЯ

Цель этой темы состоит в изучении вопросов точности, эффективности и самого процесса реализации математического обеспечения как функции следующих переменных: компилятор, язык, точность вы-

числений и уровень модульности. В ней определяются три математические задачи, которые должны быть реализованы в виде программ следующими способами:

1. На Фортране с одинарной точностью. Должны быть использованы три различных компилятора.
2. На Фортране с удвоенной точностью с использованием одного компилятора.
3. На Алголе.
4. На Паскале.
5. На Бейсике.
6. С использованием небольшой модульности (для одного из языков по вашему выбору) с максимально одним уровнем подпрограмм (не считая функций)
7. Со значительной модульностью на Фортране и Паскале или Алголе. Некоторые указания относительно модульности приводятся в постановке каждой из задач.

По крайней мере одна из программных реализаций должна иметь хороший стиль программирования и хорошую документацию.

Необходимо организовать эксперимент для оценки влияний отмеченных выше переменных на эти три программы в шести реализациях. Должны быть предприняты меры предосторожности, чтобы программы выполняли одни и те же вычисления в пределах ограничений, продиктованных требованиями к конкретной реализации. Эксперимент должен включать в себя выполнение следующих заданий:

А. Оценить легкость программирования и отладки каждой реализации по шкале оценок от 0 до 10, где уровень 0 сравним с программированием на языке ассемблера, а 10 означает крайне быстро и просто.

Б. Измерить время, потраченное на программирование и отладку, длины программ, время их компиляции (если есть такая возможность) и время их выполнения. Провести анализ этих данных как функцию четырех переменных.

В. Сравнить точности вычисленных результатов.

Г. Рассмотреть две различные вычислительные машины, для первой из которых режим счета с удвоенной точностью на 10% медленнее, чем с одинарной точностью, а для второй — на 300%. Оценить эффект, который должен иметь место от использования этих двух машин, на проведенные измерения и на баланс в выборе между скоростью и точностью.

Д. Рассмотреть, как проводятся измерения времени выполнения программ и насколько можно быть уверенным в их правильности.

Е. Рассмотреть эффект выполнения программы в режиме интерпре-

тации (Бейсик) в сравнении с компиляцией программы и последующим ее выполнением.

Ж. Выбрать одну из программ и вставить в нее большое количество промежуточных выдач на печать. Сравнить время выполнения этой реализации со временем выполнения программы без выдач и рассмотреть относительную стоимость операторов ввода-вывода и арифметических операторов.

З. Представить обобщающую аттестацию сильных и слабых сторон рассматриваемых языков программирования с точки зрения реализации математического обеспечения.

Три приведенные ниже математические задачи типичны для численных расчетов, хотя они в некоторой степени искусственны, поскольку предусматривается большой объем вычислений и потенциально высокий уровень модульности для столь коротких программ. Вычисления включают в себя четыре функции:

$$f_1(x) = \sin\frac{x}{2},$$

$$f_2(x) = \frac{1}{1+66x^4},$$

$$f_3(x) = \min[\sin(5x),\ 0.75],$$

$$f_4(x) = \sin\frac{x}{2} + 10^{-6} * \left(\mathrm{RANF}(0) - \frac{1}{2}\right),$$

где RANF (0) — равномерно распределенная на интервале (0, 1) случайная переменная.

Задача 1: ортогонализация Грама — Шмидта

Для $f_1(x)$, $f_2(x)$ и $f_4(x)$ выполнить
  Для $N = 2$ до 12 с шагом 5 выполнить
    Для $M = N + 1$ до $N = 26$ с шагом 3 выполнить
      1) Образовать векторы $\mathbf{x}_j = f(j * x)$ для $x = i/M$, $i = 0, 1, \ldots, M$ и для $j = 1, 2, \ldots, N$.
      2) Ортогонализировать эти векторы методом Грама — Шмидта, чтобы получить векторы $\mathbf{v}_j$.
      3) Найти максимальное значение $\mathbf{v}_j^T\mathbf{v}_k$ для $j \neq k$.
  Конец циклов по M и N.
  Напечатать таблицу найденных максимальных скалярных произведений как функцию от N и M. Хорошо форматировать таблицу и ввести в нее заголовок.
Конец цикла по f (x).

В случае использования значительной модульности ввести по крайней мере один модуль (подпрограмму) для реализации каждой из следующих операций:

Умножение вектора на константу.

Полное вычисление каждой f (x).

Печать вектора (строки таблицы).	Нормировка вектора.
Поиск наибольшей компоненты вектора.	Вычитание вектора.
Вычисление внутреннего цикла для пары N, M.	Формирование вектора.
	Скалярное произведение.
Вычисление внешнего цикла для значения N.	Сумма в методе Грама—Шмидта.

Задача 2: исключение Гаусса

Для $f_1(x)$, $f_2(x)$ и $f_4(x)$ выполнить
  Для N = 2 до 15 с шагом 3 выполнить
    1) Образовать матрицу порядка N с элементами $a_{ij} = f(i * j/N)$.
    2) Образовать вектор истинного решения и соответствующий вектор **b** правых частей.
    3) Решить систему $A\mathbf{x} = \mathbf{b}$ методом исключения Гаусса для поиска $\bar{\mathbf{x}}$.
    4) Вычислить $\|\mathbf{x} - \bar{\mathbf{x}}\|/\|\mathbf{x}\|$ и $\|\mathbf{r}\|$, где **r** — невязка для $\bar{\mathbf{x}}$.
  Конец цикла по N.
Напечатать таблицу значений $\|\mathbf{x} - \bar{\mathbf{x}}\|/\|\mathbf{x}\|$ и $\|\mathbf{r}\|$ для каждой функции f(x) и каждого значения N. Хорошо форматировать таблицу и ввести в нее заголовок.
Конец цикла по f(x).

В случае использования значительной модульности ввести по крайней мере один модуль (подпрограмму) для реализации каждой из следующих операций:

Полное вычисление каждой f(x).	Обработка правой части.
Внутренний цикл для одного значения N.	Вычисление a_{ij}.
Формирование вектора истинного решения.	Вычисление **b**.
Вычисление одной компоненты вектора **b**.	Формирование строки матрицы A.
Вычисление нормы вектора.	Формирование A.
Обработка одного столбца в методе исключения Гаусса.	Вычисление **r**.
Вычисление множителей.	Исключение Гаусса.
Обработка одной строки в методе исключения Гаусса.	Выбор ведущего элемента.
	Обратная подстановка.

Задача 3: интегрирование методом Ромберга на отрезке (—2, 2)

Для $f_1(x)$, $f_2(x)$, $f_3(x)$ и $f_4(x)$ выполнить
  Для H = 1.0, 0.5, 0.25, ..., 0.015625 выполнить

1) Образовать массив $F(N,M) = f(-2 + (N-1)\,2^{1-M})$ для $M = 1$ до 7 и для $N = 1$ до 2^{M+1}.
2) Для NEXTRAP = 1 до 7 выполнить

   Образовать массив TRAP(M,NEXTRAP) для M = 1, ..., 7, где TRAP(M,1) представляет собой значение $\int_{-2}^{2} f(x)\,dx$, вычисленное по правилу трапеций на 2^{M-1} интервалах, а TRAP(M,K) — экстраполяцию по Ромбергу столбца K — 1.

   Конец цикла по NEXTRAP.

Конец цикла по H.

Напечатать таблицу значений интеграла и их ошибки.

Для $f_4(x)$ предположить, что $\int_{-2}^{2} (RANF(0) - \frac{1}{2})\,dx = 0$.

Конец цикла по f(x)

В случае использования значительной модульности ввести по крайней мере один модуль (подпрограмму) для реализации каждой из следующих операций:

Единичная экстраполяция.	Формирование одного столбца массива F для фиксированного H.
Формирование столбца массива TRAP.	Формирование массива F.
Вычисление по правилу трапеций.	Формирование массива TRAP.
Вычисление одной ошибки.	Печать таблицы для одной f(x).
Все вычисления для одной f(x).	Печать строки таблицы.
Вычисления для одного значения H.	Вычисление одного элемента в массиве F.

ТЕМА 5: РОБАСТНОСТЬ ПРОГРАММЫ РЕШЕНИЯ ЛИНЕЙНЫХ УРАВНЕНИЙ

Отобрать из библиотеки столько подпрограмм решения линейных уравнений, сколько это будет целесообразно (по крайней мере три), и неправильно применить их различными способами. Затем представить ваши оценки, насколько робастна каждая подпрограмма и сколько дополнительных усилий потребуется, чтобы сделать эту подпрограмму достаточно робастной. Типичные способы неправильного применения подпрограмм:

1. Аргументы неверных типов, например: целый тип для аргументов вещественного, комплексного или логического типов или имен

функций; массивы для аргументов, не являющихся массивами; целые массивы.

2. Недопустимые значения аргументов, например: порядки матриц 0 или —1; отрицательное число правых частей; неверные значения переключателей для подпрограмм, обладающих выбором режимов вычислений.
3. Несовместимые размерности, например: 20 уравнений для матрицы 10-го порядка; различные значения реальных и указанных размерностей.
4. Экстремальные значения аргументов, например: порядок матрицы 1; порядок матрицы 1 000 000; 1 000 000 правых частей; элементы матрицы близки к машинным нулям или к наибольшему представимому на вычислительной машине числу.
5. Нерешаемые задачи, например: матрица равна нулевой или обладает менее очевидной вырожденностью; система

$$x - y = \sigma,$$
$$x - 1.0001y = \sigma + 1,$$

где значение σ близко к наибольшему представимому на вычислительной машине числу.

ТЕМА 6: ОЦЕНКА БИБЛИОТЕЧНЫХ ПОДПРОГРАММ РЕШЕНИЯ ЛИНЕЙНЫХ УРАВНЕНИЙ

Отобрать для оценки разумное число (по крайней мере три) библиотечных подпрограмм решения $A\mathbf{x}=\mathbf{b}$. Подпрограммы должны представлять различные библиотеки. Они должны оцениваться на наличие следующих характеристик:

1. Документация для пользователей. Она должна быть ясно написана, без двусмысленностей и с хорошо определенными переменными. Содержат ли инструкции примеры использования, вызывает ли правильные ассоциации выбор слов и имен?
2. Соответствует ли документация нуждам сопровождения подпрограмм. Если что-то в программе необходимо изменить, обеспечивают ли внутренние комментарии хорошую основу для понимания алгоритма?
3. Представьте себе какое-либо применение подпрограммы. Хорошо ли выбраны переменные и их имена? Существуют ли переменные, назначение которых трудно понять?
4. Скорость выполнения. Составить характеристический профиль для каждой подпрограммы, показывающий зависимость времени выполнения от порядка матриц.
5. Использование оперативной памяти. Составить характеристический профиль для каждой подпрограммы, показывающий зависимость всей используемой памяти от порядка матриц.

6. Точность для плохо обусловленных систем. Выбрать два семейства плохо обусловленных матриц (таких, как матрицы Гильберта) и составить характеристический профиль для каждой подпрограммы и каждого семейства, показывающий зависимость точности от порядка матриц.

Выполнив эти оценки, провести обобщающую аттестацию каждой подпрограммы и классифицировать их в соответствии со степенью пригодности в качестве библиотечных подпрограмм. Пояснить выбор критериев, их значимость и процедуру проведения классификации. Рассмотреть недостатки этой оценки с двух точек зрения: недостатки в оценке указанных выше шести качественных характеристик и существование других соображений, не включенных в эту аттестацию.

ТЕМА 7: ПРЯМЫЕ И ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ ЛИНЕЙНЫХ СИСТЕМ, ВОЗНИКАЮЩИХ В КРАЕВЫХ ЗАДАЧАХ ДЛЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Дискретизация обыкновенных дифференциальных уравнений конечными разностями приводит к линейным уравнениям; если рассматривается краевая задача, то уравнения образуют совместную линейную систему. Три примера такого рода задач:

$$y'' = 1, \quad y(0) = 1, \quad y(2) = 2$$

с решением $y(x) = x^2/2$;

$$y'' + \frac{\pi^2}{4} y = 0, \quad y(0) = 0, \quad y(0.5) = \sqrt{2}/2$$

с решением $y(x) = \sin(\pi x/2)$;

$$y'' + \frac{1}{x} y' \left(1 - \frac{1}{x^2}\right) y = 0, \quad y(1) = J_1(1), \quad y(2) = J_1(2)$$

с решением $y(x) = J_1(x)$,

где $J_1(x)$ — функция Бесселя первого порядка.

Для одного или всех этих уравнений выполнить следующие задания:

А. Выписать систему конечно-разностных уравнений.

Б. Оценить вычислительные затраты, требуемые для вычисления аналитических решений с шестью десятичными цифрами в 100 и 1000 точках интервала. Определить и использовать разложения в ряд Тэйлора для этих вычислений.

В. Оценить до проведения любых вычислений те вычислительные затраты, которые потребуются для решения конечно-разностных уравнений в 100 и 1000 точках при помощи:

1. Исключения Гаусса.

2. Итерационного метода Якоби.
3. Итерационного метода Гаусса — Зейделя.

Г. Вычислить решения конечно-разностных уравнений при помощи каждого из трех методов из задания В. Сравнить реальные затраты с оценками задания В.

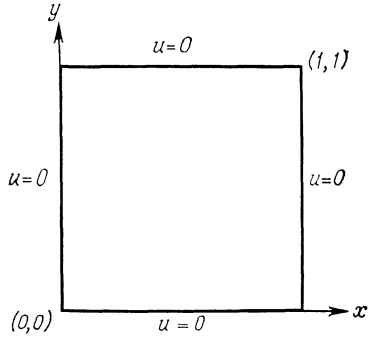
Д. Оценить применимость различных методов приближенного решения краевых задач для дифференциальных уравнений.

ТЕМА 8: РЕШЕНИЕ ЛИНЕЙНЫХ СИСТЕМ, ВОЗНИКАЮЩИХ В КРАЕВЫХ ЗАДАЧАХ ДЛЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ В ЧАСТНЫХ ПРОИЗВОДНЫХ

Техника конечных разностей может быть применена к уравнению в частных производных

$$u_{xx}+u_{yy}=6xye^{x}e^{y}\ (xy+x+y-3)$$

во многом тем же способом, что и для обыкновенных дифференциальных уравнений. Для такого подхода выполнить следующие задания:



Решение
$u=3e^{x}e^{y}(x-x^{2})(y-y^{2})$

А. Получить систему линейных уравнений при помощи дискретизации уравнения в частных производных. Объяснить, каким образом краевые условия входят в эту систему. Использовать один и тот же размер шага h для каждой из переменных x и y.

Б. Показать, каким образом применить к решению полученной системы библиотечную подпрограмму для ленточных матриц. Заметим, что выбор ведущего элемента здесь не требуется.

В. Применить эту библиотечную подпрограмму для составления небольшого характеристического профиля, показывающего зависимость ошибок от шага конечных разностей. Найти формулу,

которая аппроксимирует этот профиль. Нарисовать также характеристический профиль, показывающий зависимость времени выполнения подпрограммы от шага h. Использовать значение h в пределах от 0.25 до 0.05.

Г. Составить подпрограмму, применяющую итерационный метод Гаусса — Зейделя для решения линейной системы из задания А. Определить приемлемый способ окончания итерации для случая, когда точность решения линейной системы только ненамного выше, чем ошибка при аппроксимации решения уравнения в частных производных.

Д. Сравнить рабочие характеристики двух методов из заданий В и Г по отношению ко времени выполнения подпрограмм и использованной памяти.

ТЕМА 9: ПРЯМЫЕ И ИТЕРАЦИОННЫЕ МЕТОДЫ ДЛЯ ПЛОХО ОБУСЛОВЛЕННЫХ ЛИНЕЙНЫХ СИСТЕМ

Цель этой темы состоит в сравнении точности, достигаемой прямыми и итерационными методами для плохо обусловленных задач. Выбрать несколько матриц (по крайней мере четыре), имеющих порядок в пределах от 6 до 12, для которых метод Гаусса — Зейделя сходится и которые являются плохо обусловленными. Отобрать векторы истинных решений и сгенерировать соответствующие правые части для системы $A\mathbf{x}=\mathbf{b}$. Затем выполнить следующие задания:

А. Решить системы методом исключения Гаусса.

Б. Применить к системам итерационный метод Гаусса — Зейделя, продолжая итерации до тех пор, пока будет происходить дальнейшее улучшение точности.

В. Исследовать и обсудить поведение итераций после того, как прекращается улучшение точности.

Г. Принять истинное решение в качестве начального приближения к решению в методе Гаусса — Зейделя и обсудить последующие итерации.

Д. Сравнить точность, полученную методом исключения Гаусса и итерационным методом Гаусса — Зейделя.

ТЕМА 10: ЭЛЕМЕНТАРНЫЕ ОТРАЖЕНИЯ ДЛЯ $A\mathbf{x}=\mathbf{b}$

В гл. 11 было отмечено, что элементарные отражения могут быть применены для решения системы $A\mathbf{x}=\mathbf{b}$. Настоящая тема должна развить и оценить этот метод. Предлагаются для выполнения следующие задания:

А. Дать детальное описание алгоритма решения $A\mathbf{x}=\mathbf{b}$.

Б. Провести детальный подсчет числа операций для этого алгоритма.

В. Составить подпрограмму решения $\mathbf{Ax}=\mathbf{b}$ с использованием этого алгоритма. Обсудить замысел интерфейса с пользователем и подготовить набор инструкций по применению подпрограммы

Г. Сравнить скорость выполнения этой подпрограммы и библиотечной подпрограммы, которая использует метод исключения Гаусса с частичным выбором ведущего элемента. Подготовить характеристические профили, показывающие зависимость времени выполнения подпрограмм от порядка матриц (до 25-го порядка).

Д. Продолжить сравнения задания Г для анализа точности решения плохо обусловленных задач. Для сравнения использовать случайные матрицы.

Е. Продолжить сравнения для плохо обусловленных задач. Подготовить для каждой подпрограммы характеристические профили, показывающие зависимость точности от числа обусловленности.

Ж. Провести обобщающие сравнения этих двух подпрограмм и обосновать ваш выбор одной из них для включения в библиотеку.

ТЕМА 11: ЭЛЕМЕНТАРНЫЕ ВРАЩЕНИЯ ДЛЯ $\mathbf{Ax}=\mathbf{b}$

Повторить тему 10 для применения элементарных вращений к процессу решения $\mathbf{Ax}=\mathbf{b}$.

ТЕМА 12: ИСКЛЮЧЕНИЕ ЖОРДАНА ДЛЯ $\mathbf{Ax}=\mathbf{b}$

После того как был использован метод исключения Гаусса для приведения матрицы A к треугольной форме U, можно применить процесс исключения для приведения матрицы U к диагональной форме. Например, последняя строка, умноженная на соответствующие множители, может вычитаться из предыдущих строк, чтобы обратить в нуль элементы последнего столбца матрицы U. Затем предпоследняя строка матрицы U может быть использована, чтобы обратить в нуль элементы предпоследнего столбца, и так далее Цель этой темы состоит в развитии и оценке этого метода. Предлагаются для выполнения следующие задания:

А. Дать детальное описание алгоритма Показать, как сохранить все множители для последующего применения при обработке правой части.

Б. Завершить тему выполнением заданий от Б до Ж темы 10.

ТЕМА 13: ЭКСПЕРИМЕНТАЛЬНОЕ ОПРЕДЕЛЕНИЕ ОПТИМАЛЬНОГО МНОЖИТЕЛЯ ВЕРХНЕЙ РЕЛАКСАЦИИ SOR

Существует теория SOR-итераций, которая позволяет аналитически определять оптимальный множитель верхней релаксации (матрица

Q в гл. 10 имеет константы q на диагонали). Для многих приложений множитель q должен определяться экспериментально. Простой способ его экспериментального определения состоит в систематической оценке скорости сходимости для различных значений q и последующего выбора наилучшего q. Это и является целью темы. Предлагаются следующие задания:

А. Рассмотреть две матрицы: A_1 — трехдиагональная матрица, у которой на диагонали расположены 2, а выше и ниже диагонали расположены — 1; A_2 — пятидиагональная матрица, у которой на диагонали расположены 2, выше и ниже диагонали расположены — 1, а на последующих верхней и нижней диагоналях расположены 1/2. Построить векторы правых частей так, чтобы вектор $\mathbf{x}=(1, 1, \ldots, 1)^T$ являлся точным решением каждой из систем $A_1\mathbf{x}=\mathbf{b}_1$ и $A_2\mathbf{x}=\mathbf{b}_2$.

Б. Для каждого значения q=1, 1.1, 1.2, . . . , 1.9, 2.0 применить 100 SOR-итераций, начиная с $\mathbf{x}^{(0)}=0$. Выполнить этот процесс для матриц A_1 и A_2, имеющих порядок 10, 20 и 50. Измерить ошибку e в конце 100 итераций и положить $p=\sqrt[100]{e}$. Значение p представляет собой «среднюю» скорость сходимости итерации; ошибка уменьшалась в это число раз после каждой итерации. Заметим, что ошибка для $\mathbf{x}^{(0)}$ равна 1.

В Для каждого из этих шести случаев составить график, характеризующий p в зависимости от q и оценить оптимальное значение q. Если график слишком грубый, выполнить дополнительные расчеты, чтобы заполнить пробелы.

Г Обсудить поведение характеристических профилей и их значение для поиска оптимальных SOR-множителей.

ТЕМА 14: ЭКСПЕРИМЕНТЫ НАД ИТЕРАЦИОННЫМИ МЕТОДАМИ ГАУССА — ЗЕЙДЕЛЯ И ЯКОБИ

Эта тема посвящена исследованию различных видов доминирования одной части матрицы над другой и как это влияет на итерационные методы. Предлагаются следующие задания:

А Составить две подпрограммы: одна из них реализует итерационный метод Якоби, другая — Гаусса—Зейделя.

Б Составить подпрограмму генерации случайных матриц A порядка N в виде L+U, где L — нижняя треугольная матрица и U — верхняя треугольная матрица. Эта подпрограмма должна иметь два других параметра MODE и SIGMA. Должны удовлет-

воряться следующие соотношения:

$$\text{MODE}=1 \quad \sum_{\substack{ij\\ i}} |l_{ij}| = \text{SIGMA} * \sum_{ij} |u_{ij}|,$$

$$\text{MODE}=2 \quad \sum_{j=1}^{N} |l_{ij}| = \text{SIGMA} * \sum_{l=i+1}^{N} |u_{ij}| \quad \text{при} \quad i=1, 2, \ldots, N,$$

$$\text{MODE}=3 \quad |l_{ij}| = \text{SIGMA} * |u_{ij}| \quad \text{при} \quad i \neq j.$$

Таким образом, L «доминирует» над U тремя различными способами.

В. Спроектировать эксперимент по исследованию влияния SIGMA и MODE на сходимость этих двух методов. Отобрать пару средних значений N и диапазон значений SIGMA для каждого значения MODE.

Г. Провести эксперимент и представить относительное число сходящихся случаев для двух методов как функцию от SIGMA и MODE. Обсудить наблюдаемые соотношения.

ТЕМА 15: СРАВНЕНИЕ ОБЫЧНОГО И НАПРАВЛЕННОГО МЕТОДОВ ГАУССА — ЗЕЙДЕЛЯ

Направленный метод Гаусса — Зейделя означает, что следующая итерация делается для того уравнения, у которого невязка наибольшая. При нормальных обстоятельствах непозволительно дорого вычислять *все* невязки на каждой итерации. Однако для очень разреженных матриц регулярной структуры это может быть вполне практичным. Выполнить следующие задания:

А. Описать ситуации, когда затраты на перевычисление невязок оправданы. Представить конкретные процедуры для систем, которые возникают в темах 7 и 8.

Б. Описать, как эффективно использовать информацию о том, какое уравнение имеет наибольшую невязку.

В. Составить подпрограмму, реализующую направленный метод Гаусса — Зейделя. Применить процедуру, разработанную в задании Б для направления итераций. Корректировать невязки после каждой итерации непосредственным образом; здесь нет необходимости добиваться эффективности.

Г. Спроектировать и провести эксперимент для сравнения направленного метода Гаусса — Зейделя с обычным. Использовать матрицы, имеющие порядок в пределах от 5 до 20 и измерить «скорость сходимости», как это указано в пункте Б темы 13.

Д. Обсудить увеличение скорости сходимости, которого достигает направленный метод Гаусса — Зейделя; какой объем «дополнительных расходов» можно себе позволить на процедуру поиска «направления» и все же достигнуть большей эффективности по сравнению с обычным методом Гаусса — Зейделя.

ТЕМА 16: ИЗУЧЕНИЕ ОБЫЧНОЙ И МОДИФИЦИРОВАННОЙ ОРТОГОНАЛИЗАЦИИ ГРАМА — ШМИДТА

А. Представить детальные алгоритмы обычной и модифицированной ортогонализации Грама — Шмидта.

Б. Дать подсчет арифметических операций для каждого алгоритма через число N векторов длины M.

В. Спроектировать и составить подпрограммы, реализующие каждый алгоритм. Обосновать ваш проект и подготовить набор инструкций для пользователей для каждой из подпрограмм (они могут быть почти одинаковыми).

Г Дать общие требования на размеры оперативной памяти для каждой из подпрограмм как функцию N и M.

Д. Спроектировать и провести эксперимент для оценки скорости выполнения каждой подпрограммы. Подготовить характеристические профили, которые показывают поведение каждой подпрограммы.

Е. Сгенерировать векторы длины N=20 и 40 по формуле

$$\mathbf{x}_k = \left(\left(\frac{1}{N}\right)^k, \left(\frac{2}{N}\right)^k, \ldots, \left(\frac{N}{N}\right)^k \right)^T.$$

Применить эти векторы для проверки точности подпрограмм при помощи непосредственного вычисления скалярных произведений результирующих векторов.

Ж. Обрисовать в общих чертах, как использовать одну из этих подпрограмм для решения $A\mathbf{x}=\mathbf{b}$.

ТЕМА 17: ОЦЕНКА ЛИНЕЙНЫХ МЕТОДОВ НАИМЕНЬШИХ КВАДРАТОВ

Цель этой темы состоит в сравнении трех существенно различных методов решения линейной задачи наименьших квадратов. Выполнить следующие задания:

А. Спроектировать и составить подпрограмму решения $A\mathbf{x}=\mathbf{b}$ в смысле наименьших квадратов при помощи модифицированной ортогонализации Грама — Шмидта. Обосновать ваш проект и подготовить набор инструкций для пользователей подпрограммы.

Б. Повторить задание А на основе подхода с использованием нормальных уравнений; применить библиотечную подпрограмму решения нормальных уравнений методом исключения Гаусса.

В. Взять соответствующую подпрограмму Лоусона и Хэнсона для той же самой задачи.

Г. Спроектировать и провести эксперимент для сравнения скорости выполнения трех подпрограмм. Использовать N векторов длины M для $5 \leqslant N \leqslant 20$ и $10 \leqslant M \leqslant 50$. Подготовить характеристические профили, которые иллюстрируют поведение каждого метода;

обсудить соотношение между действительным временем выполнения и числом операций, приведенным в гл. 11.

Д. Подобным же образом сравнить точность трех подпрограмм. Использовать векторы из задания Е темы 16 в качестве столбцов матрицы А и измерить точность вычисленных результатов.

Е. Представить обобщающую аттестацию трех подпрограмм, основанную на проведенных наблюдениях. Обсудить любые недостатки, которые могут иметь оценки, и предложить (но не провести) дальнейшие эксперименты, которые помогут определить различия между подпрограммами.

ТЕМА 18: МОДИФИЦИРОВАННАЯ ОРТОГОНАЛИЗАЦИЯ ГРАМА — ШМИДТА С ВЫБОРОМ ВЕДУЩЕГО ВЕКТОРА

Рассмотрим алгоритм модифицированной ортогонализации Грама — Шмидта (MGS), измененной таким образом, чтобы следующим ортогонализуемым вектором являлся наибольший из всех оставшихся векторов. Для оценки этого варианта выполнить следующие задания:

А. Представить детальное описание этого алгоритма. Оценить дополнительные затраты на этот алгоритм по сравнению с MGS.

Б. Построить пример трех векторов длины 5, для которых MGS вырабатывает один «правильный» ортогонализированный вектор, а рассматриваемый вариант вырабатывает два «правильных» ортогонализированных вектора. Пример будет содержать два почти линейно зависимых вектора.

В. Спроектировать и составить подпрограмму, которая применяет этот вариант ортогонализации векторов. Обосновать ваш проект и подготовить набор инструкций для пользователей подпрограммы. Вставить в подпрограмму переключатель для работы в режимах без и с выбором ведущего вектора.

Г. Спроектировать и провести эксперимент для проверки эффективности выбора ведущего вектора Включить в него плохо обусловленные наборы векторов следующих типов:

1. Столбцы матрицы Гильберта.
2. Векторы из задания Е темы 16.
3. Векторы из задачи 1 темы 4 для $f_2(x)$.
4. Случайная смесь векторов задания 1 темы 4 для $f_1(x)$ и $f_2(x)$.

Д. Дать оценку значимости выбора ведущего вектора в MGS.

ТИПОВОЕ МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДЛЯ ЗАДАЧ ЛИНЕЙНОЙ АЛГЕБРЫ

А. BLAS: основные подпрограммы линейной алгебры.
Б. Подпрограммы пакета LINPACK.
В. Подпрограммы решения линейных алгебраических уравнений библиотеки IMSL.
Г. Подпрограммы решения совместных линейных уравнений библиотеки NAG.
Д. Подпрограммы для метода наименьших квадратов из книги Лоусона и Хэнсона.
Е. EISPACK — подпрограммы решения алгебраической проблемы собственных значений.
Ж. Местная библиотека вычислительного центра — пример Университета Пэрдью.

В этом приложении мы обсудим семь наборов программ матричного исчисления, и все они, кроме одного, доступны как компоненты типового математического обеспечения для широкого круга пользователей. Исключение составляет библиотека вычислительного центра Университета Пэрдью, которая представляет собой пример того, что́ можно найти в хорошо поддерживаемой библиотеке вычислительного центра.

Материал представлен весьма кратко, он не был предназначен как руководство для пользователя или справочник. Изложение подробно лишь в той степени, чтобы показать особенности, области применения, организацию и использование этих наборов программ. Даются ссылки для тех, кто хочет получить больше информации или использовать программы; бо́льшая часть материала в приложении непосредственно дана по первоисточникам для того, чтобы показать их характер.

Каждая из компонент типового обеспечения для матричного исчисления доступна из перечисленных источников. Не требуется, чтобы студенты имели доступ ко всем или даже к какой-либо из этих программ. Любая небольшая библиотека программ матричного исчисления достаточна, чтобы выполнить большинство задач и учебных проектов этой книги. Если некоторые из этих программ доступны на местах, то этот материал будет служить в качестве первоначального введения к их применению. Приведенные примеры

иллюстрируют различные способы, при помощи которых могут быть организованы программы матричного исчисления; в различных вычислительных центрах, возможно, применяется какая-то другая организация.

ПРИЛОЖЕНИЕ А

BLAS: ОСНОВНЫЕ ПОДПРОГРАММЫ ЛИНЕЙНОЙ АЛГЕБРЫ

Этот пакет состоит из 38 подпрограмм, которые можно вызывать из программ, написанных на Фортране, и которые предназначены для выполнения основных операций нижнего уровня вычислительной линейной алгебры. Подпрограммы разработаны К. Л. Лоусоном, Р. Дж. Хэнсоном, Д. Р. Кинкейрдом и Ф. Т. Крогом в течение 1973—1977 годов в рамках проекта, субсидированного Специальной группой по вычислительной математике (SIGNUM) Ассоциации по вычислительной технике (ACM), и основной ссылкой на их работу может служить статья (Lawson et al., 1979). Подпрограммы пакета разработаны на портабельном Фортране и на языке ассемблера для IBM 360/370, CDC 6600—7600 и UNIVAC1108.

Пакет был создан по следующим причинам:

1. Он служит методическим пособием при проектировании и написании больших программ и способствует их модульному представлению.
2. Пакет улучшает самодокументированность программ благодаря использованию стандартных имен.
3. Значительная часть времени при исполнении многих больших программ тратится на операции нижнего уровня, и версии подпрограмм пакета на языке ассемблера могут существенно снизить затраты на вычисления.
4. Некоторые из операций включают в себя алгоритмические тонкости и тонкости реализации, которые вероятнее всего игнорируются в типичном прикладном программном обеспечении.

В табл. А.1 дана сводка реализованных операций и соглашений, принятых для имен. Каждая операция имеет корневое имя (например, DOT для скалярного произведения) и приставку, указывающую на типы операндов (например, I для целых операндов). Суффиксы в именах подпрограмм скалярного произведения указывают на их модификации. Оставшаяся часть этого приложения (вплоть до примеров) полностью взята из статьи (Lawson et al., 1979).

4. Соглашения, принятые в программах

В качестве аргументов допускается задание векторов, расположенных в памяти с интервалами между элементами. Эти интервалы

Таблица А.1

Сводка имен и назначений подпрограмм пакета BLAS

Назначение	Приставки и суффиксы	Корневое имя
Скалярное произведение	SDS- DS- DQ-I DQ-A C-U C-C D- S-	-DOT-
Константа, помноженная на вектор плюс вектор	C- D- S-	-AXPY
Вращение Гивенса	D- S-	-ROTG
Применение вращения	D- S-	-ROT
Модифицированное вращение Гивенса	D- S-	-ROTMG
Применение модифицированного вращения	D- S-	-ROTM
Пересылка **x** в **y**	C- D- S-	-COPY
Перемена местами **x** и **y**	C- D- S-	-SWAP
Вторая (евклидова) норма	SC- D- S-	-NRM2
Сумма абсолютных значений *)	SC- D- S-	-ASUM
Умножение константы на вектор	CS- C- D- S-	-SCAL
Индекс элемента, имеющего максимальное абсолютное значение *)	IC- ID- IS-	-AMAX

*) Для комплексных компонент $z_j=x_j+iy_j$ эти подпрограммы вычисляют $|x_j|+|y_j|$ вместо $(x_j^2+y_j^2)^{1/2}$.

указываются параметрами приращения. Например, предположим, что вектор **x** с компонентами x_i, $i=1, \ldots, N$, расположен в массиве DX ( ) двойной точности с параметром приращения INCX. Если $INCX\geqslant 0$, то x_i расположены в DX $(1+(i-1)*INCX)$. Если $INCX<0$, то x_i расположены в DX $(1+(N-i)*|INCX|)$. Этот способ индексирования в случае $INCX<0$ позволяет избегать отрицательных индексов в массиве DX ( ) и тем самым допускает оформление подпрограмм на Фортране. Для операций 26—38 разд. 5 разрешены только положительные значения INCX, поскольку каждая из них имеет единственный векторный аргумент.

Предполагается, что циклы во всех подпрограммах обрабатывают элементы векторных аргументов в порядке возрастания индексов компонент вектора, т. е. в порядке следования x_i, $i=1, \ldots, N$. Это означает, что обработка элементов проводится в порядке, обратном их расположению в памяти, когда $INCX<0$. Если подпрограммы пакета реализуются для машин с параллельной обработкой данных, рекомендуется придерживаться, насколько это возможно, рассмотренного порядка обработки.

5. Спецификации подпрограмм пакета BLAS

Формальные параметры подпрограмм имеют следующие типы и размерности:

$mx = \max(1, N*|INCX|), \qquad my = \max(1, N*|INCY|).$

```
INTEGER  N, INCX, INCY, IMAX, QC(10)
REAL  SX(mx), SY(my), SA, SB, SC, SS
REAL  SD1, SD2, SB1, SB2, SPARAM(5), SW
DOUBLE PRECISION  DX(mx), DY(my), DA, DB, DC, DS
DOUBLE PRECISION  DD1, DD2, DB1, DB2, DPARAM(5), DW
COMPLEX  CX(mx), CY(my), CA, CW
```

Подпрограммы-функции пакета имеют следующие типы:

```
INTEGER  ISAMAX, IDAMAX, ICAMAX
REAL  SDOT, SDSDOT, SNRM2, SCNRM2, SASUM, SCASUM
DOUBLE PRECISION  DSDOT, DDOT, DQDOTI, DQDOTA, DNRM2, DASUM
COMPLEX  CDOTC CDOTU
```

Подпрограммы-функции скалярного произведения

1. SW=SDOT (N, SX, INCX, SY, INCY)

$$w := \sum_{i=1}^{N} x_i y_i.$$

2. DW=DSDOT (N, SX, INCX, SY, INCY)

$$w := \sum_{i=1}^{N} x_i y_i.$$

В DSDOT используется накопление с удвоенной точностью.

3. SW=SDSDOT (N, SB, SX, INCX, SY, INCY)

$$w := b + \sum_{i=1}^{N} x_i y_i.$$

Накопление в скалярном произведении и сложение с b производится с удвоенной точностью. Преобразование окончательного результата к одинарной точности выполняется тем же способом, как и во встроенной функции языка Фортран SNGL( ).

4. DW=DDOT (N, DX, INCX, DY, INCY)

$$w := \sum_{i=1}^{N} x_i y_i.$$

5. DW=DQDOTI (N, DB, QC, DX, INCX, DY, INCY)

$$w := b + \sum_{i=1}^{N} x_i y_i.$$

Входные данные b, x и y преобразуются внутри этой функции к представлению с расширенной точностью Результат помещается с расширенной точностью в QC ( ) и в качестве значения функции DQDOTI становится доступным с удвоенной точностью.

6. DW=DQDOTA (N, DB, QC, DX, INCX, DY, INCY)

$$w := c \quad = b + c + \sum_{i=1}^{N} x_i y_i$$

Значение c задается на входе в QC ( ) с расширенной точностью. Значение c должно быть результатом предыдущего исполнения DQDOTI или DQDOTA, поскольку в пакете не предусмотрены другие способы определения чисел с расширенной точностью. Вычисления выполняются в арифметике расширенной точности, результат помещается с расширенной точностью в QC ( ) и в качестве значения функции DQDOTA становится доступным с удвоенной точностью.

7. CW=CDOTC (N, CX, INCX, CY, INCY)

$$w := \sum_{i=1}^{N} \bar{x}_i y_i.$$

Суффикс C в имени функции CDOTC указывает, что используются комплексно-сопряженные значения компонент x_i.

8. CW=CDOTU (N, CX, INCX, CY, INCY)

$$w := \sum_{i=1}^{N} x_i y_i.$$

Суффикс U в имени функции CDOTU указывает, что не используются комплексно-сопряженные значения компонент x_i. В предыдущих восьми подпрограммах-функциях значение $\sum_{i=1}^{N}$ полагается равным нулю, если $N \leqslant 0$.

Элементарная векторная операция: $y := ax + y$

```
 9  CALL  SAXPY(N, SA, SX, INCX, SY, INCY).
10  CALL  DAXPY(N, DA, DX, INCX, DY, INCY).
11  CALL  CAXPY(N, CA, CX, INCX, CY, INCY)
```

Если a=0 или $N \leqslant 0$, немедленно происходит выход из этих подпрограмм.

Построение плоского вращения Гивенса

```
12  CALL  SROTG(SA, SB, SC, SS).
13. CALL  DROTG(DA, DB, DC, DS).
```

Каждая из этих подпрограмм для данных a и b вычисляет:

$$\sigma = \begin{cases} \operatorname{sgn}(a), & \text{если } |a| > |b|, \\ \operatorname{sgn}(b), & \text{если } |b| \geqslant |a|, \end{cases} \qquad r = \sigma (a^2 + b^2)^{1/2},$$

$$c = \begin{cases} a/r, & \text{если } r \neq 0, \\ 1, & \text{если } r = 0, \end{cases} \qquad s = \begin{cases} b/r, & \text{если } r \neq 0, \\ 0, & \text{если } r = 0 \end{cases}$$

Таким образом, вычисленные числа c, s и r удовлетворяют матричному уравнению

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}.$$

Введение параметра σ не существенно для вычисления матрицы вращения Гивенса, однако его использование допускает последующее устойчивое восстановление «c» и «a» только из одного хранимого в памяти машины числа. Для этой цели подпрограммы вычисляют

$$z = \begin{cases} s, & \text{если } |a| > |b|; \\ 1/c, & \text{если } |b| \geqslant |a| \text{ и } c \neq 0; \\ 1, & \text{если } c = 0. \end{cases}$$

Подпрограммы помещают r на место a и z — на место b, а также вычисляют c и s

Если пользователь хочет восстановить параметры c и s из z, это может быть сделано следующим образом.

если $z=1$, положить $c=0$ и $s=1$;
если $|z|<1$, положить $c=(1-z^2)^{1/2}$ и $s=z$;
если $|z|>1$, положить $c=1/z$ и $s=(1-c^2)^{1/2}$.

Применение плоского вращения

```
14 CALL SROT(N, SX, INCX, SY, INCY, SC, SS)
15 CALL DROT(N, DX, INCX, DY, INCY, DC, DS)
```

Каждая из этих подпрограмм вычисляет

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{для} \quad i = 1, \ldots, N$$

Если $N \leqslant 0$ или $c=1$ и $s=0$, немедленно происходит выход из этих подпрограмм.

Построение модифицированного преобразования Гивенса

```
16 CALL SROTMG(SD1, SD2, SB1, SB2, SPARAM)
17 CALL DROTMG(DD1, DD2, DB1, DB2, DPARAM).
```

Входные величины d_1, d_2, b_1 и b_2 определяют двумерный вектор $(a_1, a_2)^T$:

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} d_1^{1/2} & 0 \\ 0 & d_2^{1/2} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Подпрограммы вычисляют модифицированную матрицу вращения Гивенса H, которая обращает b_2, а значит, и a_2 в нуль Эта матрица представляется в памяти машины в массивах SPARAM ( ) или DPARAM ( ) следующим образом (ниже не приводятся элементы массива PARAM, которые остаются неизменными):

PARAM(1) = 1	PARAM(1) = 0	PARAM(1) = —1 случай перемасштабирования
$h_{12} = 1$, $h_{21} = -1$	$h_{11} = h_{22} = 1$	PARAM(2) = h_{11}
PARAM(2) = h_{11}	PARAM(3) = h_{21}	PARAM(3) = h_{21}
PARAM(5) = h_{22}	PARAM(4) = h_{12}	PARAM(4) = h_{12}
		PARAM(5) = h_{22}

Кроме того, PARAM(1)=—2 указывает, что H=I, где I — единичная матрица.

В результате преобразования значения d_1, d_2 и b_1 изменяются. Значение b_2, которое в результате преобразования должно быть положено равным нулю, остается неизменным в памяти.

Значение d_1 на входе должно быть неотрицательным, однако d_2 может быть отрицательным с целью удаления данных в проблеме наименьших квадратов

Применение модифицированного преобразования Гивенса

```
18  CALL SROTM(N, SX, INCX, SY, INCY, SPARAM)
19. CALL DROTM(N  DX, INCX, DY, INCY, DPARAM)
```

Пусть H — модифицированное преобразование Гивенса, определенное параметром SPARAM ( ) или DPARAM ( ). Подпрограммы вычислчют:

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} = H \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{для} \quad i = 1, \ldots, N.$$

Если $N \leqslant 0$ или H является единичной матрицей, немедленно происходит выход из этих подпрограмм.

Пересылка вектора **x** в **y**: **y**:=**x**

20. CALL SCOPY(N, SX, INCX, SY, INCY).
21. CALL DCOPY(N, DX, INCX, DY, INCY).
22. CALL CCOPY(N, CX, INCX, CY, INCY).

Если $N \leqslant 0$, немедленно происходит выход.

Перемена местами векторов **x** *и* **y**: **x**:=:**y**

23. CALL SSWAP(N, SX, INCX, SY, INCY).
24. CALL DSWAP(N, DX, INCX, DY, INCY).
25. CALL CSWAP(N, CX, INCX, CY, INCY).

Если $N \leqslant 0$, немедленно происходит выход.

Евклидова норма (или норма l_2*) вектора*

$$w = \left[\sum_{i=1}^{N} |x_i|^2 \right]^{1/2}.$$

26. SW = SNRM2(N, SX, INCX).
27. DW = DNRM2(N, DX, INCX).
28. SW = SCNRM2(N, CX, INCX).

Если $N \leqslant 0$, результат полагается равным нулю.

Сумма модулей компонент вектора

29 SW = SASUM(N, SX, INCX).
30 DW = DASUM(N, DX, INCX).
31. SW = SCASUM(N, CX, INCX).

Функции SASUM и DASUM вычисляют $w := \sum_{i=1}^{N} |x_i|$.

Функция SCASUM вычисляет

$$w := \sum_{i=1}^{N} \{|\operatorname{Re}(x_i)| + |\operatorname{Im}(x_i)|\}.$$

Если $N \leqslant 0$, то результат полагается равным нулю и немедленно происходит выход из этих функций.

Умножение вектора на скаляр: $\mathbf{x}:=a\mathbf{x}$

```
32. CALL  SSCAL(N, SA, SX, INCX).
33. CALL  DSCAL(N, DA, DX, INCX).
34. CALL  CSCAL(N, CA, CX, INCX).
35. CALL  CSSCAL(N, SA, CX, INCX).
```

Если $N \leqslant 0$, немедленно происходит выход.

Поиск индекса наибольшей компоненты вектора

```
36. IMAX = ISAMAX(N, SX, INCX).
37  IMAX = IDAMAX(N, DX, INCX).
38. IMAX = ICAMAX(N, CX, INCX).
```

Функции ISAMAX и IDAMAX вычисляют наименьший индекс i, такой, что $|x_i| = \max\{|x_j| : j=1, \ldots, N\}$. Функция ICAMAX вычисляет наименьший индекс i, такой, что $|x_i| = \max\{|\mathrm{Re}(x_j)| + |\mathrm{Im}(x_j)| : j=1, \ldots, N\}$. Если $N \leqslant 0$, результат полагается равным нулю и немедленно происходит выход из этих функций.

6. Реализация

В дополнение к версиям подпрограмм пакета на Фортране все подпрограммы, за исключением DQDOTI и DQDOTA, поставляются также на языке ассемблера для машин UNIVAC 1108, IBM 360/370 и CDC 6600 и 7600. Фортранные версии подпрограмм-функций DQDOTI и DQDOTA используют часть пакета подпрограмм для вычислений с многократной точностью, разработанного Брентом. На языке ассемблера эти две функции разработаны только для машины UNIVAC 1108.

Для CDC 6600 и 7600 только четыре подпрограммы на языке ассемблера используют преимущества конвейерной архитектуры этих машин. Четыре подпрограммы SDOT ( ), SAXPY ( ), SROT ( ) и SROTM ( ) принадлежат к числу тех подпрограмм, которые обычно используются в вычислениях в самых внутренних циклах.

7. Соответствие стандарту ANSI Фортрана

Американский национальный стандарт Фортрана ANSI X3.9 — 1966, на который мы будем ссылаться как на Фортран-66, широко поддерживается существующими компиляторами. На американский национальный стандарт Фортрана ANSI X3.9—1977 мы будем ссылаться как на Фортран-77.

Подпрограммы пакета BLAS должны содержать декларативные

операторы вида

REAL SX (MAXO (1, N*IABS (INCX)))

для точной спецификации длин массивов, где SX и IABS — формальные параметры. Однако ни в Фортране-66, ни в Фортране-77 такие операторы не допускаются. Использование оператора вида REAL SX (1) разрешается большинством компиляторов с Фортрана в тех случаях, когда трудно указать точную размерность массива. Этот вид записи размерности используется в подпрограммах пакета BLAS даже тогда, когда это не соответствует Фортрану-66. Фортран-77 допускает записи вида REAL SX(*) в таких случаях. Тем самым пакет BLAS может быть приведен в соответствие Фортрану-77 заменой 1 на * в декларативных операторах.

Пример А.1: *Произведение матриц* $C=AB$. Для данных матриц A и B размеров $M\times K$ и $K\times N$ вычислить матрицу их произведения $C=AB$ размерности $M\times N$.

Способ программирования произведения матриц основан на том, что каждый элемент c_{ij} матрицы C является скалярным произведением i-й строки матрицы A на j-й столбец матрицы B.

```
      DIMENSION A(20,20),B(15,10),C(20,15)
C
      MDA=20
      MDB=15
      MDC=20
C
      M=10
      K=15
      N=10
C     ВЫЧИСЛЯЕТСЯ СКАЛЯРНОЕ ПРОИЗВЕДЕНИЕ I-ОЙ СТРОКИ
C     A НА J-ЫЙ СТОЛБЕЦ B
C     ДЛИНА КАЖДОГО ВЕКТОРА РАВНА K.
C     ЗНАЧЕНИЕ MDA ОПРЕДЕЛЯЕТ РАССТОЯНИЕ МЕЖДУ
C     ЭЛЕМЕНТАМИ ВЕКТОРОВ-СТРОК A
                  DO 10 I=1,M
                  DO 10 J=1,N
10                C(I J)=SDOT(K,A(I,1),MDA,B(1,J),1)
```

Пример А.2: *Решение системы* $Ax=b$, *где* A — *верхняя треугольная матрица*. Требуется найти решение системы линейных алгебраических уравнений $Ax=b$, где A — верхняя треугольная невырожденная матрица размеров $N\times N$. Используемый метод основывается на том, что если мы вычислим компоненту $x_N=b_N/a_{NN}$, то имеем новую задачу с N—1 неизвестным и также с верхней треугольной матрицей, но с вектором правой части $(b_1-a_{1N}x_N, \ldots, b_{N-1}-a_{N-1,\,N}\,x_N)^T$. В этом примере вектор решения **x** помещается на месте вектора **b** в массиве B (*).

```
                  DO 20 II=1,N
                  I=N+1-II
                  B(I)=B(I)/A(I,I)
20                CALL SAXPY (I-1,-B(I),A(1,I),1,B,1)
```

Пример А.3: *Масштабирование строк матрицы* (уравновешивание

строк матрицы A размеров N×N). Иными словами, деление каждой ненулевой строки матрицы A на максимальный по модулю элемент этой строки. MDA — первый параметр размерности массива A (*, *).

```
      DO 40 I=1,N
      JMAX=ISAMAX(N,A(I,1),MDA)
      T=A(I,JMAX)
      IF(T EQ 0 E0) GO TO 40
      CALL SSCAL(N,1 E0/T,A(I,1),MDA)
40    CONTINUE
```

Например, при использовании ISAMAX ( ) для выбора ведущего элемента в методе исключения Гаусса основной цикл содержит оператор вида

```
IMAX=ISAMAX(N-J+1,A(J,J),1)+J-1
```

Здесь IMAX соответствует номеру строки, которая должна быть переставлена местами со строкой J. Тем самым значение смещения J—1 должно быть прибавлено к вычисленному значению ISAMAX( ), чтобы получить текущий номер строки для перестановки.

Пример А.4: Положить матрицу A размеров N×N равной единичной матрице, а затем положить матрицу B равной A. Отметим, что в SCOPY( ) используется 0 для задания расстояния между элементами в памяти машины первого векторного аргумента. Это дает возможность пересылать 0.E0 и 1.E0 во второй векторный аргумент.

Здесь MDA — первый параметр размерности массива A(*, *).

```
      DO 50 J=1,N
50    CALL SCOPY(N,0 E0,0,A(1,J),1)

      CALL SCOPY(N,1 E0,0,A,MDA+1)

      DO 60 J=1,N
60    CALL SCOPY(N,A(1,J),1,B(1,J),1)
```

Пример А. 5: Перестановка столбцов матрицы C размеров M×N. Номер столбца, который должен быть переставлен со столбцом J, задается в массиве IP(*) типа INTEGER и имеет значение IP(J).

```
      DO 70 J=1,N
      L=IP(J)
      IF(J NE L) CALL SSWAP(M,C(1,J),1,C(1,L),1)
70    CONTINUE
```

Пример А.6: Транспонирование матрицы A размеров N×N на том же месте в памяти (N>1). Здесь MDA — первый параметр размерности массива A (*, *).

```
      NM1=N-1
      DO 80 J=1,NM1
80    CALL SSWAP(N-J,A(J,J+1),MDA,A(J+1,J),1)
```

ПРИЛОЖЕНИЕ Б

ПОДПРОГРАММЫ ПАКЕТА LINPACK

Пакет LINPACK представляет собой набор подпрограмм на Фортране, предназначенных для решения линейных систем различных видов. Пакет охватывает квадратные матрицы: общего вида, ленточные, симметричные неопределенные, положительно определенные симметричные, треугольные и трехдиагональные, а также задачи наименьших квадратов, QR- и сингулярные разложения прямоугольных матриц. Подпрограммы спроектированы как совершенно машинно-независимые, полностью портабельные и обладающие удовлетворительной эффективностью для большинства вычислительных систем. Подробное описание пакета читатель сможет найти в работе (Dongarra et al., 1979).

LINPACK является одним из нескольких (включая описываемый ниже пакет EISPACK) крупных программистских разработок проекта NATS (National Activity to Test Software), финансируемого Национальным научным фондом и Департаментом энергетики США. Основными разработчиками пакета LINPACK являются Донгарра (J. J. Dongarra), Банч (J. R. Bunch), Молер (C. B. Moler) и Стьюарт (G. W. Stewart).

Выбор общей структуры пакета LINPACK в значительной степени определялся системой TAMPR и пакетом BLAS. Система TAMPR представляет собой инструментальную систему для разработки математического обеспечения, созданную Джимом Бойлом (Jim Boyle) и Кеном Дритцем (Ken Dritz) в Аргоннской национальной лаборатории. Она преобразовывает и форматирует структурные фортранные программы и обеспечивает автоматическую генерацию из основной программы нескольких ее версий, ориентированных на различные вычислительные условия. Так, например, основные подпрограммы пакета LINPACK используют комплексную арифметику, и система TAMPR создает из них версии одинарной и удвоенной точности для вещественной арифметики и удвоенной точности для комплексной арифметики.

Пакет BLAS (описанный в предыдущем разделе) вносит модульность и ясность в подпрограммы пакета LINPACK, а также повышает эффективность их исполнения. Однако путем применения TAMPR можно создать версию пакета LINPACK, которая не использует пакет BLAS или использует какой-нибудь другой набор основных векторных операций, как, например, набор операций для вычислительных машин с параллельной обработкой данных.

Необычное свойство алгоритмов пакета LINPACK состоит в том, что они все строго ориентированы на обработку матриц по столбцам, а не по строкам. Этим достигается существенно большая эффективность для вычислительных машин с виртуальной памятью благодаря согласованности между ориентацией Фортрана на про-

ведение операций по столбцам и страничной организацией.

Область применения пакета LINPACK можно представить себе, взглянув на нижеследующее описание программ и таблицу подпрограмм. Оставшаяся часть этого приложения полностью взята из работы Донгарра и др.

В пакете LINPACK применяются соглашения на имена подпрограмм, согласно которым каждое имя подпрограммы представляет собой закодированную спецификацию вычислений, выполняемых подпрограммой. Все имена состоят из пяти букв в виде TXXYY. Первая буква T указывает на тип элементов матрицы. Стандарт Фортрана позволяет использование трех таких типов:

S REAL
D DOUBLE PRECISION
C COMPLEX

Кроме того, некоторые диалекты Фортрана допускают данные комплексного типа удвоенной точности:

Z COMPLEX*16

Следующие две буквы XX указывают вид матрицы или ее разложения:

GE общего вида,
GB ленточная общего вида,
PO положительно определенная,
PP положительно определенная в упакованном виде,
PB положительно определенная ленточная,
SI симметричная неопределенная,
SP симметричная неопределенная в упакованном виде
HI эрмитова неопределенная,
HP эрмитова неопределенная в упакованном виде
TR треугольная,
GT трехдиагональная общего вида,
PT положительно определенная трехдиагональная,
CH разложение Холесского,
QR ортогонально-треугольное разложение,
SV сингулярное разложение.

Последние две буквы YY указывают вычисления, выполняемые конкретной подпрограммой:

FA факторизация,
CO факторизация и оценка обусловленности,
SL решение,
DI вычисление определителя и/или обратной матрицы и/или инерции матрицы,
DC разложение,
UD окаймление строкой или столбцом,

DD вычеркивание строки или столбца,
EX обмен.

Нижеследующая таблица демонстрирует все подпрограммы пакета LINPACK. Начальная буква S в именах подпрограмм может быть заменена на буквы D, C или Z, а буква C в именах подпрограмм, выполняющих операции только в комплексной арифметике, на букву Z.

	CO	FA	SL	DI
SGE	✓	✓	✓	✓
SGB	✓	✓	✓	✓
SPO	✓	✓	✓	✓
SPP	✓	✓	✓	✓
SPB	✓	✓	✓	✓
SSI	✓	✓	✓	✓
SSP	✓	✓	✓	✓
CHI	✓	✓	✓	✓
CHP	✓	✓	✓	✓
STR	✓		✓	✓
SGT			✓	
SPT			✓	

	DC	SL	UD	DD	EX
SCH	✓		✓	✓	✓
SQR	✓	✓			
SSV	✓				

Оставшиеся разделы этого введения охватывают некоторые вопросы проектирования подпрограмм и численного анализа, которые касаются пакета в целом. Каждая из глав с 1 по 11 описывает конкретные совокупности подпрограмм, упорядоченные приблизительно в соответствии с предыдущей таблицей. Каждая глава включает в себя разделы, имеющие заголовки «Обзор», «Применение» и «Примеры», которые предназначены для всех пользователей. Кроме того, многие главы содержат дополнительные разделы, имеющие заголовки «Алгоритмы», «Программирование» и «Рабочие характеристики», которые предназначены для пользователей, требующих более точную информацию. Для того чтобы сделать каждую главу совершенно самостоятельной, некоторая часть материала повторяется в нескольких родственных главах.

1. *Проектирование подпрограмм*

Выбор общей структуры пакета LINPACK в значительной степени определялся системой TAMPR и пакетом BLAS. TAMPR пред-

ставляет собой инструментальную систему для разработки математического обеспечения, созданную Бойлом и Дритцем. Система обрабатывает и форматирует программы на Фортране для выявления их структуры. Она также генерирует версии программ. «Основными версиями» всех подпрограмм пакета LINPACK являются версии, использующие комплексную арифметику; версии, которые используют вещественную арифметику с одинарной и удвоенной точностью, создавались автоматически системой TAMPR. Тем самым пользователь может переходить от одного типа арифметических операций к другому простым изменением декларативных операторов в своей программе и изменением первой буквы в применяемых подпрограммах пакета LINPACK.

При чтении текста любой подпрограммы пакета LINPACK, написанной на Фортране, вы увидите, что циклы и логические структуры явно выделены при помощи отступов, подобранных системой TAMPR.

Пакет BLAS представляет собой набор основных программ линейной алгебры, разработанный Лоусоном, Хэнсоном, Кинкейдом и Кроу. Программы пакета LINPACK отличаются повышенной скоростью выполнения, а также модульностью и ясностью исходного текста. LINPACK распространяется с версиями подпрограмм пакета BLAS, составленными на стандартном Фортране и предназначенными для обеспечения приемлемой скорости расчетов для большинства вычислительных систем. Однако для конкретной вычислительной установки можно заменить их версиями подпрограмм пакета BLAS на машинном языке и за счет этого, возможно, увеличить эффективность.

LINPACK спроектирован так, чтобы достигнуть полной машинной независимости. В нем нет машинно-зависимых констант, операторов ввода/вывода, обработки символов, операторов COMMON или EQUIVALENCE и смешанной арифметики Все подпрограммы (за исключением подпрограмм, имена которых начинаются с буквы Z) используют портабельное подмножество Фортрана, определенного верификатором PFORT, разработанным Райдером (Ryder, 1974).

В машинно-зависимых константах нет надобности потому, что почти не возникает необходимости в проверке чисел на «малость». Например, элементы, выбирающиеся в качестве ведущих элементов в методе исключения Гаусса, сравниваются с точным нулем, а не с некоторой малой величиной Проводится проверка на вырожденность вместо оценки обусловленности матрицы; это не только не вносит зависимости от машины, но также и гораздо надежнее Сходимость итераций в сингулярном разложении проверяется машинно-независимым способом при помощи операторов вида:

```
TEST1 = некоторое не малое число
TEST2 = TEST1 + некоторое возможно малое число
IF (TEST1. EQ. TEST2). . ..
```

Отсутствие смешанной арифметики означает, что подпрограммы одинарной точности не пользуются арифметическими операциями с удвоенной точностью и, следовательно, подпрограммы не требуют расширенной точности. Из этого также следует, что пакет LINPACK не содержит подпрограммы итерационного уточнения; однако пример в гл. 1 указывает, каким образом такая подпрограмма может быть добавлена в пакет с применением операций смешанной арифметики. (Некоторые подпрограммы пакета BLAS включают в себя смешанную арифметику, но они не применяются в пакете LINPACK.)

В некоторых подпрограммах пакета LINPACK могут происходить переполнение и антипереполнения. Случаи антипереполнения безвредны. Мы надеемся, что операционная система присваивает нуль таким величинам и продолжает процесс без какого-либо сообщения об ошибке. Для некоторых операционных систем может оказаться необходимым вставить управляющие карты или обращаться к специальным подпрограммам, чтобы добиться такой обработки антипереполнений.

Переполнения, если они происходят, гораздо более серьезны. Они должны рассматриваться как ошибочные ситуации, возникающие в результате неправильного применения подпрограмм или из-за необычного масштабирования. Многочисленные предосторожности предприняты в пакете LINPACK против переполнений, однако невозможно полностью предотвратить переполнения без серьезного ухудшения рабочих характеристик подпрограмм на разумно масштабированных задачах. Предполагается, что переполнения будут заставлять операционную систему останавливать вычисления и пользователь будет вынужден исправить свою программу или перемасштабировать задачу перед тем, как продолжить ее решение.

Фортран располагает матрицы в памяти по столбцам, и поэтому подпрограммы, в которых внутренний цикл пробегает столбец вверх или вниз, как например:

```
      DO  20  J=1,N
          DO  10  I=1,N
              A(I,J)=...
10        CONTINUE
20    CONTINUE,
```

дают последовательный доступ к памяти. Подпрограммы, в которых внутренний цикл пробегает по строке, дают непоследовательный доступ к памяти. LINPACK существенно «ориентирован на столбцы». Последовательный доступ предпочтительнее для операционных систем, работающих с виртуальной памятью или с какой-либо другой страничной организацией памяти. Почти все внутренние циклы встречаются внутри подпрограмм пакета BLAS, и, хотя BLAS допускает доступ к матрице по строкам, эта возможность нигде не используется в пакете LINPACK. Ориентация на обработку матриц

по столбцам требует пересмотра некоторых традиционных алгоритмов, однако в результате значительно улучшаются рабочие характеристики для вычислительных систем со страничной организацией памяти или с быстродействующей буферной памятью.

Все квадратные матрицы, которые входят в число параметров подпрограмм пакета LINPACK, указываются в списке параметров обращения тремя аргументами, например:

CALL SGEFA (A, LDA, N, . . .).

Здесь A — имя двумерного фортранного массива, LDA — первая размерность этого массива и N — порядок матрицы, размещенной в этом массиве или в части этого массива. Два параметра LDA и N имеют различный смысл и не обязаны иметь одно и то же значение. Размер памяти, отводимой для массива A, определяется декларативным оператором в вызывающей подпрограмме пользователя, и LDA является первой размерностью массива, указываемого этим оператором. Например, декларативные операторы

REAL A (50, 50)

или

DIMENSION A (50, 50)

должны сопровождаться инициализацией:

DATA LDA/50/

или оператором

LDA=50.

Значение LDA не должно меняться, если не меняется декларативный оператор. Порядок N конкретной матрицы коэффициентов может иметь любое значение, не превосходящее первую размерность массива, т. е. $N \leqslant LDA$. Значение N может изменяться программой пользователя, когда обрабатываются системы различных порядков.

Для задания прямоугольных матриц требуется четвертый аргумент, например,

CALL SQRDC (X, LDX, N, P, . . .).

Здесь матрица обозначена через X, согласно обозначениям, распространенным в статистике, LDX — первая размерность двумерного массива, N — число строк в матрице и P — число столбцов. Заметим, что соглашения Фортрана относительно типов переменных, принимаемых по умолчанию, должны быть отвергнуты объявлением P целой переменной. P — обычное обозначение, применяемое в статистике, и единственный аргумент в подпрограммах пакета LINPACK, на который не распространяется правило Фортрана относительно типа переменной, принимаемого по умолчанию.

Многие подпрограммы пакета LINPACK имеют один или два аргумента с именами JOB и INFO. JOB является всегда входным

параметром. Он устанавливается пользователем, часто включением целой константы в список параметров оператора обращения, для указания того, какие из нескольких возможных вычислений должны быть выполнены. Например, SGESL решает систему уравнений или с факторизованной матрицей, или с факторизованной транспонированной, и JOB следует положить равным нулю или ненулевому значению соответственно.

INFO всегда выходной параметр. Он применяется для сообщения различных видов диагностической информации из подпрограмм пакета LINPACK. В некоторых случаях INFO может рассматриваться как параметр сообщений об обшибках

Например, в подпрограмме SPOFA он используется для указания того, что матрица не положительно определенная. В других случаях INFO может быть одной из главных выходных величин. Например, в SCHDC он определяет ранг полуопределенной матрицы.

Несколько подпрограмм пакета LINPACK требуют больше места в памяти для промежуточных результатов, чем это обеспечивается главными параметрами. Эти подпрограммы имеют параметр WORK, который является одномерным массивом, длина которого обычно равна числу строк или столбцов обрабатываемой матрицы. Пользователя редко будет интересовать содержимое массива WORK, и поэтому он просто должен предусмотреть соответствующее объявление массива.

Большинство подпрограмм пакета LINPACK не обращаются к какой-либо другой подпрограмме пакета. Единственное исключение составляют подпрограммы оценки обусловленности с именами, заканчивающимися на CO, каждая из которых обращается к соответствующей FA-подпрограмме для факторизации матрицы. Однако почти все подпрограммы пакета LINPACK обращаются к одной или нескольким подпрограммам пакета BLAS Чтобы упростить комплектацию библиотек, тексты каждой подпрограммы пакета LINPACK включают в себя комментарии, которые перечисляют все подпрограммы пакета BLAS и функции Фортрана, требуемые этой подпрограммой.

2 *Использование*

Матрицы общего вида одинарной точности. Четыре подпрограммы SGECO, SGEFA, SGESL и SGEDI предназначены для матриц общего вида, представленных с одинарной точностью. Обычно SGECO или SGEFA вызываются один раз для факторизации конкретной матрицы, а затем вызываются SGESL и SGEDI для применения полученной факторизации столько раз, сколько это необходимо.

SGECO применяет исключение Гаусса с частичным выбором ведущего элемента для вычисления LU-разложения матрицы и затем

оценивает ее обусловленность. Список параметров обращения имеет вид

CALL SGECO (A, LDA, N, IPVT, RCOND, Z)

На входе,

A — двумерный массив размеров (LDA, N), содержащий факторизуемую матрицу.
LDA — первая размерность массива A.
N — порядок матрицы A и число элементов векторов IPVT и Z.

На выходе,

A — содержит в своем верхнем треугольнике верхнюю треугольную матрицу U и в своем строго нижнем треугольнике множители, необходимые для построения такой матрицы L, что A=LU.
IPVT — одномерный целый массив длины N, содержащий информацию о выборе ведущих элементов для построения перестановок в L. Конкретно, элемент массива IPVT(K) содержит индекс строки, переставленной с K-й строкой.
RCOND — простая переменная, содержащая оценку величины 1/k(A), обратной к числу обусловленности матрицы A. Если RCOND настолько мало, что логическое выражение 1.0+RCOND .EQ. 1.0 истинно, то A может, как правило, рассматриваться как вырожденная с точностью выполнения машинных операций. Если переменная RCOND точно равна нулю, то SGESL и SGEDI могут делить на нуль.
Z — одномерный массив длины N, используемый как рабочий. Если A близка к вырожденной матрице, то Z будет содержать близкий к нулевому вектор в том смысле, что $||Az||=RCOND \cdot ||A|| \cdot ||z||$ (см. разд 4).

SGEFA следует применять вместо SGECO, если не нужна оценка обусловленности матрицы A. Список параметров обращения к ней имеет вид

CALL SGEFA (A, LDA, N, IPVT, INFO)

На входе,

A — двумерный массив размеров (LDA, N), содержащий факторизуемую матрицу.
LDA — первая размерность массива A.
N — порядок матрицы A и число элементов вектора IPVT.

На выходе,

A — содержит в своем верхнем треугольнике верхнюю треугольную матрицу U и в своем строго нижнем тре-

угольнике множители, необходимые для построения такой матрицы L, что A=LU.

IPVT — одномерный целый массив длины N, содержащий информацию о выборе ведущих элементов для построения перестановок в L. Конкретно, IPVT(K) содержит индекс строки, переставленной с K-й строкой.

INFO — целая переменная, значение которой устанавливается подпрограммой SGEFA. Если она равна 0, то SGESL и SGEDI могут быть использованы без осложнений. Если INFO=K≠0, то SGESL и SGEDI могут делить на U(K, K)=O. Если матрица U имеет несколько нулевых диагональных элементов, то значение K будет равно индексу последнего нулевого элемента. Несмотря на то что ненулевое значение INFO формально указывает на вырожденность матрицы A, RCOND представляет собой более надежный индикатор вырожденности.

SGECO обычно применяется первой для факторизации матрицы и оценки ее обусловленности. Фактическая факторизация осуществляется подпрограммой SGEFA, которая может быть применена вместо SGECO, если число обусловленности не нужно. Время выполнения SGECO примерно в (1+9/N) раз превосходит время исполнения SGEFA. Поэтому, когда N=9, то затраты SGECO в два раза превосходят затраты SGEFA, но когда N=90, то затраты SGECO — на 10% больше.

Поскольку всякая матрица имеет LU-разложение, SGECO или SGEFA не имеют параметров, сигнализирующих об ошибке. Однако множители могут быть вырожденными и, следовательно, неприменимыми в SGESL или SGEDI. Или RCOND, или INFO должны быть проверены до обращения к SGESL.

SGESL применяет LU-разложение матрицы A для решения линейных систем вида

$$A\mathbf{x}=\mathbf{b} \text{ или } A^T\mathbf{x}=\mathbf{b},$$

где A^T — транспонированная матрица A. Список параметров обращения имеет вид

CALL SGESL (A, LDA, N, IPVT, B, JOB).

На входе,

A — двумерный массив размеров (LDA,N), содержащий факторизованную матрицу, вычисленную подпрограммой SGECO или SGEFA. Массив не меняется подпрограммой SGESL.

LDA — первая размерность массива A.

N — порядок матрицы A и число элементов в векторах B и IPVT.

IPVT — одномерный целый массив длины N, содержащий информацию о ведущих элементах, полученную подпрограммой SGECO или SGEFA.

B — одномерный массив длины N, содержащий правую часть **b** совместной системы линейных уравнений

$$A\mathbf{x}=\mathbf{b} \text{ или } A^T\mathbf{x}=\mathbf{b}.$$

JOB — указывает режим вычислений. Если JOB=0, то решается система $A\mathbf{x}=\mathbf{b}$, и если значение JOB не равно нулю, то решается система $A^T\mathbf{x}=\mathbf{b}$.

На выходе,

B — содержит решение **x**.

Матрицы общего вида удвоенной точности. Списки параметров обращения к подпрограммам DGECO, DGEFA, DGESL и DGEDI, предназначенным для матриц общего вида, представленных с удвоенной точностью, являются теми же, что и для соответствующих подпрограмм одинарной точности «S», за исключением того, что параметры A,B,RCOND,DET,Z и WORK имеют тип DOUBLE PRECISION.

Комплексные матрицы общего вида. Списки параметров обращения к подпрограммам CGECO, CGEFA, CGESL и CGEDI, предназначенным для комплексных матриц общего вида, являются теми же, что и для соответствующих подпрограмм одинарной точности «S», за исключением того, что параметры A, B, DET, Z и WORK имеют тип COMPLEX; RCOND имеет тип REAL и при использовании подпрограммы CGESL при ненулевом значении JOB решается система с комплексно-сопряженной транспонированной матрицей.

Комплексные матрицы общего вида удвоенной точности. На вычислительных системах, для которых такие матрицы представимы, списки параметров обращения к подпрограммам ZGECO, ZGEFA, ZGESL и ZGEDI, предназначенным для комплексных матриц общего вида, представленных с удвоенной точностью, являются теми же, что и для соответствующих подпрограмм одинарной точности «S», за исключением того, что параметры A, B, DET, Z и WORK имеют тип COMPLEX*16; RCOND имеет тип REAL и при использовании подпрограммы ZGESL при ненулевом значении JOB решается система с комплексно-сопряженной транспонированной матрицей.

3. Примеры

Нижеследующие программные сегменты иллюстрируют применение подпрограмм для матриц общего вида, представленных с одинарной точностью. Примеры, показывающие применение подпрограмм «D», «C» и «Z», могут быть получены изменением имен подпрограмм и типов в декларативных операторах.

Первая программа факторизует матрицу, проверяет на близость ее к вырожденной и затем решает одну систему $\mathbf{Ax}=\mathbf{b}$.

```
      REAL A(50,50),B(50),Z(50),T,RCOND
      INTEGER IPVT(50)
      DATA LDA /50/
      N = ...
      DO 20 J = 1, N
         DO 10 I = 1, N
            A(I,J) = ...
   10    CONTINUE
   20 CONTINUE
      CALL SGECO(A,LDA,N,IPVT,RCOND,Z)
      WRITE(..., ...) RCOND
      T = 1.0 + RCOND
      IF (T .EQ. 1.0) GO TO 90
      DO 30 I = 1, N
         B(I) = ...
   30 CONTINUE
      CALL SGESL(A,LDA,N,IPVT,B,0)
      DO 40 I = 1, N
         WRITE(..., ...) B(I)
   40 CONTINUE
      STOP
   90 WRITE(..., 99)
   99 FORMAT(40H MATRIX IS SINGULAR TO WORKING PRECISION)
      STOP
      END
```

(Текст в операторе 99: «Матрица вырождена в пределах точности вычислений».)

Следующий программный фрагмент заменяет матрицу C, имеющую K столбцов, на матрицу $A^{-1}C$ без явного вычисления матрицы A^{-1}.

```
      CALL SGEFA(A,LDA,N,IPVT,INFO)
      IF (INFO .NE. 0) GO TO ...
      DO 10 J = 1, K
         CALL SGESL(A,LDA,N,IPVT,C(1,J),0)
   10 CONTINUE
```

Следующий программный фрагмент заменяет матрицу C, имеющую K строк, на матрицу CA^{-1} без явного вычисления A^{-1}.

Поскольку эта операция затрагивает строки, а не столбцы матрицы C, то способ, примененный в предыдущем примере, неприменим.

```
      CALL SGEFA(A,LDA,N,IPVT,INFO)
      IF (INFO .NE. 0) GO TO ...
      DO 30 I = 1, K
         DO 10 J = 1, N
            Z(J) = C(I,J)
   10    CONTINUE
         CALL SGESL(A,LDA,N,IPVT,Z,1)
         DO 20 J = 1, N
            C(I,J) = Z(J)
   20    CONTINUE
   30 CONTINUE
```

Следующий фрагмент печатает число обусловленности и определитель матрицы. Определитель печатается в воображаемом формате E, который допускает использование четырех цифр для порядка чисел и избегает трудности, связанные с потерей порядка и переполнением.

```
      CALL SGECO(A,LDA,N,IPVT,RCOND,Z)
      IF (RCOND  EQ  0.0) GO TO ...
      COND = 1.0/RCOND
      CALL SGEDI(A,LDA,N,IPVT,DET,Z,10)
      K = INT(DET(2))
      WRITE( .. , 10) COND,DET(1),K
   10 FORMAT(13H CONDITION =   E15.5/15H DETERMINANT = , F
```

(Текст в операторе 10: «Число обусловленности = . . . Определитель = ...»)

Следующий пример иллюстрирует, каким образом фактическое число обусловленности COND матрицы может быть вычислено при помощи формирования обратной матрицы. Это вычисление будет представлять интерес прежде всего для специалистов в численном анализе, которые захотят исследовать утверждение, что значение RCOND, получаемое SGECO, представляет собой хорошую оценку для 1/COND.

```
      ANORM = 0.0
      DO 10 J = 1, N
         ANORM = AMAX1(ANORM, SASUM(N,A(1,J),1) )
   10 CONTINUE
      CALL SGECO(A,LDA,N,IPVT,RCOND,Z)
      IF (RCOND .EQ. 0.0) GO TO ...
      CALL SGEDI(A,LDA,N,IPVT,DUMMY,WORK,1)
      AINORM = 0.0
      DO 20 J = 1, N
         AINORM = AMAX1(AINORM, SASUM(N,A(1,J),1) )
   20 CONTINUE
      COND = ANORM*AINORM
      RATIO = RCOND*COND
```

Выражение

$$\text{SASUM (N, A (1,J), 1)}$$

использующее подпрограмму из пакета BLAS вычисляет

$$\sum_{I=1}^{N} |A(I,J)|.$$

ЛИСТИНГИ ПОДПРОГРАММ

```
      SUBROUTINE SGECO(A,LDA,N,IPVT,RCOND,Z)
      INTEGER LDA,N,IPVT(1)
      REAL A(LDA,1), Z(1)
      REAL RCOND
C
C     SGECO ФАКТОРИЗУЕТ ВЕЩЕСТВЕННУЮ МАТРИЦУ
C     ИСКЛЮЧЕНИЕМ ГАУССА И ОЦЕНИВАЕТ ОБУСЛОВЛЕННОСТЬ
C     МАТРИЦЫ.
C
C     ЕСЛИ RCOND НЕ НУЖНО, SGEFA НЕМНОГО БЫСТРЕЕ.
C     ЧТОБЫ РЕШИТЬ A*X=B, НАДО ПРИМЕНИТЬ SGESL ВСЛЕД
C     ЗА SGECO. ЧТОБЫ ВЫЧИСЛИТЬ INVERSE(A)*C, НАДО
C     ПРИМЕНИТЬ SGESL ВСЛЕД ЗА SGECO. ЧТОБЫ ВЫЧИСЛИТЬ
C     DETERMINANT(A), НАДО ПРИМЕНИТЬ SGEDI ВСЛЕД ЗА SGECO.
C     ЧТОБЫ ВЫЧИСЛИТЬ INVERSE(A), НАДО ПРИМЕНИТЬ
C     SGEDI ВСЛЕД ЗА SGECO.
C
C     НА ВХОДЕ:
C        A            REAL(LDA, N) —
C                     ФАКТОРИЗУЕМАЯ МАТРИЦА.
C
C        LDA          INTEGER —
C                     ПЕРВАЯ РАЗМЕРНОСТЬ МАССИВА A.
C
C        N            INTEGER —
C                     ПОРЯДОК МАТРИЦЫ A.
C
C     НА ВЫХОДЕ:
C
C        A            ВЕРХНЯЯ ТРЕУГОЛЬНАЯ МАТРИЦА И
C                     МНОЖИТЕЛИ, ИСПОЛЬЗОВАННЫЕ ДЛЯ ЕЕ
C                     ПОЛУЧЕНИЯ.
C                     ФАКТОРИЗАЦИЯ МОЖЕТ БЫТЬ ЗАПИСАНА КАК
C                     A=L*U, ГДЕ L—ПРОИЗВЕДЕНИЕ
C                     ПЕРЕСТАНОВОК И ЕДИНИЧНЫХ НИЖНИХ
C                     ТРЕУГОЛЬНЫХ МАТРИЦ, А U—ВЕРХНЯЯ
C                     ТРЕУГОЛЬНАЯ МАТРИЦА.
C
C        IPVT         INTEGER(N) —
C                     ЦЕЛЫЙ ВЕКТОР ИНДЕКСОВ ПЕРЕСТАВЛЕННЫХ
C                     СТРОК.
C
C        RCOND        REAL —
C                     ОЦЕНКА ВЕЛИЧИНЫ, ОБРАТНОЙ К ЧИСЛУ
C                     ОБУСЛОВЛЕННОСТИ МАТРИЦЫ A.
```

```
C                  ДЛЯ СИСТЕМЫ A * X = B ОТНОСИТЕЛЬНЫЕ
C                  ВОЗМУЩЕНИЯ В А И В НА МАШИННУЮ
C                  EPSILON МОГУТ ПОВЛЕЧЬ ВОЗМУЩЕНИЯ В X
C                  НА ВЕЛИЧИНУ EPSILON/RCOND.
C                  ЕСЛИ RCOND НАСТОЛЬКО МАЛО, ЧТО
C                  ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ
C                  1.0+RCOND .EQ. 1.0
C                  ИСТИННО, ТО А МОЖЕТ БЫТЬ ВЫРОЖДЕННОЙ
C                  С МАШИННОЙ ТОЧНОСТЬЮ. В ЧАСТНОСТИ,
C                  RCOND РАВНО НУЛЮ, ЕСЛИ УСТАНОВЛЕНА
C                  ТОЧНАЯ ВЫРОЖДЕННОСТЬ ИЛИ
C                  ПРОИЗОШЛА ПОТЕРЯ ПОРЯДКА
C                  ОЦЕНКИ.
C
C      Z           REAL(N)—
C                  РАБОЧИЙ ВЕКТОР, СОДЕРЖИМОЕ КОТОРОГО
C                  ОБЫЧНО НЕ ИМЕЕТ ЗНАЧЕНИЯ. ЕСЛИ А БЛИЗКА
C                  К ВЫРОЖДЕННОЙ МАТРИЦЕ, ТО Z—
C                  ПРИБЛИЗИТЕЛЬНО НУЛЕВОЙ ВЕКТОР В ТОМ
C                  СМЫСЛЕ,
C                  ЧТО NOPM(A * Z) = RCOND * NORM(A) * NORM(Z)
C
C     LINPACK. ЭТА ВЕРСИЯ ДАТИРУЕТСЯ 14.08.1978.
C     КЛИВ МОУЛЕР, УНИВЕРСИТЕТ ШТАТА НЬЮ-МЕКСИКО,
C     АРГОННСКАЯ НАЦИОНАЛЬНАЯ ЛАБОРАТОРИЯ.
C
C     ПОДПРОГРАММЫ И ФУНКЦИИ
C
C     LINPACK SGEFA
C     BLAS SAXPY, SDOT, SSCAL, SASUM
      SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
      INTEGER LDA,N,IPVT(1),INFO
      REAL A(LDA,1)
C
C     SGEFA ФАКТОРИЗУЕТ ВЕЩЕСТВЕННУЮ МАТРИЦУ
C     ИСКЛЮЧЕНИЕМ ГАУССА.
C
C     SGEFA ОБЫЧНО ВЫЗЫВАЕТСЯ ПОДПРОГРАММОЙ SGECO,
C     ОДНАКО К НЕЙ МОЖНО НЕПОСРЕДСТВЕННО ОБРАЩАТЬСЯ,
C     ЕСЛИ ЗНАЧЕНИЕ RCOND НЕ НУЖНО.
C     (ВРЕМЯ ДЛЯ SGECO) = (1+9/N) * (ВРЕМЯ ДЛЯ SGEFA).
C
C   НА ВХОДЕ:
C      A           REAL(LDA, N)—
C                  ФАКТОРИЗУЕМАЯ МАТРИЦА.
C
C      LDA         INTEGER—
C                  ПЕРВАЯ РАЗМЕРНОСТЬ МАССИВА A.
C
C      N           INTEGER—
C                  ПОРЯДОК МАТРИЦЫ A.
C
C   НА ВЫХОДЕ:
C
C      A           ВЕРХНЯЯ ТРЕУГОЛЬНАЯ МАТРИЦА И
C                  МНОЖИТЕЛИ, ИСПОЛЬЗОВАННЫЕ ДЛЯ ЕЕ
```

```
C                     ПОЛУЧЕНИЯ. ФАКТОРИЗАЦИЯ МОЖЕТ БЫТЬ
C                     ЗАПИСАНА КАК A=L*U, ГДЕ L—
C                     ПРОИЗВЕДЕНИЕ ПЕРЕСТАНОВОК И
C                     ЕДИНИЧНЫХ НИЖНИХ ТРЕУГОЛЬНЫХ
C                     МАТРИЦ, А U—ВЕРХНЯЯ ТРЕУГОЛЬНАЯ
C                     МАТРИЦА.
C
C        IPVT         INTEGER(N)—
C                     ЦЕЛЫЙ ВЕКТОР ИНДЕКСОВ ПЕРЕСТАНОВОК
C                     СТРОК.
C
C        INFO         INTEGER—
C                     =0  НОРМАЛЬНОЕ ЗНАЧЕНИЕ,
C                     =K  ЕСЛИ U(K,K) .EQ. 0.0.
C                         ЭТО НЕ УСЛОВИЕ ОШИБКИ ДЛЯ
C                         ПОДПРОГРАММЫ, ОДНАКО ЭТО
C                         УКАЗЫВАЕТ, ЧТО SGESL ИЛИ SGEDI
C                         БУДУТ ДЕЛИТЬ НА НУЛЬ, ЕСЛИ К НИМ
C                         БУДЕТ ПРОИЗВЕДЕНО ОБРАЩЕНИЕ.
C                         ИСПОЛЬЗОВАТЬ RCOND В SGECO
C                         ДЛЯ НАДЕЖНОГО УКАЗАНИЯ
C                         ВЫРОЖДЕННОСТИ.
C
C     LINPACK. ЭТА ВЕРСИЯ ДАТИРУЕТСЯ 14.08.1978.
C     КЛИВ МОУЛЕР, УНИВЕРСИТЕТ ШТАТА НЬЮ-МЕКСИКО,
C     АРГОННСКАЯ НАЦИОНАЛЬНАЯ ЛАБОРАТОРИЯ.
C
C     ПОДПРОГРАММЫ И ФУНКЦИИ
C
C     BLAS SAXPY, SSCAL, ISAMAX
C
C     ВНУТРЕННИЕ ПЕРЕМЕННЫЕ:
C
      REAL T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C
C     ИСКЛЮЧЕНИЕ ГАУССА С ЧАСТИЧНЫМ ВЫБОРОМ
C     ВЕДУЩЕГО ЭЛЕМЕНТА.
C
      INFO=0
      NM1=N-1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K=1, NM1
         KP1=K+1
C
C        ПОИСК L=ИНДЕКСУ ПЕРЕСТАВЛЯЕМОЙ СТРОКИ.
C
```

ПРИЛОЖЕНИЕ В

ПОДПРОГРАММЫ РЕШЕНИЯ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ БИБЛИОТЕКИ IMSL

IMSL означает название фирмы International Mathematical and Statistical Libraries, Inc. (Международные математические и статистические библиотеки, Инк.), которая поставляет обширную библиотеку математических подпрограмм. Стратегия фирмы состоит в том, что низкая стоимость программного продукта, согласованная с его широким распространением, приведет к коммерческому успеху. Библиотека насчитывает свыше 400 подпрограмм, и ее специальные версии доступны для вычислительных машин Burroughs, CDC, Data General, DEC, Hewlett-Packard, Honeywell, IBM, Univac и Xerox. Библиотека сдается в аренду приблизительно за 100 долларов в месяц и модифицируется каждые 12—18 месяцев. Дальнейшую информацию о библиотеке и ее составе можно найти в работе IMSL Library General Information, vol. 1, опубликованной фирмой IMSL, Inc.

B.1. Справочник пользователя библиотеки IMSL

Справочное руководство по использованию любой библиотеки должно быть не требующим дополнительных материалов справочником пользователя. Ниже приводится вводный материал из руководства библиотеки IMSL, посвященный линейным алгебраическим уравнениям, чтобы показать подход фирмы к обеспечению пользователя документацией.

Линейные алгебраические уравнения

Эта глава содержит подпрограммы для решения систем линейных алгебраических уравнений. Сюда включены также подпрограммы для решения линейных задач наименьших квадратов и для выполнения сингулярного разложения матриц. Нижеследующее рассмотрение суммирует и объясняет возможности программ, включенных в гл. L.

Краткие сведения о возможностях подпрограмм гл. L

Эта глава содержит следующие подпрограммы:

Решение линейных уравнений:

Вещественные заполненные матрицы.

LEQT1F — решение с экономией памяти.

LEQT2F — решение с высокой точностью.

Положительно определенные матрицы — способ хранения в памяти, принятый для симметричных матриц.

LEQT1P — решение с экономией памяти.
LEQT2P — решение с высокой точностью.

Положительно определенные матрицы — способ хранения в памяти, принятый для ленточных симметричных матриц.
LEQ1PB — решение с экономией памяти.
LEQ2PB — решение с высокой точностью.

Симметричные неопределенные матрицы.
LEQ1S — решение с экономией памяти.
LEQ2S — решение с высокой точностью.

Комплексные заполненные матрицы.
LEQT1C — решение с экономией памяти.
LEQ2C — решение с высокой точностью.

Вещественные ленточные матрицы — способ хранения в памяти, принятый для ленточных матриц.
LEQT1B — решение с экономией памяти.
LEQT2B — решение с высокой точностью.

Обращение матриц:

Вещественные заполненные матрицы.
LINV1F — обращение с экономией памяти.
LINV2F — обращение с высокой точностью.

Положительные определенные матрицы — способ хранения в памяти, принятый для симметричных матриц.
LINV1P — обращение с экономией памяти.
LINV2P — обращение с высокой точностью.

Положительно определенные матрицы — способ хранения в памяти, принятый для ленточных симметричных матриц.
LIN1PB — обращение с экономией памяти.
LIN2PB — обращение с высокой точностью.

Симметричные неопределенные матрицы.
LEQ1S — обращение с экономией памяти.
LEQ2S — обращение с высокой точностью.
Обращение матрицы на том же месте памяти и/или решение системы.
LINV3F — способ хранения в памяти для полных матриц.
LINV3P — положительно определенные матрицы — способ хранения, принятый для симметричных матриц.

Комплексные заполненные матрицы.
LEQT1C — обращение с экономией памяти.
LEQ2C — обращение с высокой точностью.

Вещественные ленточные матрицы — способ хранения в памяти, принятый для ленточных матриц.

LEQT1B — обращение с экономией памяти.
LEQT2B — обращение с высокой точностью.

Разложение, подстановка, уточнение.
Вещественные заполненные матрицы.
LUDATF — разложение.
LUELMF — подстановка.
LUREFF — уточнение.

Положительно определенные матрицы — способ хранения, принятый для симметричных матриц.
LUDECP — разложение.
LUELMP — подстановка.
LUREFP — уточнение.

Положительно определенные матрицы — способ хранения в памяти, принятый для ленточных симметричных матриц.
LUDAPB — разложение.
LUELPB — подстановка.
LUREPB — уточнение.

Возможности для прямоугольных матриц.
LLSQF — решение линейной задачи наименьших квадратов.
LSVDF — псевдообращение.
LSVDF — сингулярное разложение.

Примечание: подпрограммы LUDAPB, LUDATF и LUDECP могут быть использованы для вычисления определителя матрицы, а LSVDB может быть использована для вычисления сингулярного разложения двудиагональной матрицы (главная диагональ плюс верхняя диагональ).

Характерные возможности

Подпрограммы LEQT1F и LEQT2F содержат много особенностей, обычно не присутствующих в подпрограммах решения линейных уравнений. Эти особенности включают в себя:

1. Уравновешивание строк матриц.
2. Частичный выбор ведущего элемента.
3. Апостериорную проверку точности (по желанию пользователя).
4. Применение итерационного уточнения, если только это затребовано.

До публикации диссертации Джеймса Банча в 1969 г. (Калифорнийский университет в Беркли) не существовало устойчивого алгоритма для симметричных неопределенных матриц, который сохранял бы симметрию на всем протяжении вычислений. Фирма IMSL реализовала этот алгоритм в подпрограммах LEQ1S и LEQ2S. Подпрограммы LLSQF, LSVDF и LSVDB основаны на подпрограм-

мах из книги (Lawson, C., and Hanson, R., Solving Least Squares Problems, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974).

Соглашения на имена подпрограмм этой главы

Все имена подпрограмм этой главы начинаются с буквы L и заканчиваются следующими буквами:

F = вещественная заполненная матрица;
P = симметричная положительно определенная матрица; способ хранения в памяти, принятый для симметричных матриц;
PB = симметричная положительно определенная матрица; способ хранения в памяти, принятый для ленточных симметричных матриц;
S = симметричная матрица; способ хранения в памяти, принятый для симметричных матриц;
C = комплексная заполненная матрица;
B = ленточная матрица; способ хранения в памяти, принятый для ленточных матриц.

Специальные инструкции по применению

Подпрограммами гл. L применяются различные способы размещения матриц в памяти, например, способы для симметричных, ленточных симметричных и ленточных матриц, рассмотренные во введении в разделе 5.6. «Способы хранения матриц и векторов».

Некоторые из подпрограмм библиотеки IMSL для решения линейных уравнений имеют параметр IDGT, представляющий собой десятичные разряды точности элементов матрицы A. Обработка этого параметра меняется от одной подпрограммы к другой. Чтобы понять назначение этого параметра, рассмотрим его применение отдельными подпрограммами.

LEQT1F: если значение IDGT больше нуля, то подпрограмма вычисляет решение $\overline{X}$ системы AX=B. Пусть $\bar{\mathbf{x}}$ — столбец матрицы $\overline{X}$ и $\mathbf{b}$ — соответствующий столбец матрицы B. Возникает вопрос: является ли $\bar{\mathbf{x}}$ точным решением некоторой системы $\overline{A}\bar{\mathbf{x}}=\bar{\mathbf{b}}$, где матрица $\overline{A}$ согласуется поэлементно с матрицей A с точностью до первых IDGT цифр, а вектор $\bar{\mathbf{b}}$ согласуется с вектором $\mathbf{b}$ с точностью до первых IDGT цифр? Если вектор $\bar{\mathbf{x}}$ суть такое точное решение, то он принимается в качестве хорошего ответа. В противном случае вырабатывается предупреждающее сообщение. Этим способом дается гарантия правильности полученного ответа (если гарантия на самом деле должна быть дана). Если значение IDGT равно 0, проверка на точность опускается

Например, пусть IDGT=3. Тогда пользователь получает или

(а) решение возмущенной задачи, у которой возмущения не влияют на первые три цифры элементов A и **b**, или
(б) предупреждающее сообщение.

LEQT2F: Эта подпрограмма действует тем же способом, что и LEQT1F. Если вычисленное решение не выдерживает описанную выше проверку, то вырабатывается предупреждающее сообщение и подпрограмма пробует применить итерационное уточнение. Итерационное уточнение дорого как с точки зрения времени выполнения подпрограммы, так и с точки зрения размеров требуемой памяти. Поэтому LEQT2F применяет подпрограмму итерационного уточнения, только если она необходима. Итерационное уточнение последовательно улучшает ответ до тех пор, пока он не станет правильным с точностью выполнения машинных операций. Если матрица столь плохо обусловленная, что процесс уточнения не сходится, то вырабатывается окончательная ошибка. Если значение IDGT равно 0, то LEQT2F опускает проверку точности и выполнение итерационного уточнения.

LEQT1P, LEQT2P: Входной параметр IDGT также включается в список параметров обращения к этим подпрограммам. В настоящей версии подпрограммы LEQT1P он не используется. В подпрограмме LEQT2P параметр IDGT на выходе принимает значение, равное числу цифр в наибольшем по абсолютной величине элементе вектора решения, которые не менялись на первой итерации уточнения.

LINV1F, LINV2F, LINV1P, LINV2P: Параметр IDGT включен в список параметров этих подпрограмм обращения матриц, поскольку каждая из них в свою очередь обращается к подпрограммам решения линейных уравнений. За дальнейшей информацией относительно проверки точности пользователь может обратиться к работе (B. A. Chartres and J. C. Geuder, «Computable error bounds for direct solution of linear equations», JACM, Volume 14, January 1967, pp. 63—71).

Тонкости на заметку

Многие подпрограммы этой главы выдают сообщения об ошибках: «матрица алгоритмически вырождена» или «матрица алгоритмически не положительно определена». Численная фиксация вырожденности или определенности отличается от математической. Численная вырожденность (или алгоритмическая вырожденность) просто означает, что алгоритм потерпел неудачу, поскольку встретился малый ведущий элемент (малый или отрицательный в подпрограммах для положительно определенных матриц). Матрица данных может быть или не быть математически вырожденной. Если матрица данных математически вырождена, вероятнее всего будет сделано сообщение об ошибке. Подобным же образом для матрицы, которая

математически положительно определена, но почти вырождена, может быть получено сообщение о том, что она алгоритмически не является положительно определенной. Если матрица не положительно определена, вероятнее всего будет сделано сообщение об ошибке.

Пользователь может захотеть провести независимую проверку приемлемости полученного приближенного решения $\mathbf{x} \approx A^{-1}\mathbf{b}$. Следующее рассмотрение предназначено для того, чтобы дать некоторую информацию относительно этого вопроса. Дальнейшие детали могут быть найдены в рекомендуемой литературе.

Предполагается, что даны квадратная матрица A порядка n и вектор **b** правых частей и что вычислено приближенное решение $\mathbf{x} \approx A^{-1}\mathbf{b}$. Показатель p рабочей характеристики может быть определен следующим образом:

$$p = \max_{1 \le i \le n} \left[\frac{\left| b_i - \sum_{j=1}^{n} a_{ij} x_j \right|}{BN + AN \cdot \sum_{j=1}^{n} |x_j|} \right]$$

$$BN = \max_{1 \le i \le n} |b_i| \ ,$$

$$AN = \max_{1 \le i, j \le n} |a_{ij}| \ .$$

Если p мало (сравнимо с точностью машинных операций), то **x** — точное решение системы с матрицей коэффициентов A+E и вектором **b**+**f** правой части, где

$$E = (e_{ij}), \qquad |e_{ij}| \leqslant p \cdot AN,$$
$$\mathbf{f} = (f_i), \qquad |f_i| \leqslant p \cdot BN.$$

Более того, если существуют такие E и **f**, то p мало. В большинстве случаев матрица A и вектор **b** подвергаются воздействию входных ошибок округления при вычислении их элементов или ошибок в экспериментальном определении элементов. В этих случаях небольшой показатель p адекватно удостоверяет приемлемость приближенного решения **x**.

Версии подпрограмм решения линейных уравнений гл. L с экономией памяти обычно получают приближенное решение **x**, которое имеет небольшой показатель p. Однако существуют примеры, для которых это не так. См. [3, стр. 78], где приводится один такой пример. Когда производится обращение к LEQT1F или к LEQT2F и при этом выдерживается проверка точности IDGT, то p гарантировано меньше 10^{-IDGT}.

Версии подпрограмм решения линейных уравнений гл. L высокой точности вырабатывают или точное (с машинной точностью)

приближенное решение, или предупреждающее сообщение. (Не существует доказательства этого утверждения; не было также построено какого-либо контрпримера.)

Изредка может быть желательно вычислить верхнюю границу ошибки приближенного решения **x**. Если имеется приближенная обратная матрица $C \approx A^{-1}$ (она может быть вычислена одной из подпрограмм гл. L), то следующая формула дает такую границу

$$\|x - A^{-1}b\| \leqslant \frac{\|C(b - Ax)\|}{1 - \|CA - I\|},$$

где $\|CA - I\| < 1$, $\|\cdot\|$ означает согласованную векторно-матричную норму, такую, что $\|I\| = 1$. Рекомендуется применять удвоенную точность для вычислений по этой формуле, см. [4].

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА:

1. Forsythe G. E. and Moler C. E., Computer Solution of Linear Algebraic Systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1967. [Русский перевод: Дж. Форсайт, К. Молер. Численное решение систем линейных алгебраических уравнений —М.: Мир, 1969.]
2. Wilkinson J. H., Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.
3. Wilkinson J. H., "The solution of ill-conditioned linear equations", Mathematical Methods for Digital Computers, Editors, A. Ralston and H. Wilf, John Wiley, New York, 1967, Chapter 3.
4. Aird T. J. and Lynch R. E., "Computable accurate upper and lower error bounds for approximate solutions of linear algebraic systems", ACM TOMS, 1(3), 1975, 217—231.
5. Stewart G. W., Introduction to Matrix Computations, Academic Press, New York, 1973.

ИМЯ ПОДПРОГРАММЫ—LEQT1B

НАЗНАЧЕНИЕ —РЕШЕНИЕ ЛИНЕЙНЫХ УРАВНЕНИЙ — СПОСОБ ХРАНЕНИЯ В ПАМЯТИ ДЛЯ ЛЕНТОЧНЫХ МАТРИЦ—С ЭКОНОМИЕЙ ПАМЯТИ.

ИСПОЛЬЗОВАНИЕ —CALL LEQT1B (A,N,NLC,NUC,IA,B,M,IB,IJOB, XL,IER).

ПАРАМЕТРЫ

A —ВХОДНАЯ/ВЫХОДНАЯ МАТРИЦА РАЗМЕРОВ N НА (NUC+NLC+1).СМ. ПАРАМЕТР IJOB.

N —ПОРЯДОК МАТРИЦЫ A И ЧИСЛО СТРОК В B. (НА ВХОДЕ)
NLC —ЧИСЛО НИЖНИХ КОДИАГОНАЛЕЙ МАТРИЦЫ A. (НА ВХОДЕ)
NUC —ЧИСЛО ВЕРХНИХ КОДИАГОНАЛЕЙ МАТРИЦЫ A. (НА ВХОДЕ)
IA —ЧИСЛО СТРОК МАТРИЦЫ A, ТОЧНО РАВНОЕ ПЕРВОЙ РАЗМЕРНОСТИ В ДЕКЛАРАТИВНОМ ОПЕРАТОРЕ В ВЫЗЫВАЮЩЕЙ ПОДПРОГРАММЕ (НА ВХОДЕ)
B —ВХОДНАЯ/ВЫХОДНАЯ МАТРИЦА РАЗМЕРОВ N НА M. НА ВХОДЕ B СОДЕРЖИТ M ПРАВЫХ ЧАСТЕЙ УРАВНЕНИЯ AX=B. НА ВЫХОДЕ СОДЕРЖИТ МАТРИЦУ РЕШЕНИЙ X. ЕСЛИ IJOB=1, B НЕ ИСПОЛЬЗУЕТСЯ.
M —ЧИСЛО ПРАВЫХ ЧАСТЕЙ (СТОЛБЦОВ В B). (НА ВХОДЕ)
IB —ЧИСЛО СТРОК МАТРИЦЫ B, ТОЧНО РАВНОЕ ПЕРВОЙ РАЗМЕРНОСТИ В ДЕКЛАРАТИВНОМ ОПЕРАТОРЕ В ВЫЗЫВАЮЩЕЙ ПОДПРОГРАММЕ (НА ВХОДЕ)
IJOB —ВХОДНОЙ ПАРАМЕТР ВЫБОРА РЕЖИМА I РАБОТЫ ПОДПРОГРАММЫ.
I=0—ФАКТОРИЗАЦИЯ МАТРИЦЫ A И РЕШЕНИЕ УРАВНЕНИЯ AX=B. НА ВХОДЕ A СОДЕРЖИТ МАТРИЦУ КОЭФФИЦИЕНТОВ УРАВНЕНИЯ AX=B, ГДЕ A—КВАДРАТНАЯ ЛЕНТОЧНАЯ МАТРИЦА ПОРЯДКА N. A РАЗМЕЩАЕТСЯ В ПАМЯТИ СПОСОБОМ, ПРИНЯТЫМ ДЛЯ ЛЕНТОЧНЫХ МАТРИЦ, И ПОТОМУ ИМЕЕТ РАЗМЕРЫ N НА (NLC+NUC+1). НА ВЫХОДЕ A СОДЕРЖИТ МАТРИЦУ U ИЗ LU-РАЗЛОЖЕНИЯ МАТРИЦЫ A ПЕРЕСТАНОВКОЙ СТРОК. U РАЗМЕЩАЕТСЯ В ПАМЯТИ СПОСОБОМ ДЛЯ ЛЕНТОЧНЫХ МАТРИЦ.
I=1—ФАКТОРИЗАЦИЯ A. A СОДЕРЖИТ ТУ ЖЕ ВХОДНУЮ/ВЫХОДНУЮ ИНФОРМАЦИЮ, ЧТО И ПРИ IJOB=0.
I=2—РЕШЕНИЕ УРАВНЕНИЯ AX=B. ЭТОТ РЕЖИМ ПРЕДПОЛАГАЕТ, ЧТО LEQT1B УЖЕ ПРИМЕНЯЛАСЬ ПРИ IJOB=0 ИЛИ 1 И МАТРИЦА A УЖЕ ФАКТОРИЗОВАНА. В ЭТОМ СЛУЧАЕ ВЫХОДНЫЕ МАТРИЦЫ A И XL ДОЛЖНЫ БЫТЬ СОХРАНЕНЫ ДЛЯ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ПОДПРОГРАММОЙ LEQT1B
XL —РАБОЧИЙ МАССИВ РАЗМЕРОВ N*(NLC+1). ПЕРВЫЕ NLC*N ЭЛЕМЕНТОВ XL СОДЕРЖАТ КОМПОНЕНТЫ МАТРИЦЫ L LU-РАЗЛОЖЕНИЯ МАТРИЦЫ A ПЕРЕСТАНОВКОЙ СТРОК. ПОСЛЕДНИЕ N ЭЛЕМЕНТОВ СОДЕРЖАТ ИНДЕКСЫ ПЕРЕСТАВЛЕННЫХ СТРОК.
IER —ПАРАМЕТР ОШИБКИ. (НА ВЫХОДЕ) ОШИБКА IER=129 УКАЗЫВАЕТ, ЧТО МАТРИЦА A АЛГОРИТМИЧЕСКИ ВЫРОЖДЕНА (СМ. ВВЕДЕНИЕ К ГЛАВЕ L).

ТОЧНОСТЬ/АППАРАТУРА —ОДИНАРНАЯ И УДВОЕННАЯ/H 32,
—ОДИНАРНАЯ/H36,H48,H60

ТРЕБУЕМЫЕ ПОДПРОГРАММЫ БИБЛИОТЕКИ IMSL —UERTST, UGETIO.

ПРИМЕЧАНИЕ —ИНФОРМАЦИЯ О СПЕЦИАЛЬНЫХ ПРИМЕЧАНИЯХ И СОГЛАШЕНИЯХ ДОСТУПНА ИЗ ВВЕДЕНИЯ К РУКОВОДСТВУ ПО ИСПОЛЬЗОВАНИЮ ИЛИ ИЗ ПОДПРОГРАММЫ UHELP БИБЛИОТЕКИ IMSL.

Алгоритм

LEQT1B вычисляет перестановкой строк LU-разложение квадратной ленточной матрицы A порядка N, расположенной в памяти способом, принятым для ленточных матриц, и/или решает систему уравнений AX=B.

LEQT1B применяет уравновешивание строк и частичный выбор ведущего элемента.

См. статью:
Martin R. S. and Wilkinson J. H., "Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices", Numerishe Mathematik, 9(4) 1967, 279—301.

Замечания по программам

1. Когда IJOB=1, параметры B, M и IB не используются в подпрограмме.
2. Входная матрица A уничтожается, когда IJOB=0 или 1. Когда IJOB=0 или 2, матрица B уничтожается.
3. Определитель A может быть вычислен после применения LEQT1B следующим образом:

```
      DET = 1.0
      IXL = NLC*N
      DO 10 J=1,N
        IXL = IXL+1
        IP = XL(IXL)
        DET = DET*A(J,1)
        IF(IP .NE  J) DET=-DET
   10 CONTINUE
```

4. LEQT1B может быть применена для вычисления матрицы, обратной к ленточной. Для этого надо обратиться к LEQT1B с M=N, B=единичной матрице порядка N и IJOB=0. Когда N велико, может оказаться выгодным вычислять обратную матрицу последовательно по столбцам. Для этого первым обращением к LEQT1B матрица A факторизуется. Затем надо последовательно обращаться к ней с M=1, B=столбцу единичной матрицы и IJOB=2. Столбец матрицы B заменяется на соответствующий столбец матрицы, обратной к A.

Пример

В этом примере вводится матрица A третьего порядка и решается уравнение AX=B.

```
ВВОД :

INTEGER N,NLC,NUC,IA,M,IB,IJOB,IER
REAL    A(3,3),B(3,3),XL(6)

A   = [ 0.    1.   2. ]
      [ 2.    5.   1. ]
      [ 1.   17.   0. ]

N   =  3
NLC =  1
NUC =  1
IA  =  3

B   = [ 1.   0.   0. ]
      [ 0.   1.   0. ]
      [ 0.   0.   1. ]

M   =  3
IB  =  3
IJOB = 0
CALL LEQT1B(A,N,NLC,NUC,IA,B,M,IB,IJOB,XL,IER)
  :
  :
END

ВЫВОД :

B   = [  5.25    -2.125     .125   ]
      [ -2.125    1.065    -.0625  ]
      [   .125   - .0625    0625   ]

IER =  0
```

ИМЯ ПОДПРОГРАММЫ —LEQT1F
НАЗНАЧЕНИЕ —РЕШЕНИЕ ЛИНЕЙНЫХ УРАВНЕНИЙ— МАТРИЦА ЗАПОЛНЕННАЯ— С ЭКОНОМИЕЙ ПАМЯТИ.

ИСПОЛЬЗОВАНИЕ —CALL LEQT1F(A,M,N,IA,B,IDGT, WKAREA,IER)

ПАРАМЕТРЫ

A —НА ВХОДЕ МАТРИЦА РАЗМЕРОВ N НА N, СОДЕРЖАЩАЯ МАТРИЦУ КОЭФФИЦИЕНТОВ СИСТЕМЫ AX=B.
НА ВЫХОДЕ A ЗАМЕНЯЕТСЯ НА ЕЕ LU-РАЗЛОЖЕНИЕ ПЕРЕСТАНОВКОЙ СТРОК.

M —ЧИСЛО ПРАВЫХ ЧАСТЕЙ. (НА ВХОДЕ)

N —ПОРЯДОК МАТРИЦЫ A И ЧИСЛО СТРОК МАТРИЦЫ B.(НА ВХОДЕ)

IA —ЧИСЛО СТРОК МАТРИЦ A И B, ТОЧНО РАВНОЕ ПЕРВОЙ РАЗМЕРНОСТИ В ДЕКЛАРАТИВНОМ ОПЕРАТОРЕ В ВЫЗЫВАЮЩЕЙ ПРОГРАММЕ. (НА ВХОДЕ)

B —НА ВХОДЕ МАТРИЦА РАЗМЕРОВ N НА M, СОДЕРЖАЩАЯ ПРАВЫЕ ЧАСТИ СИСТЕМЫ AX=B. НА ВЫХОДЕ СОДЕРЖИТ РЕШЕНИЕ X.

IDGT —ВХОДНОЙ РЕЖИМ. ЕСЛИ IDGT БОЛЬШЕ 0, ТО ЭЛЕМЕНТЫ МАТРИЦ A И B ПРЕДПОЛАГАЮТСЯ ВЕРНЫМИ С IDGT ДЕСЯТИЧНЫМИ ЦИФРАМИ, И ПОДПРОГРАММА ВЫПОЛНЯЕТ ПРОВЕРКУ ТОЧНОСТИ. ЕСЛИ IDGT=0, ПРОВЕРКА НА ТОЧНОСТЬ ОПУСКАЕТСЯ.

WKAREA — РАБОЧИЙ МАССИВ ДЛИНЫ, БОЛЬШЕЙ ИЛИ РАВНОЙ N.
IER — ПАРАМЕТР ОШИБКИ. (НА ВЫХОДЕ) ОШИБКА IER=129 УКАЗЫВАЕТ, ЧТО МАТРИЦА А АЛГОРИТМИЧЕСКИ ВЫРОЖДЕНА (СМ. ВВЕДЕНИЕ К ГЛАВЕ L.) ПРЕДУПРЕЖДАЮЩАЯ ОШИБКА IER=34 УКАЗЫВАЕТ, ЧТО ПРОВЕРКА НА ТОЧНОСТЬ ПОТЕРПЕЛА НЕУДАЧУ. ВЫЧИСЛЕННОЕ РЕШЕНИЕ МОЖЕТ ИМЕТЬ ОШИБКУ БОЛЬШУЮ, ЧЕМ ЭТО МОЖЕТ БЫТЬ ОБЪЯСНЕНО НЕОПРЕДЕЛЕННОСТЬЮ ВХОДНЫХ ДАННЫХ. ЭТО ПРЕДУПРЕЖДЕНИЕ МОЖЕТ БЫТЬ СДЕЛАНО ТОЛЬКО ТОГДА, КОГДА IDGT БОЛЬШЕ О НА ВХОДЕ. (СМ. ВВЕДЕНИЕ К ГЛАВЕ L.)

ТОЧНОСТЬ/АППАРАТУРА — ОДИНАРНАЯ И УДВОЕННАЯ/H32
— ОДИНАРНАЯ/H36,H48,H60

ТРЕБУЕМЫЕ ПОДПРОГРАММЫ БИБЛИОТЕКИ IMSL — LUDATF, LUELMF, UERTST, UGETIO

ПРИМЕЧАНИЕ — ИНФОРМАЦИЯ О СПЕЦИАЛЬНЫХ ПРИМЕЧАНИЯХ И СОГЛАШЕНИЯХ ДОСТУПНА ИЗ ВВЕДЕНИЯ К РУКОВОДСТВУ ПО ИСПОЛЬЗОВАНИЮ ИЛИ ИЗ ПОДПРОГРАММЫ UHELP БИБЛИОТЕКИ IMSL.

Алгоритм

LEQT1F решает систему линейных уравнений AX=B с заполненной вещественной матрицей порядка N. Матрица решений порядка N записывается на месте матрицы B. Основное достоинство этой подпрограммы состоит в том, что она требует меньше памяти, чем LEQT2F.

Эта подпрограмма применяет исключение Гаусса (алгоритм Краута) с уравновешиванием и частичным выбором ведущего элемента.

См. книгу:
Forsythe, George and Moler, Cleve B., Computer Solution of Linear Algebraic Systems, Englewood Cliffs, N. J., Prentice-Hall, Inc., 1967, Chapter 9. [Русский перевод: Дж. Форсайт, К. Молер. Численное решение систем линейных алгебраических уравнений.— М.: Мир, 1969.]

Примечание. Эта подпрограмма применяет алгоритм Краута для разложения матрицы (в противоположность подпрограмме исключения в приведенной выше ссылке).

Точность

Если IDGT больше нуля, то предполагается, что элементы матрицы A верны с точностью до IDGT десятичных цифр. Решение X будет точным решением (если не учитывать ошибки округления) системы с матрицей $\bar{A}$, которая согласуется с матрицей A с точностью до первых IDGT десятичных цифр. Если решение не может быть получено с точностью, с которой проводятся вычисления, то выдается

предупреждающее сообщение (IER=34). Если значение IDGT равно нулю, то проверка точности опускается.

Пример

В этом примере решается система линейных уравнений AX=B, где A является матрицей третьего порядка и B — матрица размеров 3 на 4. Поскольку IDGT≠0, то производится проверка на точность

```
ВВОД :

REAL A(4,4),B(4,4),WKAREA(16)
INTEGER M,N,IA,IDGT,IER

N     =   3
M     =   4
IA    =   4
IDGT  =   3

A     =   [ 33.000    16.0      72.0      x ]
          [-24.000   -10.0     -57.0      x ]
          [- 8.000   - 4.0     -17.0      x ]
          [   x        x         x        x ]

B     =   [  1.0       0.0       0.0    -359.0 ]
          [  0.0       1.0       0.0     281.0 ]
          [  0.0       0.0       1.0      85.0 ]
          [   x        x         x         x   ]

 CALL LEQT1F (A,M,N,IA,B,IDGT,WKAREA,IER)
       .
       .
       .
 END

ВЫВОД :

 IER   =   0

 A     =   [ -8.0      -4.0      -17.0      x ]
           [  3.0       2.0       -6.0      x ]
           [ -4.1250   -0.25       0.375    x ]
           [   x        x         x         x ]

 B     =   [ -9.6666   -2.6666   -32.0       1.0 ]
           [  8.0       2.5       25.5      -2.0 ]
           [  2.6666     .6666     9.0      -5.0 ]
           [   x        x         x          x   ]
```

Примечание. Символ x указывает на элементы, не используемые подпрограммой LEQT1F.

ПРИЛОЖЕНИЕ Г

ПОДПРОГРАММЫ РЕШЕНИЯ СОВМЕСТНЫХ ЛИНЕЙНЫХ УРАВНЕНИЙ БИБЛИОТЕКИ NAG

NAG означает название фирмы Numerical Algorithms Group, Ltd., которая поставляет обширную библиотеку математических подпрограмм. Стратегия фирмы состоит в том, что низкая стоимость программного продукта, согласованная с его широким распространением, приведет к коммерческому успеху. Библиотека насчитывает свыше 300 подпрограмм, и ее специальные версии доступны для вычислительных машин Burroughs, CDC, Cray, DEC, Harris, Hewlett-Packard, IBM, ICL, Interdata, Mudcomp, Nord, Philips, Prime, Siemens, Telefunken, Univac, Xerox. Библиотека сдается в аренду приблизительно за 100 долларов в месяц и модифицируется каждые 12—18 месяцев. Доступна версия библиотеки на Алголе-60 и на подмножестве Алгола-68. Дальнейшую информацию о библиотеке и ее составе можно найти в руководстве по использованию библиотеки NAG, опубликованном фирмой.

Г.1. Вводная глава библиотеки NAG

Справочное руководство по использованию любой библиотеки должно быть не требующим дополнительных материалов справочником пользователя. Ниже приводится вводный материал из руководства библиотеки NAG, посвященный совместным линейным уравнениям, чтобы показать подход фирмы к обеспечению пользователя документацией.

F04 — совместные линейные уравнения. Вводная глава
1. Область применения подпрограмм главы

В этой главе речь идет о решении матричного уравнения $AX=B$, где B может быть одним вектором или несколькими правыми частями и матрица A может быть вещественной, комплексной, симметричной, положительно определенной, ленточной, прямоугольной размеров $m \times n (m>n)$ или разреженной.

2. Предварительные сведения о задачах

Система линейных уравнений может быть записана в виде $\mathbf{Ax}=\mathbf{b}$, где известная матрица A с вещественными или комплексными элементами имеет размеры $m \times n$ (m строк и n столбцов), известный вектор правых частей $\mathbf{b}$ имеет m компонент (m строк и один столбец), а искомый вектор решения $\mathbf{x}$ имеет n компонент (n строк и один столбец). Может быть также задано p векторов $\mathbf{b}_i$, $i=1, 2, \ldots, p$, правых частей, и тогда уравнения могут быть записаны в виде $AX=B$, где искомая матрица X имеет своими p столбцами решения

уравнений $A\mathbf{x}_i=\mathbf{b}_i$, $i=1, 2, \ldots, p$. Большинство подпрограмм имеют дело с последним случаем, однако для простоты здесь рассматривается только случай $p=1$.

Наиболее распространенная задача определения единственного решения системы $A\mathbf{x}=\mathbf{b}$ имеет место для $m=n$ и невырожденной матрицы A, т. е. $\text{rank}(A)=n$. Этот случай рассматривается ниже в разд. 2.1. Следующей наиболее распространенной задачей, рассматриваемой ниже в разд. 2.2., является задача определения решения системы $A\mathbf{x}=\mathbf{b}$ в смысле наименьших квадратов при $m>n$ и $\text{rank}(A)=n\neq\text{rank}(A, \mathbf{b})$, где $(A, \mathbf{b})$ — матрица размеров $m\times(n+1)$, образованная окаймлением матрицы A столбцом $\mathbf{b}$. Все другие случаи в некотором смысле «вырождены», и они кратко рассматриваются в разд. 2.3.

2.1. Единственное решение системы $A\mathbf{x}=\mathbf{b}$

Большинство подпрограмм этой главы решает именно эту задачу. Вычисления начинаются треугольным разложением $A=LU$, где L и U соответственно нижняя и верхняя треугольные матрицы. Решение получается последовательным решением более простых систем

$$L\mathbf{y}=\mathbf{b}, \qquad U\mathbf{x}=\mathbf{y}.$$

Первая решается прямой подстановкой, а вторая — обратной.

В частном случае, когда A — вещественная симметричная положительно определенная матрица, $U=L^T$ (имеет место разложение Холесского $A=LL^T$). В других случаях (включая случай, когда A — симметричная, но не положительно определенная матрица) U имеет единичные диагональные элементы (разложение Краута). Если A — ленточная матрица, матрицы L и U имеют соответствующую ширину ленты.

Вследствие ошибок округления вычисленное «решение», скажем, $\mathbf{x}_0$ является только приближением к истинному решению $\mathbf{x}$. Это приближение иногда удовлетворительно и согласуется с $\mathbf{x}$ несколькими цифрами, но если задача плохо обусловлена, то $\mathbf{x}$ и $\mathbf{x}_0$ могут иметь немного или даже вовсе не иметь общих цифр, и в таком случае не имеет смысла оценивать «точность» $\mathbf{x}_0$.

Чтобы получить эту информацию и «исправить» решение $\mathbf{x}_0$, когда оно осмысленно (см. следующий абзац), вычисляется вектор невязки $\mathbf{r}=\mathbf{b}-A\mathbf{x}_0$ с расширенной точностью, и вектор поправки $\mathbf{d}$ получается решением системы $LU\mathbf{d}=\mathbf{r}$. Новое приближенное решение $\mathbf{x}_0+\mathbf{d}$ обычно более точное, и соответствующий процесс повторяется, пока

(а) последующие поправки пренебрежимо малы или

(б) они дальше не уменьшаются.

Необходимо подчеркнуть, что «истинное» решение $\mathbf{x}$ может не быть осмысленным, т. е. правильным с точностью до всех выписанных цифр, если элементы матрицы A и вектора $\mathbf{b}$ известны с опреде-

ленностью до, скажем, p цифр, где p — меньше, чем длина слова вычислительной машины. Тогда первый вектор коррекции $\mathbf{d}$ дает некоторую полезную информацию о числе цифр в «решении», которая, вероятно, останется неизменной по отношению к максимально возможным неопределенностям в элементах. Полезная альтернативная информация получается решением двух систем уравнений, одна из которых имеет заданные элементы, а другая имеет элементы, округленные до $p-1$ цифры. Затем подсчитывается число цифр, с точностью до которых оба решения согласуются. Для плохо обусловленных задач это число может быть удивительно небольшим или даже равным нулю.

2.2. Решение системы $A\mathbf{x}=\mathbf{b}$, $m>n$, $\operatorname{rank}(A)=n$ *в смысле наименьших квадратов*

Решение системы в смысле наименьших квадратов представляет собой вектор $\hat{\mathbf{x}}$, который минимизирует сумму квадратов невязок

$$S=(A\hat{\mathbf{x}}-\mathbf{b})^{\mathrm{T}}(A\hat{\mathbf{x}}-\mathbf{b})=\|A\mathbf{x}-\mathbf{b}\|_2^2.$$

Решение получается на двух этапах:

(1) Применяются преобразования отражения для приведения матрицы A к «более простой форме» посредством разложения $QA=R$, где R имеет вид $\begin{pmatrix}\hat{R}\\0\end{pmatrix}$. Здесь $\hat{R}$ — невырожденная верхняя треугольная матрица порядка n, а 0 — нулевая матрица размеров $(m-n)\times n$. Подобные операции преобразуют вектор $\mathbf{b}$ к виду $Q\mathbf{b}=\mathbf{c}$, где $\mathbf{c}=\begin{pmatrix}\mathbf{c}_1\\\mathbf{c}_2\end{pmatrix}$. Здесь $\mathbf{c}_1$ имеет n строк и $\mathbf{c}_2$—$(m-n)$ строк.

(2) Искомое решение в смысле наименьших квадратов получается обратной подстановкой из системы

$$\hat{R}\hat{x}=\mathbf{c}_1$$

Вследствие ошибок округления вычисленное решение $\hat{x}_0$ является только приближением к искомому решению $\hat{\mathbf{x}}$, однако, как и в разд. 2.1, оно может быть уточнено «итерационным улучшением». Первый вектор коррекции $\mathbf{d}$ представляет собой решение задачи наименьших квадратов

$$A\mathbf{d}=\mathbf{b}-A\hat{\mathbf{x}}_0=\mathbf{r}$$

Поскольку матрица A неизменна, эти вычисления требуют меньше времени, чем вычисление первоначального вектора $\hat{\mathbf{x}}_0$. Процесс может повторяться до тех пор, пока дальнейшие коррекции (а) не станут пренебрежимо малы или (б) не перестанут уменьшаться.

2.3. Особые случаи

(а) Каковы бы ни были размеры $(m\times n)$ матрицы A, уравнения не имеют решения, если $\operatorname{rank}(A, \mathbf{b})$ не будет равен $\operatorname{rank}(A)$. Если эти

ранги равны, то решение единственно только тогда, когда $m \geqslant n$ и rank (A)=n. (Подпрограммы случая 2.1 для m=n проверяют, равняется ли n рангу матрицы A, и если это равенство не выполнено, то передают сообщение об ошибке). Если rank (A)=r<n, то однородные линейные уравнения $\mathbf{Ax}=0$ имеют n—r независимых ненулевых решений $\mathbf{v}_i$.

3. Рекомендации по выбору и применению подпрограмм

3.1. Общее обсуждение

Большинство подпрограмм этой главы решают линейные уравнения $\mathbf{Ax}=\mathbf{b}$, когда матрица A квадратная порядка n и ожидается единственное решение системы (случай 2.1). Если оказывается, что это не так, то подпрограммы осуществляют аварийный выход. Матрица A может быть вещественной или комплексной общего вида или может иметь несколько специальных видов: (а) вещественная симметричная положительно определенная (все собственные значения больше нуля), (б) ленточная со свойствами (а) и (в) вещественная и разреженная. Некоторые подпрограммы вырабатывают только первое приближение, а другие корректируют его методом итерационного улучшения, рассмотренного в разд. 2.

О тех подпрограммах, которые вычисляют только первое приближение, говорят, что они вычисляют «приближенное» решение, тогда как о подпрограммах, которые применяют итерационное улучшение, говорят, что они вычисляют «точное» решение.

Необходимо подчеркнуть, что расточительно в смысле затрат машинного времени и памяти применять неподходящие подпрограммы, например, подпрограмму для комплексного случая, когда заданная система вещественна. Не разумно также применять специальные подпрограммы для положительно определенной матрицы, если это свойство не известно заранее.

Подпрограммы предназначены либо для вычисления приближенного решения, т. е. неоднократного решения линейных уравнений, либо для получения точного решения последовательными итерационными уточнениями этого первого приближения. Последнее, конечно, более дорого в смысле затрат машинного времени и памяти, поскольку каждое уточнение включает в себя решение n линейных уравнений, и исходная матрица A и ее LU-разложение должны размещаться в памяти вместе с первым и последовательно уточненным приближениями к решению. На практике требования на использование памяти «корректирующими» подпрограммами в два раза превосходят требования «аппроксимирующих» подпрограмм, хотя дополнительные затраты на машинное время допустимы, поскольку используется та же самая матрица и то же LU-разложение при каждом решении линейных уравнений.

Несмотря на дополнительные затраты «корректирующих» под-

программ, они имеют подавляющее превосходство над «аппроксимирующими» подпрограммами. Без по крайней мере одной коррекции невозможно дать какую-либо оценку ни числа точных цифр в решении, ни числа «осмысленных» цифр по отношению к степени неопределенности в элементах матрицы (см. разд. 2).

Подпрограммы для вещественных разреженных матриц следует применять только тогда, когда число ненулевых элементов очень мало, менее 10% от числа n^2 элементов матрицы, и матрица не имеет относительно малой ширины ленты. Для случая 2.2, когда $m>n$ и ищется единственное решение в смысле наименьших квадратов, имеются две подпрограммы для вещественной матрицы A, одна из которых вычисляет итерационные подправки. Если $m<n$, то $\text{rank}(A)<n$, и решение в смысле наименьших квадратов не единственно, и подпрограммы осуществляют аварийный выход. И снова, как в случае 2.1, «корректирующая» подпрограмма предпочтительнее во всех отношениях (за исключением времени выполнения и памяти) перед «аппроксимирующей» подпрограммой. Если оказывается, что $\text{rank}(A, \mathbf{b})=\text{rank}(A)=n$, то решение системы $A\mathbf{x}=\mathbf{b}$ в смысле наименьших квадратов единственно, т. е. $\|A\mathbf{x}-\mathbf{b}\|_2^2=0$.

Если подпрограмма для случая 2.2 терпит неудачу из-за того, что $\text{rank}(A)=r<n$, то может быть применена подпрограмма вычисления «решения минимальной длины в смысле наименьших квадратов». В настоящее время имеется только одна подпрограмма для «приближенного» решения. Эта подпрограмма пытается также вычислить ранг r матрицы по заданному допуску, который используется, чтобы принять решение о том, может ли элемент рассматриваться в качестве нулевого. Однако только метод сингулярного разложения, описанный ниже, может дать надежное указание ранга.

Все другие случаи вырожденности (случай 2.3) покрываются одной подпрограммой FO1BHA/F, которая применяет сингулярное разложение матрицы A. Она выдает или вектор решения минимальной длины системы $A\mathbf{x}=\mathbf{b}$, или вектор решения минимальной длины системы в смысле наименьших квадратов (как это поясняется в разд. 2), а также вычисляет векторы $\mathbf{v}_i$, представляющие собой ненулевые решения однородной системы $A\mathbf{x}=0$ (случай $p=0$).

Трудность в применении этой подпрограммы состоит в определении ранга матрицы A ($\text{rank}(A)=r$). Аварийные ситуации в подпрограммах в случаях 2.1 и 2.2 показывают, что $\text{rank}(A)\neq n$, однако они выдают значение r. Ранг теоретически определим при помощи сингулярного разложения, однако в тяжелых случаях ошибки округления могут вызвать непреодолимые трудности.

Подпрограммы этой главы распадаются на две легко определяемые категории:

(1) Подпрограммы черного ящика.

Обычно системы $A\mathbf{x}_i=\mathbf{b}_i$, $i=1, 2, \ldots, p$, могут быть решены, когда p и все $\mathbf{b}_i$ известны заранее. Тогда следует применять подпрограммы

черного ящика, а они будут обращаться к подпрограммам категории (2) и из гл. FO1 и FO3.

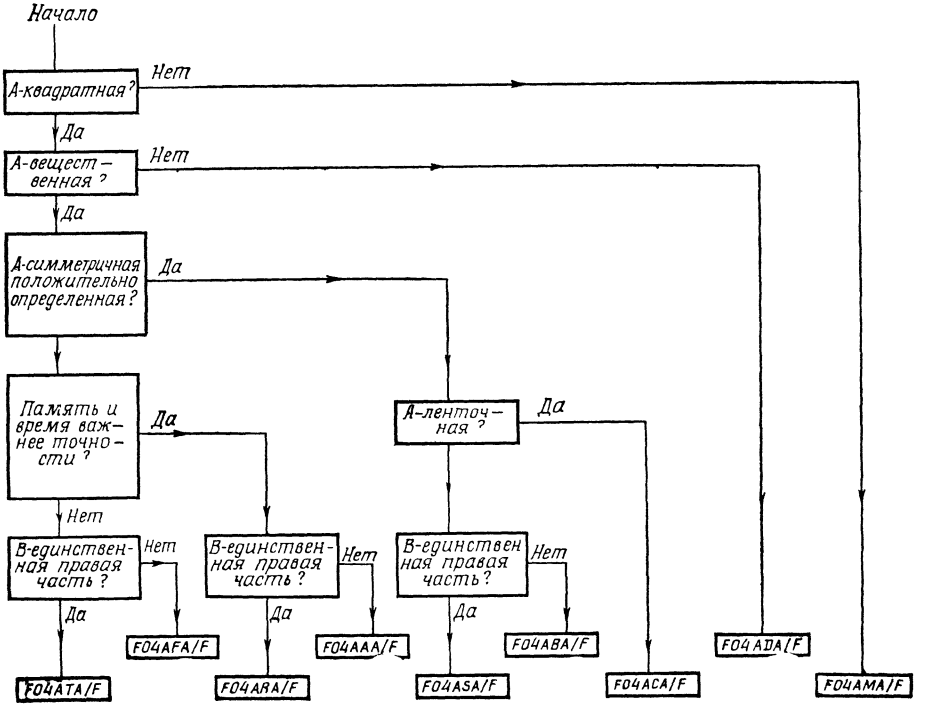
(2) Подпрограммы общего назначения.

В некоторых задачах может так случиться, что число p не известно заранее и в процессе вычислений могут «генерироваться» новые правые части. Поскольку матрица A не меняется, удобно и экономно осуществить ее треугольное разложение только один раз. Это делается подпрограммами из гл. FO3 или FO4, и необходимая информация используется при последовательных обращениях к подпрограммам этой категории.

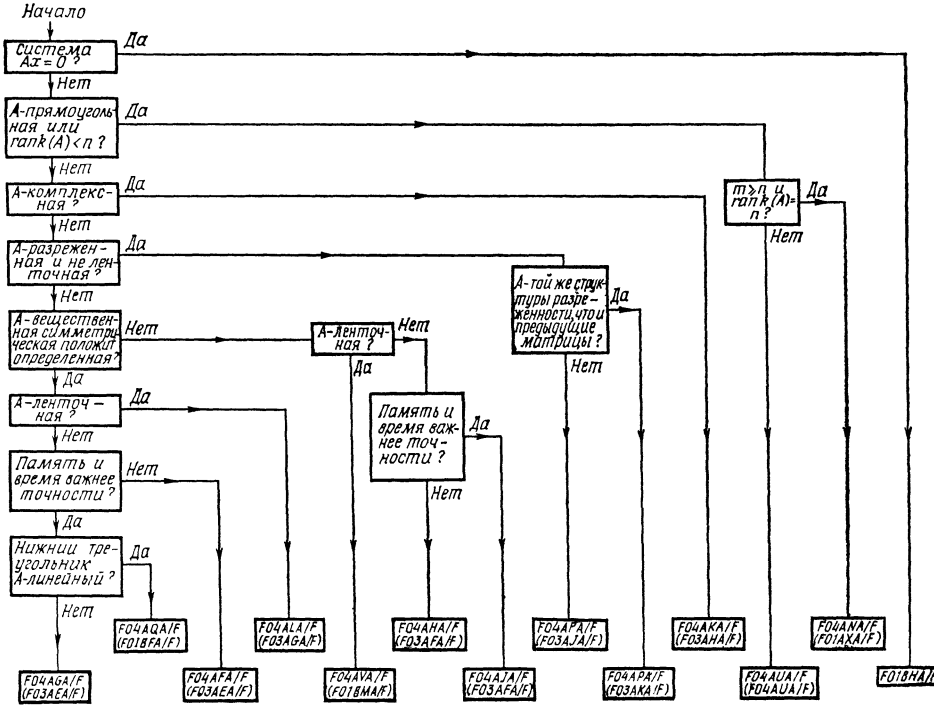
3.3. Деревья решений

Если на любом этапе принятия решений ответом на вопрос является «не знаю», на приводимых ниже блок-схемах следует идти по направлению «нет».

(1) Подпрограммы черного ящика



(2) Подпрограммы общего назначения
Имя подпрограммы, которую следует применять для треугольного разложения матрицы A, приводится в скобках под именем подпрограммы решения системы линейных уравнений.



Содержание главы F04
Важно: навести справку в соответствующем документе о реализации для проверки того, что подпрограмма доступна на требуемом языке и в нужной реализации.

а) *Подпрограммы черного ящика.*
Эти подпрограммы решают матричное уравнение $AX=B$:

Таблица Г.1

Имя подпрограммы	Назначение	Номер издания
F04AAA/F	Приближенное решение системы $AX=B$ с вещественной матрицей и многими правыми частями методом Краута.	2
F04ABA/F	Точное решение системы $AX=B$ с вещественной симметричной положительно определенной матрицей и многими правыми частями методом Холесского	2
F04ACA/F	Приближенное решение системы $AX=B$ с вещественной ленточной симметричной положительно определенной матрицей и многими правыми частями методом Холесского	2
F04ADA/F	Приближенное решение системы $AX=B$ с комплексной матрицей и многими правыми частями методом Краута.	2
F04AEA/F	Точное решение системы $AX=B$ с вещественной матрицей и многими правыми частями методом Краута	2
F04AMA/F	Точное решение в смысле наименьших квадратов системы $AX=B$ m уравнений с n неизвестными, $m \geqslant n$, и многими правыми частями	2
F04ARA/F	Приближенное решение системы $Ax=b$ с вещественной матрицей и единственной правой частью методом Краута.	4
F04ASA/F	Точное решение системы $Ax=b$ с вещественной симметричной положительно определенной матрицей и единственной правой частью методом Холесского.	4
F04ATA/F	Точное решение системы $Ax=b$ с вещественной матрицей и единственной правой частью методом Краута.	4

б) *Подпрограммы общего назначения*

Таблица Г.2

Имя подпрограммы	Назначение	Номер издания
FO4AFA/F	Точное решение системы $AX=B$ с вещественной симметричной положительно определенной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AEA/F.	2
FO4AGA/F	Приближенное решение системы $AX=B$ с вещественной симметричной положительно определенной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AEA/F.	2
FO4AHA/F	Точное решение системы $AX=B$ с вещественной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AFA/F.	2
FO4AJA/F	Приближенное решение системы $AX=B$ с вещественной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AFA/F.	2
FO4AKA/F	Приближенное решение системы $AX=B$ с комплексной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AHA/F.	2
FO4ALA/F	Приближенное решение системы $AX=B$ с вещественной симметричной положительно определенной ленточной матрицей и многими правыми частями, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AGA/F.	2
FO4ANA/F	Приближенное решение в смысле наименьших квадратов системы $AX=B$ m уравнений с n неизвестными, $m \geqslant n$, и многими правыми частями, у которой матрица разложена на треугольные матрицы подпрограммой FO1AXA/F.	2
FO4APA/F	Приближенное решение системы $Ax=b$ или $A^Tx=b$ с вещественной разреженной матрицей и одной правой частью, у которой матрица A разложена на треугольные матрицы подпрограммой FO3AJA/F или FO3AKA/F. (Вычисляется также Ab или A^Tb.)	3
FO4AQA/F	Приближенное решение системы $Ax=b$ с вещественной симметричной положительно определенной матрицей и одной правой частью, у которой матрица A разложена в произведение LDL^T подпрограммой FO1BFA/F. Осуществляется экономия памяти.	3
FO4AUA/F	Вычисление приближенного минимального решения в смысле наименьших квадратов системы $AX=B$ с матрицей размеров $m \times n$ ($m > n$, $\text{rank}(A) \leqslant n$) и многими правыми частями, у которой матрица A факторизована подпрограммой FO1BKA/F.	5

Продолжение табл. Г.2

Имя подпрограммы	Назначение	Номер издания
FO4AVA/F	Приближенное решение системы AX=B с вещественной ленточной матрицей и многими правыми частями, у которой матрица разложена на треугольные матрицы подпрограммой FO1BMA/F.	6

Примечание. Окончание имен подпрограмм на A/F указывает на то, что подпрограмма составлена или на Алголе-60 (окончание A), или на Фортране (окончание F); например, AO2AAA/F означает, что на Алголе-60 подпрограмма имеет имя AO2AAA, а на Фортране — AO2AAF.

1. Назначение

FO4ARF вычисляет приближенное решение системы $\mathbf{Ax}=\mathbf{b}$ с вещественной матрицей и одной правой частью методом Краута.
Важно: перед применением этой подпрограммы ознакомиться с документацией о соответствующей машинной реализации для контроля правильности толкования выделенных курсивом терминов и других зависящих от реализации деталей.

2. Спецификация (Фортран IV)

```
      SUBROUTINE FO4ARF (A, IA, B, N, C, WKSPCE, IFAIL)
C     INTEGER IA, N, IFAIL
C     real A (IA, N), B (N), C (N), WKSPCE (N)
```

3. Описание

Для заданной системы линейных уравнений $\mathbf{Ax}=\mathbf{b}$ подпрограмма сначала факторизует матрицу A к виду PA=LU методом Краута с частичным выбором ведущего элемента, где P — матрица перестановок, L — нижняя треугольная и U — верхняя треугольная матрицы. Приближенное решение $\mathbf{x}$ находится прямой и обратной подстановками из систем $\mathbf{Ly}=\mathbf{Pb}$ и $\mathbf{Ux}=\mathbf{y}$, где $\mathbf{b}$ — вектор правой части системы. В процессе вычислений применяется накопление *дополнительной точности* для скалярных произведений.

4. *Литература*

[1] Wilkinson J. H. and Reinsch C. Handbook for Automatic Computation. Volume II, Linear Algebra, Springer-Verlag, 1971, pp. 93—110. [Русский перевод: Уилкинсон, Райнш. Справочник алгоритмов на языке Алгол. Линейная алгебра.— М.: Машиностроение, 1976.]

5. Параметры

A — *вещественный* массив размеров (IA, p), где p≥N. На входе A содержит элементы вещественной матрицы системы. На выходе A содержит разложение Краута с единичной диагональю у матрицы U.

IA — INTEGER.
На входе IA задает первую размерность массива A в декларативном операторе вызывающей (под)программе (IA≥N). На выходе не меняется.

B — *вещественный* массив длины по крайней мере N. На входе B содержит элементы вектора правой части. На выходе не меняется, однако см. разд. 11.

N — INTEGER.
На входе N задает порядок матрицы A.
На выходе не меняется.

C — *вещественный* массив длины по крайней мере N. На выходе C содержит вектор решения.

WKSPCE — *вещественный* массив длины по крайней мере N. Используется в качестве рабочего.

IFAIL — INTEGER.
На входе значение IFAIL должно быть определено. Для пользователей, не знакомых с этим параметром (описанным в гл. P01), рекомендуемое значение равно 0. Если подпрограмма не обнаружит ошибку (см. разд. 6), на выходе IFAIL содержит 0.

6. Указатели ошибок

Ошибки, обнаруживаемые подпрограммой:
IFAIL=1— матрица A вырождена, возможно, вследствие ошибок округления.

7. Дополнительные подпрограммы

Подпрограмма обращается к подпрограммам F03AFF, F04AJF, P01AAF и X02AAF из библиотеки NAG.

8. Время исполнения

Требуемое время исполнения приблизительно пропорционально N^3.

9. Память

Массивов, объявленных внутри подпрограммы, нет.

10. Точность.

Точность вычисленного решения зависит от обусловленности исходной матрицы. Детальный анализ ошибок см. в [1], стр. 107.

11. Дальнейшие комментарии

Если подпрограмма вызывается с одним и тем же идентификатором для параметров B и C, то вектор решения записывается на место правой части.

12. Ключевые слова

Приближенное решение линейных уравнений.
Факторизация Краута.
Вещественная матрица.
Единственная правая часть

13. Пример

Требуется решить систему линейных уравнений $A\mathbf{x}=\mathbf{b}$, где

$$A=\begin{pmatrix} 33 & 16 & 72 \\ -24 & -10 & -57 \\ -8 & -4 & -17 \end{pmatrix} \quad \text{и} \quad \mathbf{b}=\begin{pmatrix} -359 \\ 281 \\ 85 \end{pmatrix}.$$

Программа

Этот пример программы с одинарной точностью может потребовать поправок
1) для использования в реализации с удвоенной точностью,
2) для использования и с той и с другой точностью в определенных реализациях.

Полученные результаты могут слегка отличаться.

```
C     F04ARF EXAMPLE PROGRAM TEXT
C     NAG COPYRIGHT 1975
C     MARK 4.5 REVISED
C
      REAL A(4,4), B(6), C(6), WKS(18)
      INTEGER NIN, NOUT, I, N, J, IA, IFAIL
      DATA NIN /5/, NOUT /6/
      READ (NIN,99999) (WKS(I),I=1,7)
      WRITE (NOUT,99997) (WKS(I),I=1,6)
      N = 3
      READ (NIN,99998) ((A(I,J),J=1,N),I=1,N), (B(I),I=1,N)
      IA = 4
      IFAIL = 1
      CALL F04ARF(A, IA, B, N, C, WKS, IFAIL)
      IF (IFAIL.EQ.0) GO TO 20
      WRITE (NOUT,99996) IFAIL
      STOP
   20 WRITE (NOUT,99995) (C(I),I=1,N)
      STOP
99999 FORMAT (6A4, 1A3)
99998 FORMAT (3F5.0)
99997 FORMAT (4(1X/), 1H , 5A4, 1A3, 7HRESULTS/1X)
99996 FORMAT (25HOERROR IN F04ARF IFAIL = , I2)
99995 FORMAT (10HOSOLUTIONS/(1H , F4.1))
      END
```

```
Данные
F04ARF
   33   16   72
  -24  -10  -57
   -8   -4  -17
 -359  281   85


Результаты
F04ARF

Решения
 1.0
-2.0
-5.0
```

ПРИЛОЖЕНИЕ Д

ПОДПРОГРАММЫ ДЛЯ МЕТОДА НАИМЕНЬШИХ КВАДРАТОВ ИЗ КНИГИ ЛОУСОНА И ХЭНСОНА

В Приложении Д приводится документация и список подпрограмм на Фортране из книги Лоусона и Хэнсона (1974).

Эти подпрограммы применимы для большинства линейных или нелинейных задач наименьших квадратов Основные идеи метода наименьших квадратов представлены в гл. 11 настоящей книги, и, как мы отметили там, любому, кто решает серьезные задачи наименьших квадратов, следует ознакомиться с книгой Лоусона и Хэнсона.

Подпрограммы работают под управлением шести главных программ, которые спроектированы так, чтобы демонстрировать различные алгоритмы и применение программных модулей. Организация подпрограмм приводится на рис. Д 1.

Указатель основных программ Лоусона и Хэнсона представлен в табл. Д. 1, а используемых подпрограмм — в табл. Д. 2. Этот набор включает в себя шесть главных программ (ведущих программ), один набор данных и 14 основных модулей. Определения даются

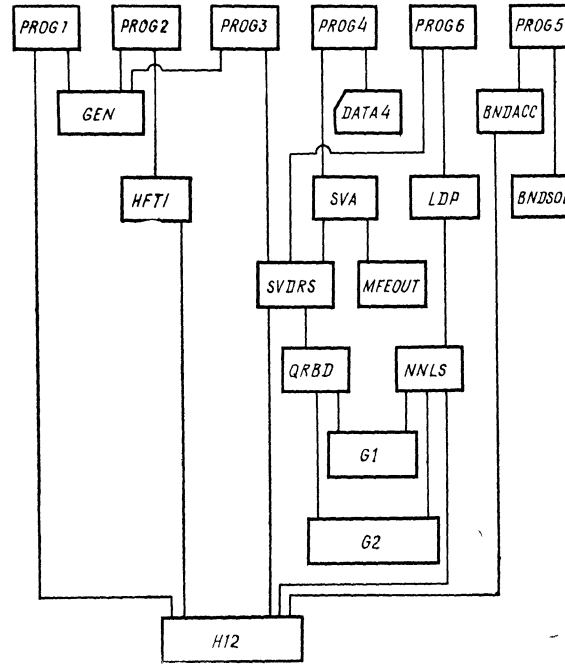


Рис. Д.1. Организация подпрограмм Лоусона и Хэнсона Кроме того, DIFF обращается к HFTI, ORBD, LDP и NNLS.

через имена, введенные в книге Лоусона и Хэнсона (1974). Независимое представление материала могло бы быть гораздо длиннее, и в любом случае пользователю этих программ следует иметь доступ к той книге.

Приводимое ниже руководство по использованию модуля HFTI иллюстрирует подход, применяемый авторами для документирования подпрограмм. Это руководство не включается в начальные комментарии рассмотренных подпрограмм.

Руководство по использованию подпрограммы HFTI: решение задачи наименьших квадратов при помощи преобразований отражения

Вызываемые подпрограммы H12, DIFF
Назначение

Эта подпрограмма решает линейную задачу наименьших квадратов или набор линейных задач наименьших квадратов, имеющих одну и ту же матрицу, но различные векторы правых частей. Данные за-

дачи представляют собой матрицу A размеров M×N, матрицу B размеров M×NB и параметр абсолютного допуска τ. NB векторов-столбцов матрицы B образуют векторы $\mathbf{b}_j$ правых частей для NB различных задач наименьших квадратов.

$$A\mathbf{x}_j \approx \mathbf{b}_j,\ j=1, 2, \ldots, NB.$$

Этот набор задач может быть также записан в виде матричной задачи наименьших квадратов:

$$AX \approx B,$$

где X — матрица размеров N×NB, имеющая своими столбцами векторы $\mathbf{x}_j$.

Таблица Д.1

Указатель главных программ и элемента данных DATA4

Имя программы	Номер стр текста на Фортране	Назначение программы
PROG1	279	Демонстрирует алгоритмы HFT, HS1 и COV из гл. 11 и 12 Обращается к подпрограммам H12 и GEN.
PROG2	281	Демонстрирует алгоритмы HFTI и COV из гл. 14 и 12 соответственно. Обращается к подпрограммам HFTI и GEN
PROG3	283	Демонстрирует алгоритм сингулярного разложения из гл. 18. Обращается к подпрограммам SVDRS и GEN.
PROG4	285	Демонстрирует сингулярный анализ, включая вычисление норм решения Левенберга-Марквардта и нормы невязки, из гл. 18, 25 и 26. Обращается к подпрограмме SVA и вводит элемент данных DATA4.
DATA4	285	Это не программа, а набор из 15 образцов перфокарт, содержащих данные для ввода программой PROG4 Этот пример обсуждается в гл. 26.
PROG5	286	Демонстрирует ленточно-ограниченный алгоритм последовательного накопления из гл 27, разд. 2 и 4. Алгоритм применяется для выравнивания данных кубическим сплайном (на равномерной сетке) таблицы данных. Обращается к подпрограммам BNDACC и BNDSOL
PROG6	288	Решает задачу линейного выравнивания с ограничением, данную в качестве примера в гл. 23. Программа иллюстрирует типичное применение подпрограммы LDP, которая в свою очередь применяет MNLS PROG6 обращается также к SVDRS.

Таблица Д.2

Указатель подпрограмм

Имя подпрограммы	Номер стр. руководства пользователя	Номер стр. текста на Фортране	Назначение подпрограмм
HFTI	254	290	Реализует алгоритм HFTI из гл. 14. Обращается к подпрограммам H12 и DIFF.
SVA	256	292	Реализует сингулярный анализ и анализ Левенберга-Марквардта из гл. 18 и 25. Печатает интересующие величины. Обращается к подпрограммам SVDRS MFEOUT
SVDRS	260	295	Вычисляет сингулярное разложение из гл. 18. Обращается к подпрограммам H12 и QRBD.
QRBD	262	298	Вычисляет сингулярное разложение двудиагональной матрицы из гл. 18. Обращается к подпрограммам G1, G2 и DIFF.
BNDASS и BNDSOL	264	301 и 302	Реализует ленточно-ограниченный алгоритм последовательного накопления из гл. 27, разд. 2. Обращается к подпрограмме H12.
LDP	267	303	Решает задачу наименьшего отклонения из гл 23. Обращается к подпрограммам NNLS и DIFF.
NNLS	269	304	Вычисляет решение в смысле наименьших квадратов при условии, что все переменные неотрицательны, как это описано в гл. 23. Обращается к подпрограммам H12, G1, G2 и DIFF
H12	271	308	Строит и применяет преобразование Хаусхолдера из гл. 10.
G1 и G2	274	309	Строит и применяет вращение Гивенса из гл 10
MFEOUT	275	310	Печатает двумерный массив в одном из двух удобных форматов.
GEN	277	311	Генерирует последовательность чисел для использования при конструировании тестовых данных. Применяется подпрограммами PROG1, PROG2 и PROG3.
DIFF	278	311	Вычисляет разность между двумя аргументами, представленными с плавающей точкой.

Заметим, что если B — единичная матрица размеров M×M, то X представляет собой матрицу, псевдообратную к A.

Метод

Эта подпрограмма сначала преобразует расширенную матрицу [A : B] к матрице [R : C] умножением слева на матрицы преобразований отражения с перестановками столбцов. Все поддиагональные элементы в матрице R равны нулю, а ее диагональные элементы удовлетворяют неравенству $|r_{ii}| \geqslant |r_{i+1,\, i+1}|$, $i=1, 2, \ldots, l-1$, где $l=\min(M, N)$.

Подпрограмма устанавливает псевдоранг системы KRANK равным числу диагональных элементов матрицы R, превосходящих τ. Будет вычислена минимальная длина векторов решения $\hat{\mathbf{x}}_j$, $j=1, 2, \ldots,$ NB, для задач, определенных первыми KRANK строками матрицы [R : C].

Если относительная неопределенность в матрице данных B равна ρ, то предлагается устанавливать значение τ приблизительно равным $\rho \| A \|$.

Дальнейшие детали алгоритма см. в «Алгоритм HFTI», гл. 14.

Использование

DIMENSION A (MDA, n_1), [B (MDB, n_2) или B (m_1)],
RNORM (n_2), H (n_1), G (n_1)
INTEGER IP (n_1)
CALL HFTI (A, MDA, M, N, B, MDB, NB, TAU, KRANK,
RNORM, H, G, IP)

Параметры размерностей должны удовлетворять неравенствам MDA$\geqslant$M, $n_1 \geqslant$N, MDB$\geqslant$ max (M, N), $m_1 \geqslant$ max (M, N), $n_2 \geqslant$NB.

Параметры подпрограммы определяются следующим образом:

A (,), MDA, M, N — В начале массив A (,) содержит M×N — матрицу A задачи наименьших квадратов AX$\approx$B. Первый параметр размерности массива A (,) равен MDA, который должен удовлетворять неравенству MDA$\geqslant$M. Допустимо как M$\geqslant$N, так и M<N. Не существует ограничения на ранг матрицы A. Содержимое массива A (,) будет изменено подпрограммой. См. рис. 14.1, иллюстрирующий пример окончательного содержимого массива A (,).

B( ), MDB, NB — Если NB=0, то подпрограмма не будет использовать массив B. Если NB>0, то вначале массив B ( ) должен содержать

	M×NB-матрицу B задачи наименьших квадратов Ax≈B, а на выходе массив B будет содержать N×NB-матрицу решения X. Если NB≥2, массив B должен быть двумерным с первым параметром размерности MDB≥max (M, N). Если NB=1, то массив B ( ) может быть или двумерным, или одномерным. В последнем случае значение MDB произвольно, однако некоторые компиляторы с Фортрана требуют, чтобы MDB был задан каким-либо целым значением, скажем MDB=1.
TAU	Абсолютный параметр допуска, задаваемый пользователем для определения псевдоранга.
KRANK	Устанавливается подпрограммой равным псевдорангу матрицы A.
RNORM( )	На выходе RNORM (J) будет содержать евклидову норму вектора невязки для задачи, определенной j-м вектором-столбцом массива B (,), j=1, 2, . . ., NB.
H( ), G( )	Рабочие массивы. См. рис. 14.1, иллюстрирующий пример окончательного содержимого этих массивов.
IP( )	Массив, в который подпрограмма записывает индексы, описывающие перестановки векторов-столбцов. См. рис. 14.1, иллюстрирующий окончательное содержимое этого массива.

Пример использования

См. PROG2 как пример использования этой подпрограммы.

ПРИЛОЖЕНИЕ E

EISPACK — ПОДПРОГРАММЫ РЕШЕНИЯ АЛГЕБРАИЧЕСКОЙ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ

Задача вычисления собственных значений и собственных векторов матрицы гораздо труднее задачи решения системы линейных уравнений. В действительности вплоть до 1960-х годов, когда был открыт QR-алгоритм, не было известно ни одного надежного (численно устойчивого) метода. Анализ и реализация хороших алгорит-

мов вычисления собственных значений обычно требует усилий специалиста по численному анализу, если не специалиста по матричному исчислению. Пакет EISPACK представляет собой первый пример того, как следует воплощать знания экспертов в вычислительных программах для последующего их широкого применения.

Подпрограммы пакета созданы на основе набора алгольных процедур, разработанных Уилкинсоном и Райншем в конце 1960-х годов. Использование языка Алгол для этих процедур ограничивало их применение и поэтому был разработан проект реализации этих процедур на Фортране с целью разработки высокоэффективных версий для различных машин, их проверки и вообще, с целью разработки высококачественного пакета программ. Главными разработчиками этого проекта (получившего наименование NATS—National Activity to Test Software) были Б. Т. Смит, Дж. М. Бойл, Дж. Дж. Донгарра, Б. С. Гарбоу, И Икебе, В. К. Клема и К. Б. Молер. Подпрограммы пакета EISPACK представлены и документированы в работе (Smith et al., 1976) и (Garbow et al., 1977).

Таблица Е.1

Основные случаи задач на собственные значения

Виды матриц / Классификация задач	Комлексная общего вида	Комплексная эрмитова	Вещественная общего вида	Вещественная симметричная	Вещественная симметричная трехдиагональная	Специальная вещественная трехдиагональная
Все собственные значения и соответствующие собственные векторы	XX	XX	XX	XX	XX	XX
Все собственные значения	XX	XX	XX	XX	XX	XX
Все собственные значения и выборочные собственные векторы	XX		XX			
Некоторые собственные значения и соответствующие собственные векторы		XX		XX	XX	XX
Некоторые собственные значения		XX		XX	XX	XX

Символами XX обозначены рекомендуемые основные цепочки подпрограмм, описанные в руководстве пакета EISPACK.

Примененный подход состоял в создании систематизированного набора подпрограмм, в котором каждый модуль реализует отдельную фазу процесса решения. Эти модули собираются различными способами для проведения конкретных вычислений в задаче на собственные значения. Возникает много различных случаев; первый уровень схемы организации пакета показан в табл. Е. 1

В таблице даны рекомендуемые основные цепочки подпрограмм пакета для 22 случаев из 30. В некоторых случаях проводится дальнейшее углубление схемы организации пакета, основанное на других характеристиках матрицы (например, ее порядок). Большое число модулей в пакете привело к необходимости (по крайней мере для типичного пользователя) собирать эти основные цепочки в отдельные подпрограммы для наиболее распространенных случаев. Опытный пользователь может успешно собрать модули для обработки почти любой конкретной матрицы. Управляющая программа была составлена для IBM OS/360—370, которая в действительности представляет собой проблемно-ориентированный язык специального назначения для решения алгебраической проблемы собственных значений.

ПРИЛОЖЕНИЕ Ж

МЕСТНАЯ БИБЛИОТЕКА ВЫЧИСЛИТЕЛЬНОГО ЦЕНТРА ПРИМЕР УНИВЕРСИТЕТА ПЭРДЬЮ

Типичный вычислительный центр приобретает программы из различных источников. Эти источники включают в себя:

1. Программы, разработанные в самом вычислительном центре.
2. Программы из других организаций, получаемые через друзей, поиском через устные расспросы и другими неформальными способами.
3. Программы, опубликованные в научной литературе; например, пакет BLAS и программы наименьших квадратов Лоусона и Хэнсона.
4. Наборы программ (обычно ориентированные на одну область применения), доступные из некоммерческих источников, как например из правительственных источников (EISPACK или LINPACK) или от групп пользователей (группы SHARE пользователей IBM).
5. Общие библиотеки из коммерческих источников, как, например, фирмы IMSL, Inc. или NAG, Ltd.
6. Одно время все основные производители машин поставляли бесплатно значительные библиотеки программ вместе со своими вычислительными машинами. Теперь библиотеки продаются

отдельно; библиотека SL-MATH фирмы IBM все еще широко доступна.

7. Специализированные прикладные программы из различных источников.

Программы матричного исчисления широко используются, и сотни таких программ доступны из всех этих источников. Более того, многие составляют свои собственные программы матричного исчисления, исходя из рассуждений (обычно ошибочных), что таким способом они могут получить нужную программу быстрее.

Мы исследуем программы решения линейных уравнений в вычислительном центре Университета Пэрдью как пример того, что имеется в распоряжении пользователя в хорошей библиотеке вычислительного центра. Вычислительный центр имеет библиотеку IMSL, пакет LINPACK плюс семь разработанных своими силами программ. Четыре из них похожи друг на друга: LINEQ1, LINEQ2, LINEQ3 и LINEQ4. Инструкция к подпрограмме LINEQ1 приводится ниже. LINEQ2 предназначена для систем с удвоенной точностью, а LINEQ4 — для комплексных систем. LINEQ3 является версией LINEQ2 и использует тройную точность для итерационного уточнения. Эти подпрограммы имеют две машинно-зависимые особенности: внутренний цикл исключения выполняется на языке ассемблера и применяется динамическое распределение памяти для генерации рабочих массивов, необходимых для алгоритма. Они используют также преимущество того свойства компилятора с распространенного, но не стандартного Фортрана, которое позволяет избегать передачи в подпрограммы второй размерности массивов. Внутри подпрограмм массивы объявлены так:

```
REAL A (N, 1), B (N, 1), X (N, 1)
```

Это не приводит к фиксации ошибки (благодаря схеме распределения памяти для массивов Фортрана), хотя такой способ объявления массивов является причиной выхода индекса за объявленные границы.

Имеются две программы для специальных матриц: SYMEQ1 для симметричных матриц и GELB для ленточных матриц (не обязательно с одной и той же шириной ленты по обе стороны диагонали). Ниже также приводится инструкция к подпрограмме GELB. Наконец, имеется необычная подпрограмма BOUNDS, которая очень точно реализует оценки ошибок Эйрда — Линча (рассмотренные в гл. 9). Подпрограмма применяет интервальную арифметику и удвоенную точность для того, чтобы получить очень точно истинные границы ошибок. Инструкция для BOUNDS представляет собой завершающий пример.

Библиотека вычислительного центра имеет одно распространенное досадное свойство: есть слишком много альтернатив для

одного и того же приложения. Существует около 100 программ решения линейных уравнений, среди которых надо делать выбор: 12 из библиотеки IMSL, 65 из пакета LINPACK, 6 местных подпрограмм плюс многие подпрограммы, содержащиеся в таких пакетах или системах, как SPSS, SOUPAC, BMD и др. Несмотря на то что многие из них не применимы к одной и той же задаче, пользователям трудно решить, какая подпрограмма наиболее соответствует их задаче. Результатом часто является почти случайный выбор.

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР УНИВЕРСИТЕТА ПЭРДЬЮ
ПОДПРОГРАММА LINEQ1
СИСТЕМНАЯ БИБЛИОТЕКА.
Назначение:

(1) Решает вещественное матричное уравнение $AX=B$ с NR правыми частями.
(2) Вычисляет матрицу, обратную к вещественной матрице A.
(3) Вычисляет вещественное матричное выражение $A^{-1}B$.

Использование: CALL LINEQ1 (A, B, X, ND, N, NR, S)
Описание параметров:

A — вещественная матрица коэффициентов размеров $(N\times N)$.
B — вещественный массив правых частей размеров $(N\times NR)$.
X — вещественный массив размеров $(N\times NR)$ вычисленных векторов решения.
ND — число строк массивов A, B и X в декларативном операторе программы пользователя.
N — число решаемых уравнений.
NR — число правых частей.
S — целая переменная, принимающая ненулевое значение только тогда, когда матрица A — вырожденная с машинной точностью.

Эта подпрограмма может быть использована для обращения матрицы A посредством решения системы уравнений $AX=B$, где B — единичная матрица размеров $(N\times N)$. В этом случае $NR=N$ и обратная матрица помещается в X. Единичная матрица размеров $(N\times N)$ может быть образована следующими операторами Фортрана:

```
      DO 2 I=1,N
      DO 1 J=1,N
1     B(I,J)=0.0
2     B(I,I)=1.0
```

Матричное выражение $X=A^{-1}B$ может быть вычислено непосредственно, так как X является решением матричного уравнения $AX=B$. Этот метод дает более точный ответ быстрее, чем он может быть получен вычислением A^{-1} и последующим умножением A^{-1} на B.

Примечания: Массивы A и B не портятся подпрограммой. В процессе исполнения длина вашей программы будет увеличена этой подпрограммой для хранения необходимых временных переменных.

Метод: Матрица A факторизуется на нижнюю и верхнюю треугольные матрицы L и U, и затем последовательно решаются системы LZ=B и UX=Z. Применяется накопление скалярных произведений с удвоенной точностью и итерационное уточнение, поэтому решение очень точное всякий раз, когда S на выходе равно нулю.

Ralston and Wilf, Mathematical Methods for Digital Computers, Volume 2, Wiley, 1967

Составлено Дэвидом С. Додсоном.

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР УНИВЕРСИТЕТА ПЭРДЬЮ

ПОДПРОГРАММА GELB.

Назначение: Решение системы совместных линейных уравнений с матрицей коэффициентов ленточной структуры.

Использование: CALL GELB (R, A, M, N, MUD, MLD, EPS, IER)

Описание параметров:

R — матрица правых частей размеров M на N (уничтожается). На выходе R содержит решение уравнений.

A — вектор длины MA, который содержит ленту матрицы коэффициентов, хранящуюся по строкам в первых последовательных словах памяти (см. примечания ниже).

M — число уравнений системы.

N — число векторов правых частей.

MUD — число верхних кодиагоналей (т. е. кодиагоналей выше главной диагонали).

MLD — число нижних кодиагоналей (т. е. кодиагоналей ниже главной диагонали).

EPS — входная константа, которая используется как относительный допуск для проверки потери значимости.

IER — результирующий параметр ошибки, кодированный следующим образом:
IER=0 — нет ошибки,
IER=—1 — результат не получен из-за неверных входных параметров M, MUD, MLD или из-за того, что ведущий элемент на любом шаге исключения равен 0.

IER=K — предупреждение о возможной потере значимости, обнаруженной на K+1 шаге исключения, на котором ведущий элемент был меньше или равен внутреннему

допуску EPS, умноженному на наибольший по абсолютной величине элемент матрицы A.

Примечания: Предполагается, что лента матрицы коэффициентов, шириной MUD+MLD+1, располагается в памяти по строкам в первых ME последовательных словах памяти из общего числа MA требуемых слов, где

$MA = M * MC - ML * (ML + 1)/2$, $ME = MA - MU * (MU + 1)/2$,
$MC = MIN(M, 1 + MUD + MLD)$, $ML = MC - 1 - MLD$,
$MU = MC - 1 - MUD$

Предполагается, что матрица R правых частей располагается в памяти по столбцам в N×M последовательных словах памяти. На выходе матрица решений R также располагается в памяти по столбцам.

Входные параметры M, MUD, MLD должны удовлетворять следующим ограничениям

MUD не меньше нуля,
MLD не меньше нуля,
MUD + MLD не больше 2 * M — 2.

Если эти ограничения не удовлетворяются, вычисления не производятся и IER полагается равным —1.

Рекомендуется выбирать относительный допуск IER между 0.00001 и 0.000001.

Подпрограмма выдает результат, если удовлетворены ограничения на входные параметры и если ведущие элементы на всех шагах исключения отличны от 0. Однако предупреждение IER=K, если оно выдается, указывает на возможную потерю значимости. В случае хорошо масштабированной матрицы и при подходящем допуске EPS, IER=K может быть объяснено тем, что матрица A имеет ранг K. Если матрица A не имеет нижних кодиагоналей, то подпрограмма не выдает предупреждающего сообщения.

Метод: Решение вычисляется исключением Гаусса только с перестановкой столбцов, чтобы сохранить ленточную структуру в оставшихся коэффициентах матрицы.

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР УНИВЕРСИТЕТА ПЭРДЬЮ

НАЗВАНИЕ: BOUNDS

Авторы: T. J. Aird, Robert E. Lynch.

Назначение: Вычисление границ ошибки решений систем линейных уравнений. Границы ошибок приближенного решения X вычисляются для специфицированных неопределенностей элементов как в матрице коэффициентов A, так и в правой части B.

Использование: CALL BOUNDS (AA, BB, X, A, ND, N, NR, BND)

В приведенных ниже рассуждениях X рассматривается как

приближенное решение системы линейных уравнений с матрицей коэффициентов A и правой частью B. Предполагается также, что элементы в A и B получены предшествующими вычислениями или при помощи экспериментальных измерений физических данных. Поэтому они подвергаются воздействию определенных ошибок. Задача состоит в определении влияния, которое ошибки оказывают на решение.

Границы ошибок вычисляются из расчета максимально возможного отклонения между X и решением любой системы с матрицей коэффициентов AO, которая удовлетворяет неравенствам:

$$AA(1, I, J) \leqslant AO(I, J) \leqslant AA(2, I, J)$$

для I, J=1, 2, . . ., N, и матрицей правых частей BO, которая удовлетворяет неравенствам:

$$BB(1, I, K) \leqslant BO(I, K) \leqslant BB(2, I, K)$$

для I=1, 2, . . ., N и K=1, 2, . . ., NR.
В этом случае говорят, что AO и BO удовлетворяют условиям, заданным матрицами AA и BB соответственно. Более точно, BND (1, K) и BND (2, K) вычисляются для K=1, 2, . . ., NR так, что неравенства

$$BND(1, K) \leqslant MAX\,[ABS(T(I, K) - X(I, K)),\ I=1, 2, \ldots, N] \leqslant BND(2, K)$$

выполняются для решения T системы AO*T=BO всякий раз, когда AO и BO удовлетворяют условиям, заданным матрицами AA и BB соответственно.

Описание параметров:

AA — матрица размеров (2×N×N), которая задает неопределенности в A.

BB — матрица размеров (2×N×NR), которая задает неопределенности в B.

X — матрица размеров (N×NR), которая содержит NR приближенных решений X(.,1), . . ., X(., NR), соответствующих NR правым частям B(.,1), . . ., B(., NR)

A — матрица коэффициентов размеров (N×N). Матрица A удовлетворяет неравенствам
AA(1, I, J)≤A(I, J)≤AA(2, I, J).
A уничтожается в процессе вычислений.

ND — граница размерности для AA, BB, X и A.
AA должна иметь границы размерностей (2, ND, N).
BB должна иметь границы размерностей (2, ND, NR).
X должна иметь границы размерностей (ND, NR).
A должна иметь границы размерностей (ND, N).

N — число уравнений.

NR — число правых частей.
BND — матрица размеров (2×NR), в которую помещаются вычисленные верхняя и нижняя границы.

Примечания:

1) Интервалы для AA и BB могут быть также заданы в обратном порядке. Это означает, что для любых I, J, K допустимы неравенства AA (1, I, J)≥ AA (2, I, J) или BB (1, I, K)≥BB (2, I, K).

2) Обычно матрица A используется для вычисления приближенного решения X и затем используется для вычисления элементов матрицы AA.

Например,

$$AA(1, I, J) = A(I, J) * (1 - 1.0E-8),$$
$$AA(2, I, J) = A(I, J) * (1 + 1.0E-8)$$

будет означать, что A (I, J) известна с 8 цифрами. То же самое имеет место для B и BB.

3) BND (1, K)=BND (2, K)=—1.0 указывает, что границы ошибок не могут быть вычислены подпрограммой. Когда это условие имеет место, то или матрица была определена как вырожденная, или матрица C (приближенно равная обратной к матрице A) не удовлетворяет основному неравенству:

$$NORM(I - C*AO) < 1$$

для всех матриц AO, которые удовлетворяют условиям, заданным матрицей AA.

4) Ошибки округления принимаются во внимание в процессе вычислений границ ошибок.

Метод:

Основная формула границы ошибок имеет вид

$$NORM(C*R)/(1+D) \leqslant NORM(ERROR) \leqslant NORM(C*R)/(1-D)$$

где C — приближенная обратная к A матрица, которая должна удовлетворять неравенству

$$D = NORM(I - C*AO) < 1$$

и R=BO—AO*X. В подпрограмме используется норма MAX. Применяется интервальная арифметика для вычисления интервалов, которые содержат величины в формуле границы ошибок. Таким образом, верхняя и нижняя границы вычисляются для NORM (C*R) на основе AA и BB, а верхняя граница вычисляется для D на основе AA.

Литература

Aird T. J. and R. E. Lynch (1975). Computable accurate upper and lower error bounds for approximate solutions of linear algebra systems. *ACM Trans Math. Software*, **1**, pp. 217—231.

Bjorck A. (1967). Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT*, **7**, pp. 1—21.

Blue J. L. (1975). Automatic numerical quadrature—DQUAD. *Bell Laboratories Report*, Murray Hill, N.J.

deBoor C. W. (1971). CADRE: An algorithm for numerical quadrature, in J. Rice (ed.), Mathematical Software, Academic Press, New York, pp. 417—450.

Dongarra J. J., J. R. Bunch, C. B. Moler, and G. W. Stewart (1979). *LINPACK Users Guide*. Soc. Indust. Appl. Math., Philadelphia, 368 pages.

Forsythe G. E. and W. R. Wasow (1960). Finite Difference Methods for Partial Differential Equations. John Wiley, New York, 444 pages. [Русский перевод: Вазов В. Р., Форсайт Дж. Разностные методы решения дифференциальных уравнений в частных производных.— М.: ИЛ, 1963.]

Fox L. (1965). Introduction to Numerical Linear Algebra. Oxford University Press, Oxford, 327 pages.

Garbow B. S., J. M. Boyle, J. J. Dongarra and C. B. Moler (1977). Matrix Eigensystem Routines—EISPACK Guide Extensions. Lecture Notes in Computer Science, vol. 51. Springer-Verlag, Berlin, 343 pages.

Gentleman M. (1973). Least squares computations by Givens transformations without square roots. J. Inst. Math. Appl., 12, pp. 329—336.

Lawson C. L., R. J. Hanson, D. R. Kincaird and F. T. Krogh (1979). Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software*, **5**, pp. 308—323.

Lawson C L. and R. J. Hanson (1974). Solving Least Squares Problems. Prentice-Hall, Englewood Cliffs, 340 pages.

Oettle W. and W. Prager (1964). Compatibility of approximate solutions of linear equations with given error bounds for coefficients and right-hand sides. Numer. Math., **6**, pp. 405—409.

Parlett B. N. and Y. Wang (1975). The influence of the compiler on the cost of mathematical software—in particular on the cost of triangular factorization. *ACM Trans Math. Software*, **1**, pp. 35—46.

Rice J. R. (1976). Parallel algorithms for adaptive quadrature III: program correctness *ACM Trans. Math. Software*, **2**, pp. 1—30.

Rice J. R. (1976a). The algorithm selection problem, in M. Rubicof and M. Yovits (eds.), Advances in Computers, vol. 15. Academic Press, New York, pp. 65—118.

Rice J. R. (1966). Experiments with Gram-Schmidt orthogonalization. *Math. Comp.*, **20**, pp. 325—328.

Rice J. R. (1966a). A theory of condition. *SIAM J. Num. Anal.*, 3, pp. 287—310.

Ryder B. G. (1974). The PFORT verifier. *Software Practice and Experience*, **4**, pp. 359—378.

Smith B. T., J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema and C. B. Moler (1976). Matrix Systems Routines—EISPACK Guide. Lecture Notes in Computer Science, vol. 6 (2d ed.). Springer-Verlag, Berlin, 551 pages

Stewart G. W. (1973). Introduction to Matrix Computations Academic Press, New York, 441 pages.
Varga R. S. (1962). Matrix Iterative Analysis. Prentice-Hall, Englewood Cliffs, 322 pages.
Von Neumann J. and H. H. Goldstine (1947): Numerical inverting of matrices of high order. *Bull. Amer. Math. Soc.*, 53, pp. 1021—1089.
Wilkinson J. H. (1963). Rounding Errors in Algebraic Processes. Prentice-Hall, Englewood Cliffs, 161 pages.
Young D. M. (1971). Iterative Solution for Large Linear Systems. Academic Press, New York, 563 pages.
Фаддеев Д. К., Фаддеева В. Н. Вычислительные методы линейной алгебры. Изд. 2-е.— М.—Л.: Физматгиз, 1963.

Именной указатель

Предметный указатель

Оглавление