

ДЖ. ГОЛУБ, Ч. ВАН ЛОУН

# МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

ИЗДАТЕЛЬСТВО «МИР»

ДЖ. ГОЛУБ, Ч. ВАН ЛОУН

# МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

Перевод с английского  
Ю. М. Нечепуренко,  
А. Ю. Романова,  
А. В. Собянина и Е. Е. Тыртышникова

под редакцией  
В. В. Воеводина



МОСКВА «МИР»  
1999

# **Matrix Computations**

SECOND EDITION

**Gene H. Golub**

Department of Computer Science  
Stanford University

**Charles F. Van Loan**

Department of Computer Science  
Cornell University

The John Hopkins University Press  
Baltimore and London

УДК 512.83

ББК 22.193

Г 62

**Голуб Дж., Ван Лоун Ч.**

Г62    Матричные вычисления: Пер. с англ. – М.: Мир, 1999. – 548 с., ил.  
ISBN 5-03-002406-9

Книга известных американских математиков-вычислителей представляет собой удачное сочетание учебного пособия и справочника по методам численной алгебры. Изложение сжатое, в рецептурной форме, без доказательств. Книгу отличают методические достоинства: каждый раздел содержит задачи для читателей-студентов и обзор научной литературы – для специалистов.

Для математиков-вычислителей, инженеров, студентов математических и технических специальностей.

**ББК 22.193**

Федеральная целевая программа книгоиздания России

*Редакция литературы по математическим наукам*

**ISBN 6-03-002406-9 (русск.)  
ISBN 0-8018-3739-1 (англ.)**

© 1989 The John Hopkins University Press  
All rights reserved  
© перевод на русский язык, коллектив  
переводчиков, 1999

## *Предисловие редактора перевода*

Вниманию читателя предлагается новая книга по линейной алгебре. И, естественно, возникают вопросы: «Что же действительно новое можно найти в ней и чем конкретно она отличается от других книг по линейной алгебре?»

Как никакая другая ветвь математики, линейная алгебра самым тесным образом переплелась с многочисленными приложениями, являясь либо предметом, либо инструментом исследований. С основами линейной алгебры в той или иной мере знаком каждый человек, соприкасающийся с вычислениями и тем более имеющий дело с решением больших задач на современных суперЭВМ. Круг лиц, интересующихся линейной алгеброй, необычайно широк. Он весьма неоднороден по уровню квалификации и включает в себя студентов, аспирантов, молодых специалистов, а также специалистов высшей квалификации. Все эти обстоятельства делают очень трудным написание книг по линейной алгебре. Желание отразить последние достижения и сделать при этом изложение достаточно простым – всегда противоречивы.

Член Национальной Академии наук США, профессор Стенфордского университета Дж. Голуб и профессор Корнелльского университета Ч. Ван Loan являются специалистами в области линейной алгебры, известными не только в США, но и во всем мире. Их имена хорошо знакомы и специалистам в нашей стране. Несколько лет назад они предприняли попытку описать в доступной форме последние достижения в области численных методов линейной алгебры. Эта попытка воплотилась в книгу «Матричные вычисления», первое издание которой было встречено с интересом. За последние пять лет авторы провели значительную работу по улучшению содержания книги: в частности, были улучшены многие доказательства, включен материал, касающийся параллельных вычислений. Результатом этой работы было второе издание книги «Матричные вычисления». Именно оно и предлагается вниманию читателя.

Авторам удалось успешно сочетать строгость и простоту изложения. Книга читается довольно легко. В ней имеется много нового материала, который еще не публиковался в систематизированном виде на русском языке. Поэтому она будет полезна практически всем специалистам в области вычислительной математики: студентам и аспирантам – как пособие для освоения основ линейной алгебры, прикладникам – как источник рецептов решения задач, опытным специалистам – как пища для размышлений. Последнее обстоятельство особенно важно для нашего читателя, так как представленный в книге материал хорошо отражает уровень исследований за рубежом в области численных методов линейной алгебры.

Объем второго издания книги «Матричные вычисления» увеличен более чем на треть по сравнению с первым. Введение нового материала во многом объясняется быстрым ростом исследований в области параллельных вычислений. Литература по этому направлению плохо систематизирована. Нет еще установленных методов исследования. Не всегда даже ясно, что в параллельных вычислениях является

## *6 Предисловие редактора перевода*

главным, а что – второстепенным. Поэтому попытка авторов навести определенный порядок очень важна. Наличие систематизированных обозначений, методов исследования и источников информации позволяет лучше понять сильные и слабые стороны предметной области. Для всех, кто занимается параллельными вычислениями, сейчас это очень нужно. Для удобства читателя списки литературы помещены по главам.

Добавим к сказанному, что в книге имеется много интересных решений и находок в отдельных доказательствах. Все это можно увидеть, познакомившись с текстом поближе. Хочется надеяться, что в ряду многих книг в области численных методов линейной алгебры предлагаемая читателям книга Дж. Голуба и Ч. Ван Loана займет достойное место.

В заключение отметим, что перевод книги выполнен группой переводчиков: гл. 7, 8, 11 перевел Ю. М. Нечепуренко, гл. 1, 2 – А. Ю. Романов, гл. 3, 4, 12 – А. В. Собянин и, наконец, гл. 6, 9 и 10 – Е. Е. Тыртышников.

*B. B. Воеводин*

*Посвящается  
Олстону С. Хаусхолдеру  
и Джеймсу Х. Уилкинсону*

## Предисловие к первому изданию

Мы придерживаемся той точки зрения, что «предназначение» численного анализа – обеспечить научную общественность эффективными библиотеками программ. Эта деятельность особенно интересна тем, что ее участникам необходимо знание как математики, так и вычислительной техники. В самом деле, разработка программ высокого качества требует и понимания математического существа решаемой задачи, и чутья к конструированию алгоритмов, и осознания специфики вычислений с конечной точностью. Цель этой книги – снабдить читателя нужными навыками в той мере, в которой они относятся к матричным вычислениям.

Начиная с середины 50-х годов в этой области был достигнут огромный прогресс. Это подтверждается существованием высококачественных программ для решения систем линейных уравнений, многих задач, решаемых методом наименьших квадратов, и задач на собственные значения. Типичный пример – подпрограммы пакетов EISPACK и LINPACK, широкое использование которых привело к повышению уровня разработки алгоритмов во многих прикладных областях. Используя модули из этих пакетов в качестве строительного материала, ученые и инженеры могут сооружать более сложные, специально настроенные на их нужды, программируемые средства. Такое положение вещей стимулирует написание хорошо структурированных программ – благоприятная тенденция в условиях, когда все больше научных результатов формулируется в виде программ.

Влияние численной линейной алгебры ощущается и в другом. Наши добрые привычки – склонность полагаться на ортогональные матрицы, осознание роли чувствительности задачи к возмущениям, тщательное рассмотрение ошибок округления – были восприняты во многих других областях исследований. Хороший пример – рост использования сингулярного разложения (SVD) как аналитического инструмента многими статистиками и инженерами в области управления. Работающие в этих областях специалисты переформулируют многие теоретические концепции на языке SVD, и в результате им становятся гораздо легче реализовывать свои идеи в условиях влияния ошибок округления и неточных данных.

Дальнейшие свидетельства растущего влияния численной линейной алгебры можно обнаружить в области разработки аппаратных средств ЭВМ. Последние продвижения в области арифметики с плавающей точкой и параллельной обработки были в немалой степени вызваны деятельностью в области матричных вычислений.

Мы написали эту книгу, чтобы охватить эту очень интересную и быстро расширяющуюся область с единых позиций. Многое было сделано после публикации в 1965 г. монументального трактата Уилкинсона «Алгебраическая проблема собственных значений». Многие из этих современных достижений уже отражены в обзорных статьях и в известных специальных монографиях, таких как «Численное решение задачи наименьших квадратов» Лоусона и Хенсона, «Симметричная проблема собственных значений» Парлетта. Мы считаем, что пришло время для

синтеза всего этого материала. В этом отношении мы рассматриваем «Матричные вычисления» как всеобъемлющую, несколько более продвинутую версию книги Стьюарта «Введение в матричные вычисления».

Мы рассчитываем на три категории читателей: студенты-выпускники технических специальностей, ученые и инженеры-вычислители, наконец, наши коллеги по численному анализу. Для каждой из этих групп у нас в книге кое-что припасено.

Для студентов (и их наставников) мы включили значительное количество упражнений. Многие из них носят вычислительный характер и могут стать ядром программного проекта. Наш собственный опыт преподавания на основе этой книги говорит о том, что задания, использующие Eispack и Linpack, существенно оживляют ее содержание. Также успешно использовалась совместно с этим текстом система Matlab – простая в обращении и выполняющая вычисления с матрицами.

Для инженеров и ученых, желающих в процессе своей работы пользоваться этой книгой как справочником, мы постарались свести к минимуму взаимозависимость отдельных глав. Кроме того, в конце практически каждого раздела мы приводим аннотированную библиографию, чтобы ускорить поиск дополнительного материала по любому заданному вопросу.

Для наших коллег по численному анализу мы щедро сопровождаем наши описания алгоритмов элементами теории возмущений и анализом ошибок. Мы хотим снабдить эту категорию читателей достаточным количеством подробностей, чтобы они могли ставить и решать матричные задачи, возникающие в их собственной деятельности. Часто толчком к исследованиям в области численной линейной алгебры служат работы, проводимые в других областях численного анализа. Так, например, некоторые из лучших методов решения разреженных линейных систем были разработаны исследователями, занимавшимися численным решением уравнений в частных производных. Подобным же образом развитие приемов модификации различных матричных разложений было стимулировано работами по квазиньютоновским методам.

Эта книга писалась шесть лет. Название менялось несколько раз: (1) «Завершающий курс матричных вычислений», (2) «Прикладные матричные вычисления», (3) «Матричные вычисления: специальный курс». Первое название, помимо некоторой претенциозности, вводит в заблуждение. Мы не претендуем на исчерпывающее изложение матричных вычислений. В частности, мы исключили многие темы из животрепещущей области вычислений с разреженными матрицами просто потому, что не хотели погружаться в теорию графов и структур данных. Второе название не подходит, потому что мы не останавливаемся в сколько-нибудь значительной степени на приложениях. По большей части рассматриваемые матричные задачи мы принимаем как данность – их происхождение не прослеживается. Мы согласны с тем, что с педагогической точки зрения это является недостатком, однако опытный преподаватель сумеет его компенсировать. А потом, мы подозреваем, что многие из наших читателей и сами будут обладать достаточным опытом, так что чрезмерная мотивировка им и не нужна. Наконец, с последним названием мы расстались ввиду того, что книга все же содержит вводный материал. Мы решили включить элементарные разделы и ради полноты, и потому, что мы думаем, что наш подход к основам предмета заинтересует преподавателей начальных курсов численного анализа.

О чём же тогда эта книга, если она неполна, не очень-то прикладная и не слишком узко специальная? Ответом на этот вопрос должен послужить краткий обзор содержания.

В первых трех главах закладываются необходимые основы. Приводится обзор матричной алгебры, фиксируются некоторые ключевые алгоритмы. Темп изложения достаточно высок. Читатели, у которых возникают трудности при решении задач из

этих первых глав, безусловно, столкнутся с ними и в остальной части книги.

Значительная часть текущих исследований в численной линейной алгебре сосредоточена на задачах, в которых матрицы имеют специальную структуру, скажем являются большими и разреженными. Искусство извлечения выгоды из наличия структуры – центральная тема гл. 5, где описываются многие методы решения систем линейных уравнений специального вида.

В гл. 6 подхвачено другое течение – возросшее стремление полагаться на ортогональные матрицы. Мы обсуждаем несколько методов ортогонализации и показываем, как применить их в методе наименьших квадратов. Особое внимание уделяется случаю неполного ранга.

Центральное место гл. 7 занимает всемогущий QR-алгоритм для решения несимметричной проблемы собственных значений. Наш методический вывод должен помочь снять налет таинственности с этого важного метода. Мы останавливаемся также на вычислении инвариантных подпространств и обобщенной проблеме собственных значений.

В гл. 8 мы продолжаем обсуждать проблему собственных значений, сосредоточиваясь на важном симметричном случае. Сначала мы описываем симметричный QR-алгоритм, а затем показываем, как симметрия позволяет построить несколько альтернативных вычислительных процедур.

До самого этого места в книге наше рассмотрение разреженности носит отрывочный характер. В гл. 5 обсуждается решение ленточных линейных систем, в гл. 7 описан метод одновременных итераций, в гл. 8 – итераций Рэлея и т. д. Главы 9 и 10, однако, полностью посвящены решению задач с разреженными матрицами. Темами обсуждения являются метод Ланьша и родственный ему метод сопряженных градиентов. Мы показываем, как эти важные алгоритмы могут быть использованы для решения многих разреженных задач на собственные значения, по методу наименьших квадратов, систем линейных уравнений.

Цель двух последних глав – проиллюстрировать широкую применимость представленных в книге алгоритмов. В гл. 11 речь идет о задаче вычисления функции от матрицы, что часто требуется делать в приложениях теории управления. Глава 12 содержит подборку матричных задач, из которых несколько демонстрируют мощь сингулярного разложения.

Практическая и теоретическая ценность этого разложения является, пожалуй, сквозной темой в этой книге. Действительно, его алгоритмические и математические свойства играют ключевую роль почти в каждой главе. Во многих отношениях данную книгу можно рассматривать как красочное развитие манускрипта нашего коллеги Алены Клайна: «Все, Что Вы Хотели Знать о Сингулярном Разложении (но боялись спросить)».

Пора сказать несколько слов о ссылках на имеющиеся программы. Мы ориентируемся на EISPACK и LINPACK, и почти каждая подпрограмма из этих пакетов так или иначе упоминается в тексте. Кроме того, многие технические доклады, ссылки на которые мы даем в аннотированных библиографиях, являются на самом деле описаниями программ. Следует подчеркнуть, однако, что с большей частью этих программ мы напрямую не работали (исключая EISPACK и LINPACK), а посему – *caveat emptor* (будьте осторожны (*лат.*)).

В создании этой книги нам помогало много специалистов. Richard Bartels помог подобрать ссылки и переработать черновик нескольких глав. Организованная им школа матричных вычислений, проведенная в августе 1977 г. в John Hopkins University, стимулировала написание этой книги.

Bob Plemmens, John Dennis, Alan Laub и Don Heller использовали фрагменты книги в преподавании и предложили нам многочисленные конструктивные замечания. George Cybenko очень помог нам в разделе о теплицевых матрицах. Во

Kågström сделал много ценных замечаний по нашей трактовке вычисления инвариантных подпространств. Per-Åke Wedin тщательно прочитал ранние версии гл. 5 и квалифицированно направлял последующие переработки. Окончательное название книги независимо предложили Uri Ascher и Roger Horn, которых мы и хотим здесь поблагодарить. Мы также благодарны автору анонимного обзора, внесшему много ценных предложений.

И последнее, по порядку, но не по значению, мы с радостью признаем влияние наших коллег Cleve Moler и Pete Stewart. Их работы, великолепно схватывающие дух нашей науки, существенно воздействовали на сбалансированность и стиль нашего собственного изложения.

# Предисловие ко второму изданию

Две причины побудили нас написать переработанную и дополненную версию «Матричных вычислений». Во-первых, после пяти лет преподавания по прежнему изданию и многочисленных замечаний наших коллег стало ясно, что многое из написанного можно было написать лучше. Во многих доказательствах и выводах была достигнута большая ясность. Кроме того, мы приняли стилизованный вариант системы Matlab, облегчающий «матрично-векторное мышление». Другая новая черта второго издания – доведение рубрикации материала до уровня подразделов. Это должно облегчить пользование книгой как учебником и как справочником.

Вторая причина появления нового издания связана с расцветом параллельных матричных вычислений. Многопроцессорные ЭВМ революционизируют роль вычислений в науке и технике, и важно, чтобы вклад численной линейной алгебры в этот процесс был документально оформлен. Мы это делаем на машинно-независимом уровне, подчеркивая общие алгоритмические идеи в противовес конкретным реализациям. Как мы говорим в новой главе о параллельных матричных вычислениях, эта область очень подвижна, и литература наводнена исследованиями отдельных случаев. Несмотря на это, в области разработки алгоритмов можно набрать дюжины новых идей, играющих ключевую роль, которые, по всей вероятности, пригодятся нам еще в течение длительного периода и созрели для обсуждения на уровне учебника.

Вот обзор новшеств второго издания (по главам):

В гл. 1 (Умножение матриц) мы вводим обозначения и фундаментальные понятия на примере умножения матриц. Большое внимание уделяется приемам работы с блочными матрицами; добавлен также новый раздел о векторно-конвейерных вычислениях.

В гл. 2 (Матричный анализ) мы собрали математические основы, необходимые для вывода и анализа алгоритмов решения линейных систем и задачи наименьших квадратов.

Гл. 3 (Линейные системы общего вида) и 4 (Линейные системы специального вида) были дополнены новым материалом о «высокопроизводительной» организации гауссова исключения и разложения Холецкого. Акцент делается на реализации, богатые матрично-векторными и матрично-матричными умножениями. Был также добавлен подраздел о положительно полуопределенных матрицах.

В гл. 5 (Ортогонализация и наименьшие квадраты) появился новый материал о блочных преобразованиях Хаусхольдера. Мы также изменили порядок изложения с целью отделить обсуждение ортогональных разложений от решения задачи наименьших квадратов.

В гл. 6 (Параллельные матричные вычисления) мы, используя операцию гахру, вводим понятия вычислений над распределенной и общей памятью. Для этих двух моделей параллельных вычислений мы обсуждаем организацию перемножения матриц и различных матричных разложений.

В гл. 7 (Несимметричная проблема собственных значений) мы добавили новый подраздел о приведении к блочно-хессенбергову виду. В гл. 8 (Симметричная проблема собственных значений) внесены два более фундаментальных изменения. Раздел о методе Якоби был полностью переработан в свете последних достижений в параллельных вычислениях. Был также добавлен раздел о новом высокопараллельном алгоритме типа «разделяй и властвуй» для трехдиагональных матриц.

В гл. 9 (Метод Ланцоша) мы добавили подраздел о методе Арнольди. В гл. 10 (Итерационные методы решения линейных систем) расширено обсуждение метода сопряженных градиентов с переобуславливанием. Единственное существенное изменение в гл. 11 (Функции от матриц) и гл. 12 (Специальные вопросы) – это новый подраздел в § 12.6 о гиперболических модификациях.

Мы в большом долгу перед многими читателями, указывавшими нам на опечатки, ошибки и недостатки изложения первого издания. Для нас было удовольствием общаться со столь заинтересованной и дружелюбно настроенной читательской аудиторией. Из большого количества наших корреспондентов мы хотели бы особо поблагодарить Å. Björck, J. Bunch, J. Dennis, T. Ericsson, O. Hald, N. Higham, A. Laub, R. Le Veque, M. Overton, B. N. Parlett, R. Plemmens, R. Skeel, E. Stickel, S. Van Huffel, R. S. Varga за их очень подробные и уместные замечания.

Мы хотим также отметить вклад лиц и организаций, сыгравших важнейшую роль в производстве и оформлении второго издания. На уровне системы LATEX нам помогали поддерживать Alex Aiken, Chris Bischof, Gil Neiger и Hal Perkins из Корнелла и Mark Kent из Станфорда. Cindy Robertson-Hubbell из Корнелла ввела в компьютер первое издание и была абсолютно незаменима на всех последующих стадиях производства. Исследовательские средства, которые предоставили Iain Duff из Harwell Laboratory, Соединенное Королевство, и Bill Morton из Оксфордского университета, сыграли существенную роль в процессе переработки. Iain Duff, Bo Kågström, Chris Paige и Nick Trefethen внесли в эту работу глубокий философский и технический вклад. Этим друзьям и коллегам адресована наша искренняя благодарность.

Рукопись прочитал целиком Nick Higham, внесший затем бесчисленное количество предложений. Без его помощи публикация второго издания была бы отсрочена и получилась бы худшего качества.

И наконец, мы хотим с признательностью отметить большой вклад нашего дорогого коллеги Джима Уилкинсона, ушедшего от нас в 1986 г. Память о нем продолжает быть большим источником вдохновения, и мы с удовольствием посвящаем этот новый том одновременно ему и Олстону Хаусхолдеру.

# Как пользоваться этой книгой

## О сокращениях

Следующие книги часто цитируются в нашем тексте:

- SLE Forsythe G. E. and Moler C. (1967). Computer Solution of Linear Algebraic Systems, Prentice Hall, Englewood Cliffs, NJ. [Имеется русский перевод: Форсайт Дж., Молер К. Численное решение систем линейных алгебраических уравнений. – М.: Мир, 1969.]
- SLS Lawson C. L. and Hanson R. J. (1974). Solving Least Squares Problems, Prentice Hall, Englewood Cliffs, NJ. [Имеется русский перевод: Лоусон Ч., Хенсон Р. Численное решение задач метода наименьших квадратов. – М.: Наука, 1986.]
- SEP Parlett B. N. (1980). The Symmetric Eigenvalue Problem, Prentice Hall, Englewood Cliffs, NJ. [Имеется русский перевод: Парлетт Б. Симметричная проблема собственных значений. Численные методы. – М.: Мир, 1983.]
- IMC Stewart G. W. (1973). Introduction to Matrix Computations, Academic Press, New York.
- AEP Wilkinson J. H. (1965). The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, England. [Имеется русский перевод: Уилкинсон Дж. Алгебраическая проблема собственных значений. – М.: Наука, 1970.]

Для этих «глобальных» ссылок мы используем приведенные мнемоники, например «Уилкинсон (AEP, гл. 5)». Библиографическая информация по конкретным разделам приводится в конце этих разделов. Основная библиография приведена в конце книги. При ссылках на пакеты программ Linpack и Eispack имеются в виду соответствующие описания

Smith B. T., Boyle J. M., Ikebe Y., Klema V. C., and Moler C. B. (1970). Matrix Eigen-system Routines: EISPACK Guide, 2nd ed., Springer-Verlag, New York.

Smith B. T., Boyle J. M., Dongarra J. J., and Moler C. B. (1972). Matrix Eigensystem Routines: EISPACK Guide Extension, Springer-Verlag, New York.

Dongarra J., Bunch J. R., Moler C. B., and Stewart G. W. (1978). LINPACK User Guide, SIAM Publications, Philadelphia.

Таким образом, фраза «подпрограмма  $xuz$  имеется в пакете Linpack, гл. 2» означает, что эта подпрограмма описывается во второй главе руководства по Linpack.

Алгольные версии многих подпрограмм из пакетов Linpack и Eispack собраны в книге

HACLA Wilkinson J. H. and Reinsch C., eds. (1971). Handbook for Automatic Computation, Vol. 2, Linear Algebra, Springer-Verlag, New York. [Имеется русский перевод: Уилкинсон Дж., Райнш К. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. – М.: Машиностроение, 1976.]

Во время написания данной книги (1989), программы, реализующие многие из представленных в ней алгоритмов, можно было получить, отправив электронную почту по любому из адресов:

netlib@anl-mcs.arga  
netlib@research.att.com  
research!netlib

Типичные запросы:

send index	{Список имеющихся библиотек}
send index for linpack	{Список программ Linpack}
send svd from eispack	{Программа вычисления SVD из Eispack}

Этот сервис электронной рассылки описан в

Dongarra J.J. and Grosse E. (1987). “Distribution of Mathematical Software Via Electronic Mail”, Comm. ACM 30, 403–407

Сообщим также, что основная библиография этой книги (в формате LATEX) может быть получена по netlib.

# К преподавателям

Эта книга может быть использована для нескольких различных типов курсов.  
Вот примерные варианты:

Название курса	Введение в матричные вычисления (1 семестр)
Программа	Главы 1–4, 5.1–5.3, 7.1–7.6, 8.1–8.2
Название курса	Матричные вычисления (2 семестра)
Программа	Главы 1–12
Название курса	Решение систем линейных уравнений и задачи наименьших квадратов (1 семестр)
Программа	Главы 3–5, 12.1–12.4, 12.6
Название курса	Задачи на собственные значения (1 семестр)
Программа	Главы 7–9, 11, 12.5
Название курса	Параллельные матричные вычисления (1 семестр)
Программа	Главы 1, 6, 8.5–8.6

В каждом случае мы настоятельно рекомендуем включение вычислительных заданий, предполагающих использование высококлассных программных пакетов, таких как Linpack и Eispack. Материал может быть еще больше оживлен использованием Matlab – простой в обращении системы, выполняющей вычисления с матрицами. В этой связи мы рекомендуем сопровождающее ее руководство

Coleman T. F. and Van Loan C. F. (1988). Handbook for Matrix Computations, SIAM Publications, Philadelphia, PA.

Оно содержит главы по Фортрану-77, по элементарным подпрограммам линейной алгебры (BLAS), по Linpack и Matlab.

Наконец, упомянем новый учебник

Hager W. W. (1988). Applied Numerical Linear Algebra, Prentice Hall, Englewood Cliffs, NJ.

который, возможно, больше, чем «Матричные вычисления», подойдет для начинающих студентов.

## Глава 1

# Умножение матриц

Систематическое изучение матричных вычислений начинается с задачи умножения матриц. Хотя математически эта задача очень проста, она таит немало богатств с вычислительной точки зрения. Мы начинаем с рассмотрения в § 1.1 нескольких возможных способов организации матричного умножения. Заблаговременно вводится язык блочных разбиений матриц, который мы используем, чтобы охарактеризовать несколько линейно-алгебраических «уровней» вычислений.

Если матрица обладает какой-то структурой, то обычно удается использовать это обстоятельство. Например, для хранения симметричной матрицы достаточно половины того места, которое требуется для размещения матрицы общего вида; умножение на вектор матрицы, в которой много нулевых элементов, может потребовать значительно меньше времени, чем вычисление произведения матрицы общего вида и вектора. Эти вопросы обсуждаются в § 1.2. В § 1.3 вводятся обозначения для блочных матриц. Блочная матрица – это такая матрица, элементами которой также являются матрицы. Это очень важное понятие как с теоретической, так и с практической точки зрения. С теоретической стороны блочные обозначения позволяют получить очень сжатые доказательства для важнейших матричных разложений, являющихся краеугольным камнем численной линейной алгебры. С вычислительной точки зрения блочные алгоритмы важны, поскольку в них много матричных умножений, – а эта операция особенно хорошо реализуется на многих высокопроизводительных ЭВМ новых архитектур.

Эти новые архитектуры требуют, чтобы разработчик алгоритмов уделял обменам с памятью не меньше внимания, чем количеству арифметических действий, что вносит в научные вычисления целое новое измерение. Мы иллюстрируем это в § 1.4, где мы рассматриваем такие важнейшие аспекты векторно-конвейерных вычислений, как длина вектора, шаг выборки, количество векторных загрузок/записей, а также степень повторного использования вектора.

## 1.1. Основные алгоритмы и обозначения

Матричные вычисления строятся на иерархии операций линейной алгебры. Скалярные произведения состоят из скалярных операций сложения и умножения; умножение матрицы на вектор составлено из скалярных произведений; перемножение матриц сводится к набору умножений матрицы на вектор. Все эти операции могут быть описаны в алгоритмическом виде или на языке линейной алгебры. Наша главная цель в этом разделе – продемонстрировать, как эти два способа дополняют друг друга. Попутно мы вводим обозначения и знакомим читателя с тем стилем мышления, на котором основываются методы матричных вычислений. Дискуссия вращается вокруг перемножения матриц, которое, как мы показываем, может быть организовано несколькими способами.

### 1.1.1. Обозначения для матриц

Пусть  $\mathbf{R}$  – множество вещественных чисел. Мы обозначаем через  $\mathbf{R}^{m \times n}$  векторное пространство всех вещественных  $m \times n$ -матриц:

$$A \in \mathbf{R}^{m \times n} \Leftrightarrow A = (a_{ij}) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, a_{ij} \in \mathbf{R}.$$

В тех случаях, когда для обозначения матрицы используется заглавная буква (например,  $A$ ,  $B$ ,  $\Delta$ ), соответствующая строчная буква с индексом  $ij$  относится к  $(i, j)$ -му элементу этой матрицы (например,  $a_{ij}$ ,  $b_{ij}$ ,  $\delta_{ij}$ ). Кроме того, для  $(i, j)$ -го элемента матрицы  $A$  мы используем обозначение  $[A]_{ij}$ . При детальной записи алгоритма мы ссылаемся на элементы матриц по правилам Фортрана:  $A(i, j)$ .

### 1.1.2. Операции над матрицами

Основные манипуляции с матрицами включают:

транспонирование ( $\mathbf{R}^{m \times n} \rightarrow \mathbf{R}^{n \times m}$ )

$$C = A^T \Rightarrow c_{ij} = a_{ji},$$

сложение ( $\mathbf{R}^{m \times n} \times \mathbf{R}^{m \times n} \rightarrow \mathbf{R}^{m \times n}$ )

$$C = A + B \Rightarrow c_{ij} = a_{ij} + b_{ij},$$

умножение матрицы на число ( $\mathbf{R} \times \mathbf{R}^{m \times n} \rightarrow \mathbf{R}^{m \times n}$ )

$$C = aA \Rightarrow c_{ij} = a_{ij},$$

умножение матрицы на матрицу ( $\mathbf{R}^{m \times r} \times \mathbf{R}^{r \times n} \rightarrow \mathbf{R}^{m \times n}$ )

$$C = AB \Rightarrow c_{ij} = \sum_{k=1}^r a_{ik} b_{kj}.$$

Это те кирпичики, из которых строятся матричные вычисления.

### 1.1.3. Обозначения для векторов

Пусть  $\mathbf{R}^n$  – векторное пространство вещественных векторов:

$$x \in \mathbf{R}^n \Leftrightarrow x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, x_i \in \mathbf{R}.$$

Мы обозначаем через  $x_i$   $i$ -ю компоненту вектора  $x$ ; при записи алгоритмов мы также используем обозначение  $x(i)$ .

Заметим, что мы отождествляем  $\mathbf{R}^n$  с  $\mathbf{R}^{n \times 1}$ , так что элементы  $\mathbf{R}^n$  – это векторы-столбцы. С другой стороны,  $\mathbf{R}^{1 \times n}$  состоит из векторов-строк:

$$x \in \mathbf{R}^{1 \times n} \Leftrightarrow x = (x_1, \dots, x_n).$$

Если  $x$  – вектор-столбец, то  $y = x^T$  – вектор-строка.

Несколько «подозрительно» выглядит операция над векторами, называемая *внешним произведением*:

$$C = xy^T, x \in \mathbf{R}^m, y \in \mathbf{R}^n.$$

Но это совершенно законное матричное умножение, поскольку количество столбцов в  $x$  совпадает с количеством строк в  $y^T$ .

Если  $C = xy^T$ , то  $c_{ij} = x_i y_j$ , так что, например,

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 5 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 12 & 15 \end{bmatrix}.$$

### 1.1.4. Операции над векторами

Существуют четыре основные операции над векторами. Если  $a \in \mathbf{R}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{z}^n$ , то мы имеем умножение вектора на число  $z = a x$  ( $z_i = ax_i$ ), сложение векторов  $z = x + y$  ( $z_i = x_i + y_i$ ), скалярное произведение  $c = x^T y$  ( $c = \sum_{i=1}^n x_i y_i$ ) и покомпонентное произведение  $z = x.*y$  ( $z_i = x_i y_i$ ). Пятая операция – *saxpy* – настолько важна для матричных вычислений, что мы дополняем ею наш список основных операций над векторами, несмотря на то, что это делает наш список избыточным. Операция *saxpy* определяется следующим образом:

$$z = ax + y \Rightarrow z_i = ax_i + y_i.$$

Название “*saxpy*” используется в пакете Linpack, в котором реализованы многие из приведенных в этой книге алгоритмов. Можно воспринимать “*saxpy*” как мнемонику для выражения «скаляр альфа, умноженный на  $x$ , плюс  $y$ » (scalar alpha  $x$  plus  $y$ ).

### 1.1.5. Вычисление скалярных произведений и операций *saxpy*

Для записи алгоритмов мы решили использовать стилизованную версию языка Matlab. Matlab – это элегантная интерактивная система, идеально приспособленная для вычислений с матрицами. См. Coleman and Van Loan (1988, гл. 4). Наш способ записи вводится постепенно на протяжении этой главы; здесь мы начинаем с функции, вычисляющей скалярные произведения.

**Алгоритм 1.1.1 (Скалярное произведение).** По  $n$ -векторам  $x$  и  $y$  этот алгоритм вычисляет их скалярное произведение  $c = x^T y$ .

```
function: c = dot(x, y);
c = 0
n = length(x)
for i = 1:n
    c = c + x(i)*y(i)
end
end dot
```

В этой процедуре размерность векторов дается функцией **length**. Оператор **for** показывает, что переменная-счетчик  $i$  принимает значения 1, 2, ...,  $n$ . Значение скалярного произведения возвращается в  $c$ . Скалярное произведение двух  $n$ -векторов требует  $n$  умножений и  $n$  сложений. Менее формально – это  $O(n)$ -операция, в том смысле что объем работы линейно зависит от размерности.

Вычисление *saxpy* – это также  $O(n)$ -операция, но она возвращает не скаляр, а вектор:

**Алгоритм 1.1.2 (Saxpy).** По  $n$ -векторам  $x$  и  $y$  и скаляру  $a$  этот алгоритм вычисляет  $z = ax + y$ .

```

function: z = saxpy( $\alpha$ , x, y)
n = length(x)
for i = 1:n
     $z(i) = \alpha x(i) + y(i)$ 
end
end saxpy

```

### 1.1.6. О формализме и стиле изложения

Необходимо подчеркнуть, что приводимые в этой книге алгоритмы, такие как **dot** и **saxpy**, лишь заключают в себе важнейшие вычислительные идеи, но никак не являются техническими инструкциями для написания программ. Более того, там, где того требует изложение, записи алгоритмов могут становиться весьма неформальными. Для описания вычислений в этом и в нескольких последующих разделах оказываются удобными функции системы **Matlab**, но дальше по ходу книги алгоритмы становятся слишком сложными для детализации на “ij”-уровне, и тогда мы прибегаем к менее формальной записи. Для читателя, который по первым главам приобретет интуитивное понимание матричных вычислений, это не должно представлять трудностей.

### 1.1.7. Умножение матрицы на вектор

Пусть  $A \in \mathbb{R}^{m \times n}$ , и мы хотим вычислить произведение  $z = Ax$ , где  $x \in \mathbb{R}^n$ . Стандартный способ вычисления заключается в последовательном подсчете скалярных произведений

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

Это приводит к следующему алгоритму.

**Алгоритм 1.1.3 (Умножение матрицы на вектор: Версия с доступом по строкам).** По  $A \in \mathbb{R}^{m \times n}$  и  $x \in \mathbb{R}^n$  следующий алгоритм вычисляет  $z = Ax$ .

```

function: z = matvec.ij(A, x)
m = rows(A); n = cols(A)
z(1:m) = 0
for i = 1:m
    for j = 1:n
         $z(i) = z(i) + A(i, j)x(j)$ 
    end
end
end matvec.ij

```

Функции **rows** и **cols** принимают в качестве аргумента матрицу и возвращают количество ее строк и столбцов соответственно. Оператор  $z(1:m) = 0$  инициализирует  $z$  как нулевой  $m \times 1$ -вектор.

Альтернативный алгоритм получится, если рассмотреть  $z = Ax$  как линейную комбинацию столбцов матрицы  $A$ , например

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 8 \\ 3 \cdot 7 + 4 \cdot 8 \\ 5 \cdot 7 + 6 \cdot 8 \end{bmatrix} = 7 \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 23 \\ 53 \\ 83 \end{bmatrix}.$$

**Алгоритм 1.1.4 (Умножение матрицы на вектор: Версия с доступом по столбцам).** По  $A \in \mathbb{R}^{m \times n}$  и  $x \in \mathbb{R}^n$  следующий алгоритм вычисляет  $z = Ax$ .

```

function:  $z = \text{matvec.ji}(A, x)$ 
     $m = \text{rows}(A); n = \text{cols}(A); z(1:m) = 0$ 
    for  $j = 1:n$ 
        for  $i = 1:m$ 
             $z(i) = z(i) + x(j) A(i, j)$ 
        end
    end
end matvec.ji
```

Заметим, что цикл по  $i$  выполняет операцию схору. Мы получили столбцовую версию, переосмыслив на уровне векторов алгоритм матрично-векторного умножения. Другим способом мы могли бы вывести столбцовую версию, поменяв порядок следования циклов в строчном алгоритме. В матричных вычислениях важно осознавать алгебраический смысл модификаций программ, таких как перестановка циклов.

### 1.1.8. Разбиение матрицы на строки и столбцы

Анализ внутренних циклов показывает, что в алгоритме 1.1.3 доступ к элементам матрицы  $A$  осуществляется по строкам, а в алгоритме 1.1.4 – по столбцам. Чтобы яснее охарактеризовать эти методы доступа, нам понадобится язык *блочных разбиений матриц*. С точки зрения строк, матрица представляет собой набор векторов-строк:

$$A \in \mathbb{R}^{m \times n} \Leftrightarrow A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix}, a_k \in \mathbb{R}^n. \quad (1.1.1)$$

Это *разбиение матрицы  $A$  на строки*. Так, разбиение на строки матрицы

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

означает, что мы предполагаем рассматривать  $A$  как набор строк, где

$$a_1^T = [1 \ 2], a_2^T = [3 \ 4], a_3^T = [5 \ 6].$$

С учетом разбиения на строки (1.1.1) мы видим, что функция **matvec.ij** устроена, в сущности, следующим образом:

```

for  $i = 1:m$ 
     $z_i = a_i^T x$ 
end
```

Иначе матрицу можно рассматривать как набор векторов-столбцов:

$$A \in \mathbb{R}^{m \times n} \Leftrightarrow A = [a_1, \dots, a_n], a_k \in \mathbb{R}_m. \quad (1.1.2)$$

Это *разбиение матрицы  $A$  на столбцы*. Для вышеприведенного примера мы положили бы  $a_1$  и  $a_2$  равными первому и второму столбцу матрицы  $A$  соответственно:

$$a_1 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, a_2 = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

С учетом разбиения (1.1.2) мы видим, что функция `matvec.ji` – это использующая `saxpy` процедура, в которой доступ к элементам матрицы  $A$  идет по столбцам:

```
z(1:m) = 0
for j = 1:n
    z = z + x_j a_j
end
```

Отметим, что в векторе  $z$  накапливается сумма, значение которой обновляется операциями `saxpy`.

### 1.1.9. Использование двоеточия

Удобным способом сослаться на какой-либо столбец или строку матрицы является использование двоеточия. Обозначим через  $A(k, :)$   $k$ -ю строку матрицы  $A$  ( $A \in \mathbb{R}^m \times n$ ):

$$A(k, :) = [a_{k1}, \dots, a_{kn}].$$

Аналогично, обозначим через  $A(:, k)$   $k$ -й столбец матрицы  $A$ :

$$A(:, k) = \begin{bmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{bmatrix}.$$

Используя эти соглашения, мы можем переписать `matvec.ij` в виде

<code>for i = 1:m</code> <code>    z(i) = A(i, :)x или</code> <code>end</code>	<code>for i = 1:m</code> <code>    z(i) = dot(A(i, :), x)</code> <code>end</code>
--	---

Использующая `saxpy` версия `matvec.ji` запишется в виде

<code>z(1:m) = 0</code> <code>for j = 1:n</code> <code>    z = z + x(j) A(:, j) или</code> <code>end</code>	<code>z(1:m) = 0</code> <code>for j = 1:n</code> <code>    z = saxpy(x(j), A(:, j), z)</code> <code>end</code>
--	---

При описании матричного алгоритма важно верно выбрать уровень и стиль описания. Использование двоеточия позволяет исключить из рассмотрения внутренние циклы, тем самым фокусируя внимание на операциях более высокого уровня. Используя функции, мы можем выявить ключевые вычислительные ядра, которые могут отражать хорошие программные разработки, например `dot` и `saxpy`.

### 1.1.10. Модификация внешним произведением

Как мы видели, метод доступа является важным атрибутом матричного алгоритма. По причинам, о которых мы скажем позже, предпочтительными оказываются алгоритмы, выбирающие элементы массивов по столбцам. Так, версия `matvec.ji` в большинстве случаев предпочитаются версии `matvec.ij`. Вспомним, что единственная разница между этими версиями – порядок следования двух циклов. Важно осознать эту взаимосвязь между порядком следования циклов и методом доступа

к данным. Для дальнейшей иллюстрации этого рассмотрим модификацию матрицы  $A$  внешним произведением:

$$A \leftarrow A + xy^T, \quad A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^m, \quad y \in \mathbb{R}^n.$$

Здесь  $\leftarrow$  означает присваивание. Вспомним, что  $xy^T$  – это просто частный случай произведения матриц, так что элементы матрицы  $A$  пересчитываются по формуле

$$a_{ij} = a_{ij} + x_i y_j, \quad i = 1:m, \quad j = 1:n.$$

Версия “ $i$ ” этого вычисления говорит, что надо к каждой строке матрицы  $A$  прибавить вектор, кратный  $y^T$ :

```
for i = 1:m
    A(i,:) = A(i,:)+x(i)*y^T
end
```

С другой стороны, версия “ $j$ ”

```
for j = 1:n
    A(:,j) = A(:,j)+y(j)*x
end
```

выбирает элементы матрицы  $A$  по столбцам. Заметим, что обе процедуры используют `saxpy`.

### 1.1.11. Вычисление операции `gaxpy`

Модификация матрицы внешним произведением векторов занимает большое место в традиционных формулировках многих важных матричных алгоритмов. Оказывается, большинство этих алгоритмов можно переформулировать так, что доминирующей становится операция `gaxpy`. Операция `gaxpy` – это просто вычисление вида

$$z = y + Ax, \quad x \in \mathbb{R}^n, \quad y \in \mathbb{R}^m, \quad A \in \mathbb{R}^{m \times n}.$$

Как и “`saxpy`”, термин “`gaxpy`” обязан своим происхождением программному обеспечению. Можно рассматривать его как мнемонику для фразы «произвольная матрица  $A$ , умноженная на  $x$  плюс  $y$ » (general  $Ax$  plus  $y$ ). Как будет объяснено в § 1.4, формулировки посредством `gaxpy` (должным образом реализованные), как правило, предпочтительнее формулировок посредством модификации внешним произведением. Это очень важное с вычислительной точки зрения положение, так что `gaxpy` достойна формального алгоритмического представления:

**Алгоритм 1.1.5 (Gaxpy).** По  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$  и  $A \in \mathbb{R}^{m \times n}$  этот алгоритм вычисляет  $z = y + Ax$ .

```
function: z = gaxpy(A, x, y)
    n = cols(A); z = y
    for j = 1:n
        z = z + x(j)*A(:,j)
    end
end gaxpy
```

Мы организовали вычисления так, что в векторе  $z$  накапливается сумма, значение которой обновляется последовательностью операций `saxpy`. При такой формулировке мы видим, что `gaxpy` является обобщением `saxpy`.

### 1.1.12. Понятие уровня

Скалярное произведение и схору – это примеры операций «уровня 1». Для операций уровня 1 количество данных и количество арифметических действий линейно зависят от размерности операции. Модификация  $m \times n$ -матрицы внешним произведением или операция схору производят квадратичный объем работы  $O(mn)$ . Это примеры операций «уровня 2».

Разработка матричных алгоритмов, содержащих много операций высокого уровня, является областью интенсивных исследований, и мы постоянно будем возвращаться к этой теме в нашей книге. К примеру, для высокой производительности алгоритма решения систем линейных уравнений может потребоваться, чтобы гауссово исключение было организовано через операции схору уровня 2. Для этого понадобится некоторое переосмысление обычного алгоритма, так как в стандартной формулировке гауссово исключение описывается на уровне 1, например, «умножить строку 1 на константу и прибавить результат к строке 2».

Кроме того, некоторые алгоритмы можно организовать таким образом, что в них будет много умножений матрицы на матрицу – это операция уровня 3. Операции уровня 3 производят кубический объем работы над квадратичным количеством данных. Если  $A$ ,  $B$  и  $C$  – матрицы, то вычисления  $C = C + AB$  и  $C = C + + AB$  – примеры операций уровня 3. Эти состоящие из большого количества действий вычисления могут быть организованы многочисленными способами, как мы сейчас продемонстрируем на примере обычного умножения матриц  $C = AB$ .

### 1.1.13. Умножение матриц

Рассмотрим задачу вычисления произведения  $2 \times 2$ -матриц  $C = AB$ . В терминах скалярных произведений каждый элемент матрицы  $C$  вычисляется как скалярное произведение:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix}.$$

В версии, использующей схору, каждый столбец матрицы  $C$  рассматривается как линейная комбинация столбцов матрицы  $A$ :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 4 \end{bmatrix} & 6 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \end{bmatrix} \end{bmatrix}.$$

Наконец, в версии, использующей внешние произведения,  $C$  получается как сумма внешних произведений:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} [5 \ 6] + \begin{bmatrix} 2 \\ 4 \end{bmatrix} [7 \ 8].$$

Оказывается, что, несмотря на математическую эквивалентность всех трех версий, их реальная производительность может существенно различаться из-за разных способов доступа к памяти. Мы продолжим обсуждение этого вопроса в § 1.4, а пока рассмотрим более детально обозначенные выше три подхода к умножению матриц. Это позволит нам поупражняться в конструировании различных вариантов алгоритмов и записи их с использованием введенных обозначений.

### 1.1.14. Умножение матриц с использованием скалярных произведений

Пусть  $A \in \mathbf{R}^{m \times r}$ ,  $B \in \mathbf{R}^{r \times n}$ , и мы хотим вычислить  $C = AB$ . Стандартная процедура заключается в последовательном вычислении элементов  $C$  в порядке слева направо сверху вниз:

**Алгоритм 1.1.6 (Умножение матриц: версия, использующая скалярное произведение).** По  $A \in \mathbf{R}^{m \times n}$  и  $B \in \mathbf{R}^{r \times n}$  алгоритм вычисляет  $C = AB$ .

```
function: C = matmat. ijk(A, B)
    m = rows(A); r = cols(); n = cols(B)
    C(1:m, 1:n) = 0
    for i = 1:m
        for j = 1:n
            for k = 1:r
                C(i, j) = C(i, j) + A(i, k) B(k, j)
            end
        end
    end
end matmat. ijk
```

В этом процедуре  $c_{ij}$  вычисляется как скалярное произведение  $i$ -й строки матрицы  $A$  и  $j$ -го столбца матрицы  $B$ . На языке блочных матриц, если мы имеем разбиения

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ a_m^T \end{bmatrix}, \quad a_k \in \mathbf{R}^r$$

и

$$B = [b_1, \dots, b_n], \quad b_k \in \mathbf{R}^r,$$

то алгоритм 1.1.6 описывается на уровне 1 как

```
for i = 1:m
    for j = 1:n
        c_{ij} = a_i^T b_j
    end
end
```

Заметим, что задача цикла по  $j$  – вычислить  $i$ -ю строку матрицы  $C$ . Мы можем подчеркнуть это, записав

```
for i = 1:m
    c_i^T = a_i^T B
end
```

где  $c_i^T$  –  $i$ -я строка  $C$ . Это описание алгоритма 1.1.6 в терминах матрично-векторных произведений можно рассматривать как описание уровня 2. Конечно, на самом верхнем уровне останется запись  $C = AB$ , не содержащая ни циклов, ни индексов.

Правильный выбор уровня описания важен как для изложения теоретических результатов, так и для разработки программного обеспечения.

### 1.1.15. Умножение матриц с использованием гахру

Предположим, что  $A$ ,  $B$  и  $C$  разбиты на столбцы:

$$\begin{aligned} A &= [a_1, \dots, a_r], a_j \in \mathbf{R}^m, \\ B &= [b_1, \dots, b_n], b_j \in \mathbf{R}^r, \\ C &= [c_1, \dots, c_n], c_j \in \mathbf{R}^m \end{aligned}$$

и мы желаем вычислить  $C = AB$ . Поскольку

$$c_j = \sum_{k=1}^r b_{kj} a_k, j = 1:n,$$

мы видим, что каждый столбец  $C$  является линейной комбинацией столбцов  $A$ . Вычисление этих сумм можно организовать как последовательность операций *gaxpy*:

**Алгоритм 1.1.7 (Умножение матриц: *gaxpy*-версия).** По  $A \in \mathbf{R}^{m \times r}$  и  $B \in \mathbf{R}^{r \times n}$  следующий алгоритм вычисляет  $C = AB$ .

```
function: C = matmat.jki(A, B)
    m = rows(A); n = cols(B); r = cols(A)
    C(1:m, 1:n) = 0
    for j = 1:n
        for k = 1:r
            for i = 1:m
                C(i, j) = C(i, j) + A(i, k) B(k, j)
            end
        end
    end
end matmat. jki
```

Мы вывели **matmat.jki**, разбив по столбцам матрицы в равенстве  $C = AB$ , но можно было получить его также изменением порядка циклов в функции **matmat.ijk**. Ясно, что “*jkI*”-алгоритм состоит из операций *gaxpy*, поскольку его можно записать в виде

```
C(1:m, 1:n) = 0
for j = 1:n
    C(:, j) = gaxpy(A, B(:, j), (:, j))
end
```

### 1.1.16. Умножение матриц с использованием внешних произведений

Давайте проделаем перестановку циклов еще раз, а затем проанализируем ее смысл в терминах линейной алгебры.

**Алгоритм 1.1.8 (Умножение матриц: версия, использующая внешние произведения).** По  $A \in \mathbf{R}^{m \times r}$  и  $B \in \mathbf{R}^{r \times n}$  алгоритм вычисляет  $C = AB$ .

```
function: C = matmat.kji (A, B)
    m = rows(A); n = cols(B); r = cols(A)
    C(1:m, 1:n) = 0
    for k = 1:r
        for j = 1:n
            for i = 1:m
                C(i, j) = C(i, j) + A(i, k) B(k, j)
            end
        end
    end
end matmat. kji
```

В этой “ $kji$ ”-формулировке происходит вычисление всех элементов  $c_{ij}$  одновременно. Для данного  $k$  два внутренних цикла осуществляют модификацию матрицы  $C$  внешним произведением  $C \leftarrow C + a_k b_k^T$ , где

$$A = [a_1, \dots, a_r], a_k \in \mathbb{R}^m \quad (1.1.3)$$

и

$$B = \begin{bmatrix} b_1^T \\ \vdots \\ b_r^T \end{bmatrix}, b_k \in \mathbb{R}^n. \quad (1.1.4)$$

Внутренний цикл в функции **matmat. kji** выполняет операцию **saxpy**. Именно, к  $j$ -му столбцу матрицы  $C$  прибавляется вектор, кратный  $a_k$ .

### 1.1.17. Перестановки циклов

Двойной цикл в алгоритме умножения матрицы на вектор может быть переупорядочен  $2! = 2$  способами. Соответственно имеется  $3! = 6$  способов упорядочения тройного цикла в алгоритме умножения матрицы на матрицу. Три варианта уже были в деталях изложены выше:  $ijk$ ,  $jki$  и  $kji$ . Каждый из шести возможных вариантов характеризуется своим способом доступа к данным и доминирующей операцией (скалярное произведение, **saxpy**). Эти сведения собраны в табл. 1.1.1. Какой же вариант предпочтительнее? Ответ на этот вопрос зависит от архитектуры компьютера, на котором будет реализован алгоритм. Подробнее это обсуждается в § 1.4.

Таблица 1.1.1. Матричное умножение; упорядочение и свойства

Порядок цикла	Внутренний цикл	Средний цикл	Внутренний цикл Доступ к данным
$ijk$	скалярное произведение	вектор $\times$ матрица	$A$ на строку, $B$ на столбец
$jik$	скалярное произведение	матрица $\times$ вектор	$A$ на строку, $B$ на столбец
$ikj$	<b>saxpy</b>	строчное <b>gaxpy</b>	$B$ на строку
$jki$	<b>saxpy</b>	столбцовое <b>gaxpy</b>	$A$ на столбец
$kij$	<b>saxpy</b>	строка внешнего произведения	$B$ на строку
$kji$	<b>saxpy</b>	столбец внешнего произведения	$A$ на столбец

### 1.1.18. О матричных соотношениях

Изучая умножение матриц с использованием внешних произведений, мы по существу установили соотношение

$$AB = \sum_{k=1}^r a_k b_k^T,$$

где  $a_k$  и  $b_k$  определены в (1.1.3) и (1.1.4).

В последующих главах мы получим очень много матричных соотношений. Иногда они выводятся алгоритмически, как это было в приведенном выше примере с внешним произведением, а в других случаях их приходится доказывать на уровне “ $ij$ ”-компонент. Как пример этого стиля доказательств приведем важный результат, характеризующий транспонирование произведения матриц.

**Теорема 1.1.1.** Если  $A \in \mathbb{R}^{m \times r}$  и  $B \in \mathbb{R}^{r \times n}$ , то  $(AB)^T = B^T A^T$ .

*Доказательство.* Если  $C = (AB)^T$ , то

$$c_{ij} = [(AB)^T]_{ij} = (AB)_{ji} = \sum_{k=1}^r a_{jk} b_{ki}.$$

С другой стороны, если  $D = B^T A^T$ , то

$$d_{ij} = [B^T A^T]_{ij} = \sum_{k=1}^r (B^T)_{ik} (A^T)_{kj} = \sum_{k=1}^r b_{ki} a_{jk},$$

так что  $C = D$ .  $\square$

Как свидетельствует это доказательство, погружение на “ $ijk$ ”-уровень отнюдь не тождественно глубокому проникновению в материал. Однако иногда это единственный способ получить алгебраический результат высокого уровня.

### 1.1.19. Об описании алгоритмов

На протяжении этой книги нам понадобится описывать алгоритмы на разных уровнях детализации. В этом разделе все алгоритмы были записаны формально, как функции. В качестве примера более вольного стиля записи алгоритмов приведем процедуру, вычисляющую матричное произведение  $C = A^T B$ .

**Алгоритм 1.1.9.** По двум  $m \times n$ -матрицам  $A$  и  $B$  алгоритм вычисляет произведение  $C = A^T B$ .

```
for i = 1:n
    for j = 1:n
        C(i, j) = A(1:m, i)^T B(1:m, j)
    end
end
```

### 1.1.20. Комплексные матрицы

Наше внимание в этой книге сфокусировано на вычислениях с вещественными матрицами – для упрощения обозначений, а также ввиду того, что большинство реальных задач имеет дело с вещественными данными. Однако в последующих разделах мы будем работать и с комплексными матрицами – там, где это уместно.

Для начала введем некоторые обозначения. Векторное пространство комплексных  $m \times n$ -матриц обозначается через  $\mathbf{C}^{m \times n}$ . Умножение на скаляр, сложение и перемножение комплексных матриц такое же, как и в вещественном случае. Однако транспонирование превращается в *сопряженное транспонирование*:

$$C = A^H \Rightarrow c_{ij} = \bar{a}_{ji}.$$

Векторное пространство комплексных  $n$ -векторов обозначается  $\mathbf{C}^n$ . Скалярное произведение комплексных  $n$ -векторов вычисляется по правилу

$$s = x^H y = \sum_{i=1}^n \bar{x}_i y_i.$$

Наконец, если  $A = B + iC \in \mathbb{C}^m \times n$ , то мы будем обозначать вещественную и мнимую части  $A$  через  $\operatorname{Re}(A) = B$  и  $\operatorname{Im}(A) = C$  соответственно.

### Задачи

**1.1.1.** Пусть даны  $A \in \mathbb{R}^{n \times n}$  и  $x \in \mathbb{R}^n$ . Используя `saxpy`, напишите алгоритм, вычисляющий первый столбец матрицы  $M = (A - x_1 I) \dots (A - x_r I)$ .

**1.1.2.** При умножении  $2 \times 2$ -матрицы обычным способом требуется вычислить восемь произведений:  $a_{11} b_{11}, a_{11} b_{12}, a_{21} b_{11}, a_{21} b_{12}, a_{12} b_{21}, a_{12} b_{22}, a_{22} b_{21}, a_{22} b_{22}$ . Составьте таблицу, показывающую порядок выполнения этих произведений для всех шести вариантов следования циклов в алгоритме умножения матриц:  $ijk, jik, kij, ikj, jki, kji$ .

**1.1.3.** Используя функции `dot` и `saxpy`, постройте алгоритм, вычисляющий  $C = (xy^T)^k$ , где  $x$  и  $y$  —  $n$ -векторы.

**1.1.4.** Пусть имеются вещественные  $n \times n$ -матрицы  $S, D, E, F$ . Укажите способ вычисления вещественных матриц  $A$  и  $B$ , использующий только три обращения к функции `matmat`. **jki** с вещественными  $n \times n$ -аргументами, где  $(A + iB) = (C + iD)(+iF)$ . Подсказка. Вычислите  $W = (C + D)(E - F)$ .

### Замечания и литература к § 1.1

Формальное описание языка Matlab, которым мы пользуемся (с некоторыми вольностями) для записи алгоритмов, можно найти в книгах:

Coleman T. F. and Van Loan C. (1988). *Handbook for Matrix Computations*, SIAM Publications, Philadelphia, PA.

Moler C. B., Little J. N., and Bangert S. (1987). *PC-Matlab User's Guide*, The Math Works Inc., 20 N. Main St., Sherborn, Mass.

Подчеркнем еще раз, что приводимые в этой книге алгоритмы не являются готовым программным продуктом. Если подходить к делу всерьез, то написание качественной программы на основе любого из наших описаний алгоритмов требует длительной и напряженной работы. Даже реализация процедур низкого уровня, таких как `saxpy` и `gaxpy`, требует времени и внимания. Подробнее эти вопросы рассмотрены в статьях:

Dongarra J., DuCroz J., Hammarling S., and Hanson R. J. (1988). "An Extended Set of Fortran Basic Linear Algebra Subprograms", *ACM Trans. Math. Soft.* 14, 1–17.

Dongarra J., DuCroz J., Hammarling S., and Hanson R. J. (1988). "Algorithm 656 An Extended Set of Fortran Basic Linear Algebra Subprograms: Model Implementation and Test Programs", *ACM Trans. Math. Soft.* 14, 18–32.

Dongarra J., DuCroz J., Duff I. S., and Hammarling S. (1988). "A Set of Level 3 Basic Linear Algebra Subprograms", Argonne National Laboratory Report, ANL-MCS-TM-88.

Lawson C. L., Hanson R. J., Kincaid D. R., and Krogh F. T. (1979a). "Basic Linear Algebra Subprograms for FORTRAN Usage", *ACM Trans. Math. Soft.* 5, 308–23.

Lawson C. L., Hanson R. J., Kincaid D. R., and Krogh F. T. (1979b). "Algorithm 539, Basic Linear Algebra Subprograms for FORTRAN Usage", *ACM Trans. Math. Soft.* 5, 324–25.

Мы также рекомендуем книгу

Rice J. R. (1981). *Matrix Computations and Mathematical Software*, Academic Press, New York. [Имеется русский перевод: Райс Дж. Матричные вычисления и математическое обеспечение. — М.: Мир, 1984.]

Влияние различных упорядочений циклов на производительность детально рассмотрено в великолепном обзоре

Dongarra J. J., Gustavson F. G., and Karp A. (1984). "Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine", *SIAM Review* 26, 91–112.

## 1.2. Учет структуры матриц

Эффективность заданного матричного алгоритма зависит от многих характеристик. В этом разделе мы рассмотрим наиболее очевидную из них – количество арифметических операций и объем памяти, требуемые процедурой. Менее очевидны такие характеристики, как способ доступа к памяти, шаг выборки и еще целый сонм издержек, возникающих при пересылке данных. Эти вопросы рассматриваются, в частности, в § 1.4.

Мы продолжаем обкатывать новые ключевые идеи на примерах умножения матрицы на матрицу и матрицы на вектор. Для демонстрации учета структуры матриц мы избрали свойства ленточности и симметричности. Ленточные матрицы содержат много нулевых элементов, поэтому неудивительно, что умножение ленточных матриц позволяет уменьшить по сравнению с общим случаем арифметические затраты, и объем требуемой памяти. В этом контексте мы обсуждаем структуры данных и понятие «флопа».

Другой пример учета особенностей структуры доставляют симметричные матрицы. Решение систем линейных уравнений и задач на собственные значения с симметричными матрицами играет видную роль в матричных вычислениях, поэтому важно приобрести хороший навык работы с ними.

Мы заканчиваем этот раздел замечанием о записи результатов вычислений на место входных данных – это еще один прием для контроля объема хранимой информации.

### 1.2.1. Ленточные матрицы и обозначения для них

Будем говорить, что матрица  $A \in \mathbf{R}^{m \times n}$  имеет *нижнюю ширину ленты*  $p$ , если  $a_{ij} = 0$  для  $i > j + p$ , и *верхнюю ширину ленты*  $q$ , если из  $j > i + q$  следует  $a_{ij} = 0$ . Вот пример матрицы  $8 \times 5$  с нижней шириной ленты 1 и верхней шириной ленты 2:

$$\begin{bmatrix} \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Здесь крестики означают произвольные ненулевые (вообще говоря) элементы матрицы. Такие обозначения очень удобны для показа структуры размещения

Таблица 1.2.1.

Тип матрицы	Нижняя ширина ленты	Верхняя ширина ленты
диагональная	0	0
верхняя треугольная	0	$n - 1$
нижняя треугольная	$m - 1$	0
трехдиагональная	1	1
верхняя двухдиагональная	0	1
нижняя двухдиагональная	1	0
верхняя хессенбергова	1	$n - 1$
нижняя хессенбергова	$m - 1$	1

нулевых и ненулевых элементов в матрице, и мы будем интенсивно их использовать. Некоторые часто встречающиеся типы ленточных матриц собраны в табл. 1.2.1.

### 1.2.2. Действия с диагональными матрицами

Для диагональных матриц у нас есть специальные обозначения. Если  $D \in \mathbb{R}^{m \times n}$ , то

$$D = \text{diag}(d_1, \dots, d_q), q = \min\{m, n\} \Leftrightarrow d_i = d_{ii}.$$

Если  $D$  – диагональная,  $A$  – произвольная матрица, то произведение  $DA$  масштабирует  $A$  по строкам, а  $AD$  – по столбцам.

Пусть  $y = Dx$ , где  $x \in \mathbb{R}^n$ ,  $y \in \mathbb{R}^m$ . Тогда  $y$  является покомпонентным произведением векторов:  $y = d.*x$ . Здесь вектор  $d \in \mathbb{R}^n$  составлен из диагональных элементов матрицы  $D$ .

### 1.2.3. Умножение треугольных матриц

Мы начнем осваивать технику работы с ленточными матрицами на примере матричного умножения  $C = AB$ , где и  $A$ , и  $B$  – верхние треугольные  $n \times n$ -матрицы. Наводящие соображения нам подскажет следующий пример с  $n = 3$ :

$$C = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ 0 & a_{22}b_{22} & a_{22}b_{23} + a_{23}b_{33} \\ 0 & 0 & a_{33}b_{33} \end{bmatrix}.$$

Этот пример позволяет предположить, что результат  $C$  является верхней треугольной матрицей, элементы которой вычисляются как укороченные скалярные произведения. В самом деле, поскольку  $a_{ik}b_{kj} = 0$  при  $k < i$  или  $j < k$ , мы видим, что  $c_{ij} = \sum_{k=1}^j a_{ik}b_{kj}$ , откуда вытекает

**Алгоритм 1.2.1.** По двум верхним треугольным  $n \times n$ -матрицам  $A$  и  $B$  следующий алгоритм вычисляет  $C = AB$ .

```

 $C(1:n, 1:n) = 0$ 
for  $i = 1:n$ 
  for  $j = 1:n$ 
    for  $k = i:j$ 
       $C(i, j) = C(i, j) + A(i, k)B(k, j)$ 
    end
  end
end
```

Чтобы дать количественную оценку достигнутой в этом алгоритме экономии, нам нужны инструменты для измерения объема вычислительной работы.

### 1.2.4. Флопы

Очевидно, умножение верхних треугольных матриц требует меньшего количества арифметических операций, чем умножение произвольных матриц. Один из

способов количественной характеристики разности дает понятие *флопа*. Флоп<sup>1)</sup> – это одна операция с плавающей точкой (floating point operation). Скалярное произведение или операция *saxpy* размерности  $n$  содержат  $2n$  флопов, поскольку каждая из них состоит из  $n$  умножений и  $n$  сложений.

Умножение матрицы на вектор  $z = Ax$ , где  $A \in \mathbb{R}^{m \times n}$ , так же, как и модификация матрицы внешним произведением векторов  $C = A + xy^T$ , содержит  $2mn$  флопов. Умножение матриц  $C = AB$ , где  $A \in \mathbb{R}^{m \times r}$  и  $B \in \mathbb{R}^{r \times n}$ , содержит  $2mnr$  флопов.

Для оценки количества флопов обычно суммируют арифметические затраты для наиболее глубоко вложенных операторов алгоритма. Для умножения матриц таким образом является оператор

$$C(i, j) = C(i, j) + A(i, k)B(k, j),$$

который содержит два флопа и выполняется  $mnr$  раз, как показывает подсчет количества проходов циклов. Отсюда мы заключаем, что умножение произвольных матриц требует  $2mnr$  флопов.

Теперь найдем объем вычислительной работы для алгоритма 1.2.1. Заметим, что на вычисление  $c_{ij}(i \leq j)$  затрачивается  $2(j - i + 1)$  флопов. Используя оценки

$$\sum_{p=1}^q p = \frac{q(q+1)}{2} \approx \frac{q^2}{2}$$

и

$$\sum_{p=1}^q p^2 = \frac{q^3}{3} + \frac{q^2}{2} + \frac{q}{6} \approx \frac{q^3}{3},$$

мы находим, что умножение треугольных матриц требует примерно  $n^3/3$  флопов:

$$\sum_{i=1}^n \sum_{j=1}^n 2(j - i + 1) = \sum_{i=1}^n \sum_{j=1}^{n-i+1} 2j \approx \sum_{i=1}^n \frac{2(n-i+1)^2}{2} = \sum_{i=1}^n i^2 \approx \frac{n^3}{3}.$$

Мы отбрасываем слагаемые низших порядков, поскольку их включение ничего по существу не добавляет к той информации, которую уже несет оценка количества флопов. Например, для алгоритма 1.2.1 точный подсчет количества операций дает  $n^3/3 + n^2 + 2n/3$  флопов. Для больших  $n$  (в типичном случае именно эта ситуация представляет интерес) точная формула не дает никакой новой полезной информации по сравнению с нашей оценкой  $n^3/3$ .

Оценка количества флопов – это по необходимости грубый подход к измерению эффективности программ, поскольку он игнорирует затраты на индексацию, обмены с памятью и многочисленные прочие издержки, возникающие при выполнении программы на компьютере. Из сравнения количества флопов не следует делать далеко идущих выводов; так, мы не можем заключить, что умножение треугольных матриц в шесть раз быстрее, чем умножение квадратных матриц. Считать флопы – это лишь «быстрая и грязная» бухгалтерия, ухватывающая лишь одно из нескольких измерений эффективности. Как мы увидим, для различных высокопроизводитель-

<sup>1)</sup> В первом издании этой книги мы определили флоп как объем вычислительной работы, затрачиваемый на операцию вида  $a_{ij} = a_{ij} + a_{ik}a_{kj}$ , т. е. сложение и умножение с плавающей точкой плюс некоторые издержки на индексацию. Таким образом, один «старый» флоп содержит два «новых» флопа». Определяя флоп как единичную операцию с плавающей точкой, мы делаем выбор в пользу большей точности при измерении арифметической сложности алгоритмов. Кроме того, новые флопы вытеснили старые флопы из работ по суперкомпьютерам, к великой радости производителей, чьи машины благодаря новой терминологии в однократно удвоили свою скорость! После некоторых мучительных раздумий мы решили присоединиться к этой перемене.

ных архитектур эффективность такого алгоритма, как умножение матриц, в существенно большей степени зависит от коммуникационных издержек, и более точное измерение эффективности могло бы потребовать учета количества операций схемы и длины векторных operandов.

### 1.2.5. Еще раз об использовании двоеточия

Скалярное произведение, вычисляемое циклом по  $k$  в алгоритме 1.2.1, можно записать компактно, если обобщить введенное в § 1.1.9 использование двоеточия. Пусть  $A \in \mathbb{R}^{m \times n}$  и целые числа  $p, q$  и  $r$  удовлетворяют неравенствам  $1 \leq p \leq q \leq n$  и  $1 \leq r \leq m$ . Тогда определим

$$A(r, p:q) = [a_{rp}, \dots, a_{rq}] \in \mathbb{R}^{1 \times (q-p+1)}.$$

Аналогично, если  $1 \leq p \leq q \leq m$  и  $1 \leq c \leq n$ , то

$$A(p:q, c) = \begin{bmatrix} a_{pc} \\ \vdots \\ a_{qc} \end{bmatrix} \in \mathbb{R}^{q-p+1}.$$

С использованием этих обозначений алгоритм 1.2.1. перепишется в виде:

```
C(1:n, 1:n) = 0
for i = 1:n
    for j = 1:n
        C(i, j) = C(i, j) + A(i, i:j) B(i:j, j)
    end
end
```

Упомянем еще одно добавление к нашим обозначениям. При использовании двоеточия допускаются отрицательные приращения. Так, если  $x$  и  $y$  –  $n$ -векторы, то  $s = x^T y(n:-1:1)$  есть сумма

$$s = \sum_{i=1}^n x_i y_{n-i+1},$$

т. е. свертка векторов  $x$  и  $y$ .

### 1.2.6. Хранение ленточных матриц

Пусть  $A \in \mathbb{R}^{n \times n}$  имеет нижнюю ширину ленты  $p$ , верхнюю ширину ленты  $q$ , и предположим, что  $p$  и  $q$  существенно меньше, чем  $n$ . Такая матрица может быть размещена в  $(p+q+1) \times n$ -массиве  $A.band$  при соглашении, что

$$a_{ij} = A.band(i-j+q+1, j) \quad (1.2.1)$$

для всех пар  $(i, j)$ , попадающих внутрь ленты. Так, если  $n = 5$ ,  $p = 1$  и  $q = 2$ , как в § 1.2.1, то мы имеем:

$$A.band = \begin{bmatrix} 0 & 0 & a_{13} & a_{24} & a_{35} \\ 0 & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & 0 \end{bmatrix}.$$

Здесь «0» – неиспользуемые элементы массива. С такой структурой данных алгоритм умножения матрицы на вектор с выборкой по столбцу может быть реализован

следующим образом:

**Алгоритм 1.2.2.** Пусть  $A \in \mathbb{R}^{n \times n}$  имеет нижнюю ширину ленты  $p$ , верхнюю ширину ленты  $q$  и хранится в формате  $A.band$  (1.2.1). Если  $x \in \mathbb{R}^n$ , то следующий алгоритм вычисляет  $z = Ax$ .

```

 $z(1:n) = 0$ 
for  $j = 1:n$ 
     $z_{top} = \max(1, j-q); z_{bot} = \min(n, j+p)$ 
     $a_{top} = \max(1, q+2-j); a_{bot} = a_{top} + z_{bot} - z_{top}$ 
     $z(z_{top}:z_{bot}) = z(z_{top}:z_{bot}) + x(j) A.band(a_{top}:a_{bot}, j)$ 
end
```

Заметим, что, храня матрицу  $A$  по столбцам в массиве  $A.band$ , мы получаем процедуру, использующую схору с выборкой по столбцам. Действительно, алгоритм (1.2.2) получается из **matvec.ji**, если учесть, что в каждой операции схору участвует вектор, содержащий лишь небольшое количество ненулевых элементов; для нахождения позиций этих элементов используется целочисленная арифметика. В результате алгоритм содержит примерно  $2n(p+q+1)$  флопов, в предположении что  $p$  и  $q$  много меньше  $n$ .

## 1.2.7. Симметрия

Будем говорить, что матрица  $A \in \mathbb{R}^{n \times n}$  *симметрична*, если  $A^T = A$ . Так, матрица

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

симметрична. Объем требуемой для хранения памяти можно уменьшить в два раза, если хранить только нижний треугольник матрицы  $A$  в одном векторе; в нашем примере  $A.vec = [1 2 3 4 5 6]$ . Вообще, в этом структуре данных элемент  $a_{ij}$  хранится следующим образом:

$$a_{ij} = A.vec((j-1)n - j(j-1)/2 + i)(i \geq j). \quad (1.2.2)$$

Посмотрим, как будет выглядеть алгоритм умножения матрицы на вектор, если матрица  $A$  размещена в массиве  $A.vec$ .

**Алгоритм 1.2.3.** Пусть  $A \in \mathbb{R}^{n \times n}$  симметрична и хранится в массиве  $A.vec$ , согласно (1.2.2). По  $x \in \mathbb{R}^n$  следующий алгоритм вычисляет  $z = Ax$ .

```

 $z(1:n) = 0$ 
for  $j = 1:n$ 
    for  $i = 1:j-1$ 
         $z(i) = z(i) + A.vec((i-1)n - i(i-1)/2 + j)x(j)$ 
    end
    for  $i = j:n$ 
         $z(i) = z(i) + A.vec((j-1)n - j(j-1)/2 + i)x(j)$ 
    end
end
```

Этот алгоритм затрачивает те же самые  $2n^2$  флопов, что и умножение произвольной матрицы на вектор. Обратите внимание, какой неуклюжей стала запись вследствие использования специальной структуры данных для хранения матрицы.

## 1.2.8. Хранение по диагоналям

Если в алгоритме осуществляется выборка элементов массивов, то необходимо стараться так организовать вычисления, чтобы внутренние циклы обращались к данным, расположенным в последовательных ячейках памяти. Например, для двумерных фортрановских циклов это сводится к написанию процедур таким образом, чтобы доступ к элементам матриц шел по столбцам, как обсуждалось в § 1.1. Проблема с алгоритмом 1.2.3 в том, что первый цикл по  $i$  не обращается к последовательным элементам массива. Это недостаток избранной структуры данных.

Чтобы обойти эту проблему, рассмотрим другую схему компактного хранения симметричных матриц, в которой матрица хранится по диагоналям. Если

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix},$$

то при хранении по диагоналям  $A$  представляется вектором

$$A.diag = [1\ 4\ 6\ 2\ 5\ 3].$$

Вообще, если  $i \geq j$ , то

$$a_{i+k,i} = A.diag(i + nk - k(k-1)/2) \quad (k \geq 0). \quad (1.2.3)$$

Введем обозначения, упрощающие описание использования этой структуры данных для умножения матрицы на вектор.

Для  $A \in \mathbb{R}^{m \times n}$  пусть  $D(A, k) \in \mathbb{R}^{m \times n}$  означает матрицу, содержащую  $k$ -ю диагональ  $A$ :

$$[D(A, k)]_{ij} = \begin{cases} a_{ij}, & j = i + k, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n, \\ 0 & \text{в противном случае.} \end{cases}$$

Так,

$$\begin{aligned} A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} &= \underbrace{\begin{bmatrix} 0 & 0 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{D(A, 2)} + \underbrace{\begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 0 \end{bmatrix}}_{D(A, 1)} + \\ &+ \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}}_{D(A, 0)} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 5 & 0 \end{bmatrix}}_{D(A, -1)} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix}}_{D(A, -2)}. \end{aligned}$$

Возвращаясь к нашему способу хранения по диагоналям, мы видим, что ненулевые элементы матриц  $D(A, 0), D(A, 1), \dots, D(A, n-1)$  хранятся последовательно в векторе  $A.diag$  согласно (1.2.3). Умножение матрицы на вектор  $z = Ax$  может быть организовано следующим образом:

$$z = D(A, 0)x + \sum_{k=1}^{n-1} (D(A, k) + D(A, k)^T)x.$$

Произведения  $D(A, k)x$  и  $D(A, k)^Tx$  обращаются к последовательно хранящимся данным. Проработав детали, мы получаем

**Алгоритм 1.2.4.** Пусть  $A \in \mathbb{R}^{n \times n}$  симметрична и хранится в массиве  $A.diag$  согласно (1.2.3). По  $x \in \mathbb{R}^n$  следующий алгоритм вычисляет  $z = Ax$ .

```

for  $i = 1:n$ 
     $z(i) = A.diag(i) x(i)$ 
end
for  $k = 1:n - 1$ 
     $t = nk - k(k - 1)/2$ 
     $\{z = z + D(A, k)x\}$ 
    for  $i = 1:n - k$ 
         $z(i) = z(i) + A.diag(i + t)x(i + k)$ 
    end
     $\{z = z + D(A, k)^T x\}$ 
    for  $i = 1:n - k$ 
         $z(i + k) = z(i + k) + A.diag(i + t)x(i)$ 
    end
end

```

Здесь внутренние циклы выполняют не операцию *saxpy*, а *покомпонентное произведение* векторов (см. § 1.1.4). Заметим, что во внутренних циклах осуществляется доступ к последовательно расположенным в памяти данным.

Для ясности мы сопроводили приведенный выше алгоритм комментариями. Комментарии заключены в фигурные скобки.

### 1.2.9. Запись поверх входных данных

Основным мотивом алгоритмов из этого раздела было экономное использование памяти. Еще один способ ограничить объем требуемой памяти – запись результатов вычислений на место входных данных. Рассмотрим задачу перемножения  $n \times n$ -матриц  $C = AB$  с условием, что получающаяся на выходе матрица  $C$  должна быть записана на место входной матрицы  $B$ . Будем отталкиваться от алгоритма без перезаписи, полученного в § 1.1.15:

```

 $C(1:n, 1:n) = 0$ 
for  $j = 1:n$ 
    for  $k = 1:n$ 
        for  $i = 1:n$ 
             $C(i, j) = C(i, j) + A(i, k)B(k, j)$ 
        end
    end
end

```

Здесь столбец  $B(:, j)$  нужен на протяжении всего цикла по  $k$ , поэтому корректная версия с перезаписью не может быть получена простой подстановкой  $B$  вместо  $C$  и опущением инициализации  $C = 0$ . Для хранения  $j$ -го столбца матрицы  $C$  требуется рабочий массив длины  $n$ , прежде чем можно будет писать поверх  $B(:, j)$ :

```

for  $j = 1:n$ 
     $w(1:n) = 0$ 
    for  $k = 1:n$ 
        for  $i = 1:n$ 
             $w(i) = w(i) + A(i, k)B(k, j)$ 
        end
    end
     $B(:, j) = w(:)$ 
end

```

В этом примере требуется рабочий массив линейных размеров (по  $n$ ). Обычно это несущественно для процедур, содержащих двумерные массивы того же порядка.

### Задачи

**1.2.1.** Постройте алгоритм, записывающий  $A^2$  на место  $A$ , где матрица  $A$ : (а) – верхняя треугольная, (б) – квадратная. Добивайтесь наименьшего объема дополнительной памяти.

**1.2.2.** Пусть  $A \in \mathbb{R}^{n \times n}$  – верхняя хессенбергова матрица и  $\lambda_1, \dots, \lambda_r$  – данные числа. Постройте использующий схору алгоритм для вычисления первого столбца матрицы  $M = (A - \lambda_1 I) \dots (A - \lambda_r I)$ .

**1.2.3.** Постройте использующий схору с выборкой по столбцам алгоритм для умножения  $n \times n$ -матриц  $C = AB$ , где  $A$  – верхняя,  $B$  – нижняя треугольная матрица.

**1.2.4.** Обобщите алгоритм 1.2.2 на случай прямоугольных ленточных матриц. Не забудьте описать выбранную структуру данных.

**1.2.5.** Матрица  $A \in \mathbb{R}^{n \times n}$  называется *эрмитовой*, если  $A^H = A$ . Если  $A = B + iC$ , то, как легко показать,  $B^T = B$  и  $C^T = -C$ . Пусть  $A$  размещена в массиве  $A.herm$  так, что  $A.herm(i,j)$  содержит  $b_{ij}$  при  $i \geq j$  и  $c_{ij}$  при  $j > i$ . Используя эту структуру данных, напишите процедуру умножения матрицы на вектор, вычисляющую  $\operatorname{Re}(z)$  и  $\operatorname{Im}(z)$  по  $\operatorname{Re}(x)$  и  $\operatorname{Im}(x)$ , так что  $z = Ax$ .

**1.2.6.** Пусть  $X \in \mathbb{R}^{n \times p}$  и  $A \in \mathbb{R}^{n \times n}$ , где матрица  $A$  симметрична и хранится по диагоналям. Постройте алгоритм, вычисляющий  $Y = X^TAX$  и записывающий результат по диагоналям. Используйте отдельные массивы для хранения матриц  $A$  и  $Y$ .

### Замечания и литература к § 1.2

Детальное описание структур данных для хранения симметричных и/или ленточных матриц имеется в руководстве по пакету Linpack.

## 1.3. Блочные матрицы и алгоритмы

Умение непринужденно работать с блочными матрицами крайне необходимо для матричных вычислений. Использование блочных обозначений упрощает вывод многих важных алгоритмов. Кроме того, все более возрастает роль блочных алгоритмов в вычислениях на суперкомпьютерах. Здесь под блочными алгоритмами мы подразумеваем по существу такие алгоритмы, которые содержат много операций умножения матрицы на матрицу. Следует ожидать, что эти алгоритмы будут более эффективны по сравнению с теми, которые работают на скалярном уровне, поскольку на каждый обмен данными будет приходиться большее количество вычислений. К примеру, умножение  $k \times k$ -матриц занимает  $2k^3$  флопов, вовлекая лишь  $2k^2$  элементов данных. В § 1.4 мы увидим, что при больших  $k$  издержки обмена данными становятся относительно менее существенными.

### 1.3.1. Обозначения для блочных матриц

Разбиения матриц на строки и столбцы суть частные случаи разбиения на блоки. В общем случае разбиение  $m \times n$ -матрицы  $A$  на блоки выглядит следующим образом:

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{pq} \end{bmatrix} \begin{matrix} m_1 \\ \vdots \\ m_p \end{matrix} .$$

$$\begin{matrix} n_1 & & n_q \end{matrix} .$$

Здесь  $m_1 + \dots + m_p = m$ ,  $n_1 + \dots + n_q = n$ ,  $A_{ij}$  означает  $(i, j)$ -й блок, или подматрицу. Блок  $A_{ij}$  имеет размерность  $m_i$  на  $n_j$ , и мы будем говорить, что  $A = (A_{ij})$  есть  $p \times q$ -блочная матрица.

### 1.3.2. Операции с блочными матрицами

Блочные матрицы можно складывать так же, как и обычные матрицы, при условии соблюдения соответствия размерностей блоков. Например, если

$$B = \begin{bmatrix} B_{11} & \cdots & B_{1q} \\ \vdots & & \vdots \\ B_{p1} & \cdots & B_{pq} \\ n_1 & & n_q \end{bmatrix} \begin{matrix} m_1 \\ \\ m_p \end{matrix},$$

то мы говорим, что матрица  $B$  разбита на блоки *согласованно* с  $A$ , и их сумма  $C = A + B$  есть  $p \times q$ -блочная матрица  $C = (C_{ij})$ , где  $C_{ij} = A_{ij} + B_{ij}$ .

Умножение блочных матриц устроено чуточку хитрее. Вот ключевой результат:

**Лемма 1.3.1.** Пусть  $A \in \mathbb{R}^{m \times r}$  и  $B \in \mathbb{R}^{r \times n}$  разбиты на блоки следующим образом:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ r_1 & r_2 \end{bmatrix} \begin{matrix} m_1 \\ m_2 \end{matrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ n_1 & n_2 \end{bmatrix} \begin{matrix} r_1 \\ r_2 \end{matrix}.$$

Тогда будем иметь

$$AB = C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \\ n_1 & n_2 \end{bmatrix} \begin{matrix} m_1 \\ m_2 \end{matrix},$$

где  $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$  для  $i = 1:2$  и  $j = 1:2$ .

*Доказательство.* Доказательство представляет собой упражнение на использование индексов.

$$\begin{aligned} [C_{ij}] &= \sum_{k=1}^r a_{(i-1)m_1+p,k} b_{k,(j-1)n_1+q} = \\ &= \sum_{k=1}^{r_1} a_{(i-1)m_1+p,k} b_{k,(j-1)n_1+q} + \\ &\quad + \sum_{k=r_1+1}^r a_{(i-1)m_1+p,k} b_{k,(j-1)n_1+q} = \\ &= \sum_{k=1}^{r_1} [A_{i1}]_{pk} [B_{1j}]_{kq} + \sum_{k=1}^{r_2} [A_{i2}]_{pk} [B_{2j}]_{kq} = \\ &= [A_{i1} B_{1j}]_{pq} + [A_{i2} B_{2j}]_{pq} = [A_{i1} B_{1j} + A_{i2} B_{2j}]_{pq}. \square \end{aligned}$$

Лемма 1.3.1 дает всю информацию об умножении  $2 \times 2$ -блочных матриц. Для общего случая имеем следующую теорему:

**Теорема 1.3.2.** Если

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{pq} \\ r_1 & & r_q \end{bmatrix} \begin{matrix} m_1 \\ \\ m_p \end{matrix}, \quad B = \begin{bmatrix} B_{11} & \cdots & B_{1t} \\ \vdots & & \vdots \\ B_{q1} & \cdots & B_{qt} \\ n_1 & & n_t \end{bmatrix} \begin{matrix} r_1 \\ \\ r_q \end{matrix}$$

и произведение  $C = AB$  разбито на блоки следующим образом:

$$C = \begin{bmatrix} C_{11} & \cdots & C_{1t} \\ \vdots & & \vdots \\ C_{p1} & \cdots & C_{pt} \\ n_1 & & n_t \end{bmatrix} \begin{matrix} m_1 \\ \\ m_p \end{matrix},$$

то

$$C_{ij} = \sum_{k=1}^q A_{ik} B_{kj}, \quad i = 1:p, j = 1:t.$$

*Доказательство.* Примените лемму 1.3.1 и индукцию. Мы опускаем детали, поскольку в них нет ничего поучительного.  $\square$

Так же, как и для обычных матриц, умножение блочных матриц может быть организовано несколькими способами. Для точного описания этих вычислений нам потребуется ввести некоторые обозначения.

### 1.3.3. Обозначения для подматриц

Рассмотрим возможные способы задания блоков в матрице, элементами которой являются числа. Пусть  $A \in \mathbb{R}^{m \times n}$  и векторы  $i = (i_1, \dots, i_r)$  и  $j = (j_1, \dots, j_c)$  имеют целочисленные компоненты, такие, что  $i_1, \dots, i_r \in \{1, 2, \dots, m\}$ ,  $j_1, \dots, j_c \in \{1, 2, \dots, n\}$ . Мы обозначаем через  $A(i, j)$   $r \times c$ -подматрицу

$$A(i, j) = \begin{bmatrix} A(i_1, j_1) & \cdots & A(i_1, j_c) \\ \vdots & & \vdots \\ A(i_r, j_1) & \cdots & A(i_r, j_c) \end{bmatrix}.$$

Компоненты векторов  $i$  и  $j$  часто пробегают непрерывный диапазон, и в этих случаях мы используем двоеточие для его обозначения. В частности, если  $1 \leq i \leq j \leq m$  и  $1 \leq p \leq q \leq n$ , то  $A(i:j, p:q)$  есть подматрица, стоящая на пересечении строк от  $i$ -й до  $j$ -й и столбцов от  $p$ -го до  $q$ -го матрицы  $A$ . Таким образом,

$$A(3:5, 1:2) = \begin{bmatrix} a_{31} & a_{32} \\ a_{41} & a_{42} \\ a_{51} & a_{52} \end{bmatrix}.$$

В заключение разговора о подматрицах напомним введенные в § 1.2.5 обозначения: если  $i$  и  $j$  – скаляры, то  $A(i, :)$  означает  $i$ -ю строку,  $A(:, j)$  –  $j$ -й столбец матрицы  $A$ .

### 1.3.4. Умножение блочной матрицы на вектор

Важный частный случай теоремы 1.3.2 – это умножение блочной матрицы на вектор. Предположим, что матрица  $A \in \mathbb{R}^{m \times n}$  и вектор  $z \in \mathbb{R}^m$  разбиты на блоки следующим образом:

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_p \end{bmatrix} \quad m_1 \quad z = \begin{bmatrix} z_1 \\ \vdots \\ z_p \end{bmatrix} \quad m_1.$$

Здесь  $m_1 + \dots + m_p = m$ . Мы называем  $A_i$   $i$ -й блочной строкой. Заметим, что если  $x \in \mathbb{R}^n$  и  $z = Ax$ , то  $z_i = A_i x$ ,  $i = 1:p$ . Таким образом, если  $m.vec = (m_1, \dots, m_p)$ , мы получаем следующую блочную версию умножения матрицы на вектор:

```
last = 0
for i = 1:p
    first = last + 1; last = first + m.vec(i) - 1
    z(first:last) = A(first:last,:)x
end
```

(1.3.1)

Конечно, произведения  $z_i = A_i x$  можно находить, используя как **matvec.ij**, так и **matvec.ji**.

Другая блочная версия умножения матрицы на вектор получится, если мы разобьем  $A$  и  $x$  на блоки следующим образом:

$$A = [A_1, \dots, A_q], \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_q \end{bmatrix}, \quad n_1 \quad n_q$$

В этом случае мы называем  $A_j$   $j$ -м блочным столбцом, и произведение  $z = Ax$  имеет вид  $z = A_1 x_1 + \dots + A_q x_q$ . Если  $n.vec = (n_1, \dots, n_q)$  – вектор размерностей блочных столбцов, то имеем следующую столбцовую версию алгоритма (1.3.1):

```

 $z(1:m) = 0; \text{last} = 0$ 
for  $j = 1:q$ 
     $first = last + 1; \text{last} = first + n.vec(j) - 1$ 
     $w = A(:, first:last) x(first:last)$ 
     $z = z + w$ 
end

```

(1.3.2)

Здесь снова отдельные произведения подматриц на векторы (вычисление  $w$ ) могут выполняться как **matvec.ij**, так и **matvec.ji**.

### 1.3.5. Перемножение блочных матриц

Так же как и обычное умножение матриц, умножение блочных матриц может быть организовано несколькими различными способами. Для простоты предположим, что каждая из матриц  $A = (A_{ij})$  и  $B = (B_{lj})$  есть  $N \times N$ -блочная матрица с  $\alpha \times \alpha$ -блоками, так что  $n = Na$ . Для такого «равномерного» разбиения на блоки теорема 1.3.2 дает, что если  $C = (C_{ij}) = AB$ , то

$$C_{ij} = \sum_{k_b=1}^N A_{ik_b} B_{k_b j}, \quad i = 1:N, j = 1:N.$$

Исходя из этой формулы, описывающей блочное скалярное произведение, можно получить следующую процедуру умножения матриц:

```

 $C(1:n, 1:n) = 0$ 
for  $i_b = 1:N$ 
     $i = (i_b - 1)\alpha + 1:i_b \alpha$ 
    for  $j_b = 1:N$ 
         $j = (j_b - 1)\alpha + 1:j_b \alpha$ 
        for  $k_b = 1:N$ 
             $k = (k_b - 1)\alpha + 1:k_b \alpha$ 
             $C(i, j) = C(i, j) + A(i, k) B(k, j)$ 
        end
    end
end

```

(1.3.3)

Благодаря нашим обозначениям, позволяющим переменным  $i, j, k$  играть роль векторных индексов, присваивание  $C(i, j) = C(i, j) + A(i, k) B(k, j)$  выглядит точно так же, как и на скалярном уровне.

Другие процедуры для умножения блочных матриц получатся, если изменить

порядок циклов в (1.3.3). Например:

```
C(1:n, 1:n) = 0
for jb = 1:N
    j = (jb - 1)a + 1:jba
    for kb = 1:N
        k = (kb - 1)a + 1:kba
        for ib = 1:N
            i = (ib - 1)a + 1:iba
            C(i, j) = C(i, j) + A(i, k)B(k, j)
        end
    end
end
```

Лучше всего можно понять эту организацию умножения матриц, если разбить  $A$  на блочные столбцы:

$$A = [ \begin{matrix} A_1 & \dots & A_N \end{matrix} ].$$

$\alpha \qquad \qquad \alpha$

Если  $B = (B_{ij})$ , где  $B_{ij} \in \mathbb{R}^{\alpha \times \alpha}$ , то

$$AB = C = [ \begin{matrix} C_1 & \dots & C_N \end{matrix} ], \quad C_j = \sum_{k=1}^N A_k B_{kj}.$$

Таким образом,  $C_j$  является *блочной линейной комбинацией* блочных столбцов матрицы  $A$ . Цикл по  $k_b$  выполняет *блочную операцию гахру*.

*Блочная версия с использованием внешнего произведения* получится, если мы переставим циклы в (1.3.3) следующим образом:

```
C(1:n, 1:n) = 0
for kb = 1:N
    k = (kb - 1)a + 1:kba
    for jb = 1:N
        j = (jb - 1)a + 1:jba
        for ib = 1:N
            i = (ib - 1)a + 1:iba
            C(i, j) = C(i, j) + A(i, k)B(k, j)
        end
    end
end
```

Для понимания этой версии лучше всего подойдут блочные разбиения

$$A = [ \begin{matrix} A_1 & \dots & A_N \end{matrix} ], \quad B = \begin{bmatrix} B_1 \\ \vdots \\ B_N \end{bmatrix}, \quad \alpha \qquad \alpha$$

так как тогда мы видим, что цикл по  $k_b$  вычисляет сумму

$$C = AB = \sum_{k_b=1}^N A_{k_b} B_{k_b}.$$

В этой формулировке мы рассматриваем матрицу  $C$  как сумму блочных внешних произведений.

### 1.3.6. Важный частный случай

В § 1.3.4 наш подход к вычислению произведения матрицы на вектор основывался на разбиении матрицы  $A$  на блочные строки и столбцы. Другой важный способ организации этого вычисления – рассмотреть  $A$  как  $2 \times 2$ -блочную матрицу, а  $x$  – как двумерный блочный вектор:

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{bmatrix}.$$

В последующий главах это блочное разбиение для задачи  $z = Ax$  будет интенсивно использоваться.

### 1.3.7. Структуры данных для блочных матриц

Массивы в фортране хранятся по столбцам. Это означает, что элементы матрицы, находящиеся в одном и том же столбце, занимают последовательные ячейки памяти. Так, если для хранения матрицы  $A \in \mathbb{R}^{4 \times 6}$  было отведено 24 ячейки памяти, то при традиционном хранении по столбцам элементы матрицы выстраиваются в памяти, как показано на рис. 1.3.1.

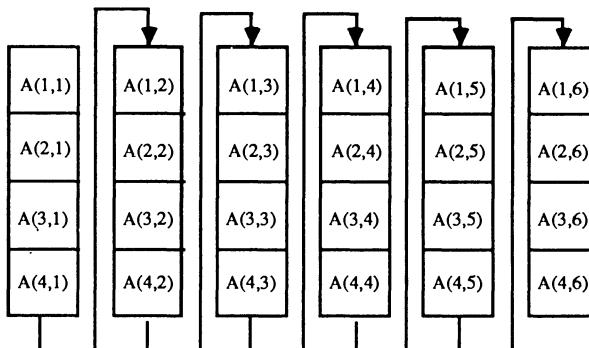


Рис. 1.3.1. Хранение по столбцам (случай  $4 \times 6$ ).

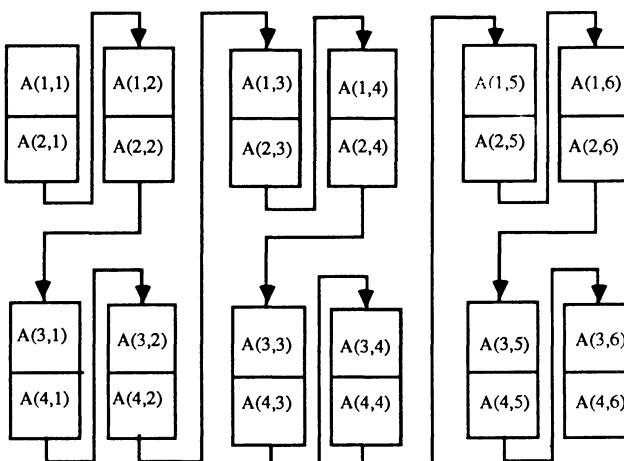


Рис. 1.3.2. Хранение блоками (случай  $4 \times 6$  с блоками  $2 \times 2$ ).

Другими словами, если  $A \in \mathbb{R}^m \times n$  хранится в  $v(1:m, 1:n)$ , то элемент  $A(i, j)$  находится в  $v((j - 1)m + i)$ . Такой формат хранения хорошо подходит для алгоритмов, которые выбирают элементы матриц по столбцам, поскольку элементы столбцов расположены в памяти непрерывно.

В некоторых блочных алгоритмах иногда бывает полезно хранить матрицы не по столбцам, а по блокам. Предположим, к примеру, что матрица  $A$  из рассмотренного выше примера есть  $2 \times 3$ -блочная матрица с  $2 \times 2$ -блоками. При блочном хранении по столбцам и при хранении по столбцам элементов каждого блока 24 элемента матрицы  $A$  располагаются в памяти, как показано на рис. 1.3.2.

Эта структура данных может быть привлекательной для блочных алгоритмов, поскольку элементы каждого блока хранятся в памяти непрерывно.

### 1.3.8. Умножение матриц, основанное на принципе «разделяй и властвуй»

В завершение этого раздела рассмотрим совершенно другой подход к задаче умножения матриц. Мы начнем обсуждение с умножения  $2 \times 2$ -блочных матриц

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

где каждый блок квадратный. В обычном алгоритме  $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$  имеются 8 умножений и 4 сложения блоков. Штрассен (1969) предложил способ вычисления  $C$  с использованием лишь 7 умножений и 18 сложений блоков:

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\ P_2 &= (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), \\ P_4 &= A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \\ C_{11} &= P_1 + P_4 - P_5 + P_7, \\ C_{12} &= P_3 + P_5, \\ C_{21} &= P_2 + P_4, \\ C_{22} &= P_1 + P_3 - P_2 + P_6. \end{aligned}$$

Эти соотношения легко проверяются подстановкой. Пусть  $n = 2m$ , так что блоки имеют размер  $m \times m$ . Подсчитывая сложения и умножения в вычислении  $C = AB$ , мы находим, что обычное умножение матриц требует  $(2m)^3$  умножений и  $(2m)^3 - (2m)^2$  сложений. Если же применить алгоритм Штрассена вместе с *обычным алгоритмом для умножения блоков*, то требуется всего  $7m^3$  умножений и  $7m^3 + 11m^2$  сложений. Если  $m \geq 1$ , то метод Штрассена обходится примерно в  $7/8$  арифметических затрат стандартного алгоритма.

Теперь заметим, что идея Штрассена может быть применена рекурсивно. В частности, алгоритм Штрассена может быть применен для нахождения каждого из произведений  $P_i$  блоков половинного размера. Таким образом, если исходные матрицы  $A$  и  $B$  имели размеры  $n \times n$  с  $n = 2^q$ , то алгоритм Штрассена можно применить многократно, и на самом нижнем уровне мы получим блоки размера  $1 \times 1$ . Но, конечно, нет никакой необходимости спускаться вниз до уровня  $n = 1$ . При достаточно малых размерах блока ( $n \leq n_{\min}$ ) может оказаться выгодным

вычислять  $P_i$ , используя стандартный алгоритм. Вся процедура в целом выглядит следующим образом:

**Алгоритм 1.3.1 (Умножение матриц по Штрассену).** Пусть  $n = 2^q$  и  $A \in \mathbb{R}^{n \times n}$ ,  $B \in \mathbb{R}^{n \times n}$ . Если  $n_{\min} = 2^d$ , где  $d \leq q$ , то следующий алгоритм вычисляет  $C = AB$ , применяя метод Штрассена рекурсивно  $q-d$  раз.

```

function:  $C = \text{strass}(A, B, n_{\min})$ 
   $n = \text{rows}(A)$ 
  if  $n \leq n_{\min}$ 
     $C = AB$ 
  else
     $m = n/2; u = 1:m; v = m + 1:n;$ 
     $P_1 = \text{strass}(A(u, u) + A(v, v), B(u, u) + B(v, v), n_{\min})$ 
     $P_2 = \text{strass}(A(v, u) + A(v, v), B(u, u), n_{\min})$ 
     $P_3 = \text{strass}(A(u, u), B(u, v) - B(v, v), n_{\min})$ 
     $P_4 = \text{strass}(A(v, v), B(v, u) - B(u, u), n_{\min})$ 
     $P_5 = \text{strass}(A(u, u) + A(u, v), B(v, v), n_{\min})$ 
     $P_6 = \text{strass}(A(v, u) - A(u, u), B(u, u) + B(u, v), n_{\min})$ 
     $P_7 = \text{strass}(A(u, v) - A(v, v), B(v, u) + B(v, v), n_{\min})$ 
     $C(u, u) = P_1 + P_4 - P_5 + P_7$ 
     $C(u, v) = P_3 + P_5$ 
     $C(v, u) = P_2 + P_4$ 
     $C(v, v) = P_1 + P_3 - P_2 + P_6$ 
  end
end strass
```

В отличие от всех ранее рассмотренных алгоритмов, процедура **strass** является рекурсивной, т. е. она вызывает сама себя. Алгоритмы, основанные на принципе «разделяй и властвуй», часто наиболее естественно описываются с использованием рекурсии.

Арифметические затраты процедуры **strass** являются нетривиально устроенной функцией  $n$  и  $n_{\min}$ . Если  $n_{\min} \geq 1$ , то достаточно подсчитать количество умножений, поскольку количество сложений примерно то же самое. Ограничившись подсчетом умножений, мы можем рассматривать только самый внутренний уровень рекурсии, поскольку именно там выполняются все умножения алгоритма. В процедуре **strass**  $q-d$  уровней рекурсии, поэтому стандартных умножений матрицы на матрицу всего  $7^{q-d}$ ; при этом перемножаются матрицы размера  $n_{\min}$ , так что всего в процедуре **strass** порядка  $s = (2^d)^3 7^{q-d}$  умножений по сравнению с  $c = (2^q)^3$  в обычном алгоритме. Заметим, что

$$\frac{s}{c} = \left(\frac{2^d}{2^q}\right)^3 7^{q-d} = \left(\frac{7}{8}\right)^{q-d}.$$

При  $d = 0$ , т. е. если мы спускаемся до  $1 \times 1$ -матриц, имеем

$$s = \left(\frac{7}{8}\right)^q c = 7^q = n^{\log_2 7} \approx n^{2.807}.$$

Таким образом, количество умножений в методе Штрассена асимптотически равно  $O(n^{2.807})$ . Однако относительная доля сложений по отношению к умножениям возрастает при уменьшении  $n_{\min}$ .

**Пример 1.3.1.** Если  $n = 1024$  и  $n_{\min} = 64$ , арифметические затраты процедуры **strass** составляют  $(7/8)^{10^{-6}} \approx 0.6$  от затрат стандартного алгоритма.

### Задачи

**1.3.1.** Пусть  $A \in \mathbb{R}^{m \times n}$  хранится по столбцам в массиве  $A.col$  ( $1:m:n$ ). Предположим, что  $n = \alpha N$ ,  $m = \beta M$  и мы рассматриваем  $A$  как  $M \times N$ -блочную матрицу с  $\beta \times \alpha$ -блоками. По данным целым числам  $i, j, r, s$ , удовлетворяющим неравенствам  $1 \leq i \leq \beta$ ,  $1 \leq j \leq \alpha$ ,  $1 \leq r \leq M$ ,  $1 \leq s \leq N$ , найдите такое  $k$ , что  $A.col(k)$  содержит элемент  $(i,j)$  из блока  $(r,s)$ . Постройте алгоритм, преобразующий  $A.col$  к блочному формату хранения матрицы  $A$ , показанному на рис. 1.3.2. Каких размеров при этом должен быть рабочий массив?

**1.3.2.** Обобщите алгоритм (1.3.4) на случай произвольной блочной матрицы.

**1.3.3.** Приспособьте процедуру **strass** для перемножения квадратных матриц произвольного порядка. Подсказка: если на каком-то шаге получилась матрица  $A$  нечетного порядка, окраймите ее нулевыми строкой и столбцом.

**1.3.4.** Пусть  $A = A_1 + iA_2 \in \mathbb{C}^m \times n$  и  $x = x_1 + ix_2 \in \mathbb{C}^n$ . Укажите способ вычисления  $y = y_1 + iy_2 = Ax$  при помощи одного вещественного вызова **matvec.ji**.

**1.3.5.** Пусть  $n$  – четное число. Определим функцию  $\mathbb{R}^n \rightarrow \mathbb{R}$ :

$$f(x) = x(1:2:n)^T x(2:n) = \sum_{i=1}^{n/2} x_{2i-1} x_{2i}.$$

(а) Покажите, что если  $x, y \in \mathbb{R}^n$ , то

$$x^T y = \sum_{i=1}^{n/2} (x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - f(x) - f(y).$$

(б) Приведите алгоритм, вычисляющий произведение  $n \times n$ -матриц  $C = AB$ , требующий  $n^3/2$  умножений при применении функции  $f$  к строкам матрицы  $A$  и столбцам матрицы  $B$ . Детали см. в Winograd (1968).

### Замечания и литература к § 1.3

Долгое время быстрые методы умножения матриц привлекали к себе достаточно много внимания. См.

Pan V. (1984). “How Can We Speed Up Matrix Multiplication?”, SIAM Review 26, 393–416.

Strassen V. (1969). “Gaussian Elimination is Not Optimal”, Numer. Math. 13, 354–356.

Winograd S. (1968). “A New Algorithm for Inner Product”, IEEE Trans. Comp. C-17, 693–694.

Практическая ценность многих из этих методов весьма сомнительна. Однако после публикации

Bailey D. (1988). “Extra High Speed Matrix Multiplication on the CRAY-2”, SIAM J. Sci. and Stat. Comp. 9, 603–607.

становится ясно, что полностью закрывать глаза на эти методы неблагородно. Bailey по существу реализовал Алгоритм 1.3.1 на компьютере CRAY-2 с  $n_{\min} = 128$ . Его программа требует около 60% времени, затрачиваемого библиотечной процедурой. Мы обсудим устойчивость алгоритма Штассена в § 2.7.7.

Наконец, отметим, что возникающие на практике матрицы часто обладают своей собственной блочной структурой. Так, блочные матрицы естественно возникают при дискретизации многих дифференциальных операторов. В марковском процессе кластеры состояний могут взаимодействовать между собой специальным образом, порождая блочно-структурированную задачу. См.

Kaufman L. (1983). “Matrix Methods for Queueing Problems”, SIAM J. Sci. and Stat. Comp. 4, 525–552.

## 1.4. Некоторые аспекты векторно-конвейерных вычислений

Действия с матрицами в основном слагаются из скалярных произведений и операций схр. Векторно-конвейерные компьютеры способны очень быстро выполнять подобные операции благодаря специальной аппаратной поддержке, учитывающей то обстоятельство, что векторная операция – это очень регулярно устроенная последовательность скалярных операций. Будет ли при использовании такого компьютера достигнута высокая производительность, это зависит от длины векторных операндов, а также ряда факторов, относящихся к пересылкам данных, таких как шаг выборки, количество векторных загрузок/записей, степень повторного использования вектора. Знакомство с этими вопросами и понимание их роли чрезвычайно полезны. Мы не будем пытаться построить всеобъемлющую модель векторно-конвейерных вычислений, позволяющую предсказывать производительность алгоритмов. Наша цель – дать читателям почувствовать основные моменты, которыми следует руководствоваться при создании эффективных программ для векторно-конвейерных компьютеров. Мы не ориентируемся на какую-то конкретную машину; заинтересованных читателей мы отсылаем к литературе, которая содержит очень много конкретных исследований такого рода.

### 1.4.1. Конвейеризация арифметических операций

Первопричиной высокой скорости векторных компьютеров является *конвейеризация*. Для осознания смысла этого понятия проведем аналогию со сборочной линией. Допустим, что сборка одного автомобиля идет по одной минуте на каждом из шестидесяти рабочих мест вдоль сборочного конвейера. Если конвейер полностью укомплектован персоналом, так что сборка нового автомобиля может начинаться каждую минуту, тогда 1000 автомобилей могут быть произведены примерно за  $1000 + 60 = 1060$  минут. Для заказа таких размеров эффективная «векторная скорость» конвейера будет 1000/1060 автомобилей в минуту. С другой стороны, при нехватке персонала, если сборка нового автомобиля может начинаться лишь один раз в час, для выпуска 1000 автомобилей потребуется 1000 часов. В этом случае эффективная «скалярная скорость» конвейера будет 1/60 автомобилей в минуту.

Именно так обстоят дела с конвейеризованной векторной операцией, скажем, со сложением векторов  $z = x + y$ . Скалярные операции  $z_i = x_i + y_i$  – это автомобили. Количество компонент вектора – это объем заказа. Если на вычисление каждого  $z_i$  уходит время  $t$ , то конвейеризованное сложение векторов длины  $n$  займет гораздо меньше времени, чем  $nt$ . Так достигается векторная скорость. Без конвейеризации векторное вычисление осуществлялось бы с той же скоростью, что и скалярное, и потребовало бы примерно  $nt$  времени.

Рассмотрим, как можно конвейеризовать последовательность операций с плавающей точкой. Обычно такие операции завершаются за несколько тактов. Например, трехтактовое сложение двух чисел  $x$  и  $y$  может протекать, как показано на рис. 1.4.1. Чтобы получить зрительный образ этой операции, представьте себе, что устройство сложения – это сборочная линия с тремя «рабочими местами», как изображено на рисунке. Входные числа  $x$  и  $y$  двигаются вдоль линии, задерживаясь



Рис. 1.4.1. Трехцикловый сумматор.

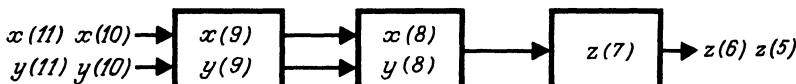


Рис. 1.4.2. Конвейерное сложение.

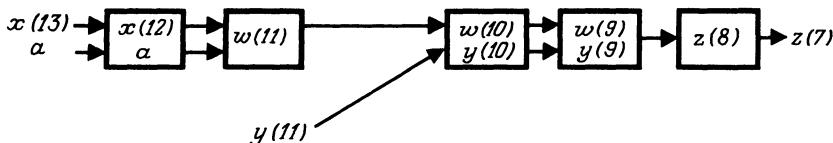


Рис. 1.4.3. Конвейерное saxpy.

на один такт у каждого рабочего места. После трех тактов на выходе появляется их сумма  $z$ . Заметим, что при выполнении одного отдельно взятого сложения в каждый момент времени активно только одно из трех рабочих мест.

Теперь рассмотрим сложение векторов  $z = x + y$ . Конвейеризация распределяет компоненты векторов  $x$  и  $y$  по всему устройству сложения. В установившемся состоянии, когда конвейер заполнен, очередная компонента  $z_i$  выдается на каждом такте. На рис. 1.4.2 изображено, как бы мог выглядеть конвейер в установившемся состоянии. В этом случае векторная скорость примерно в три раза выше скалярной скорости, поскольку для одного отдельного сложения требуется три такта.

Такие операции над векторами, как скалярное произведение и saxpy, включают как сложения, так и умножения. Мы будем предполагать, что в нашей модели векторного компьютера устройства сложения и умножения могут быть сцеплены друг с другом, образуя единый конвейер. Так, конвейеризация операции  $saxpy z = \alpha x + y$  могла бы выглядеть, как на рис. 1.4.3.

Здесь  $w = \alpha x$ . Таким образом, когда конвейер заполнен, на каждом такте мы получаем одну компоненту вектора  $z$ . Заметим, что в установившемся состоянии каждый такт выполняет одно сложение и одно умножение с плавающей точкой. В этом причина того, что пиковая производительность типичного векторного компьютера дается величиной  $2/\mu$  операций с плавающей точкой в секунду, где  $\mu$  – время одного такта.

## 1.4.2. Векторные операции

Векторно-конвейерный компьютер имеет набор *векторных инструкций*, таких как сложение векторов ( $z_i = x_i + y_i$ ), покомпонентное произведение векторов ( $z_i = x_i y_i$ ), умножение вектора на скаляр ( $z_i = \alpha x_i$ ), скалярное произведение и saxpy. Для ясности предположим, что эти операции осуществляются на *векторных регистрах*. Регистры обмениваются с памятью посредством инструкций загрузки вектора и записи вектора в память.

Важным атрибутом векторного процессора является длина его векторных регистров, которую мы обозначаем через  $v_L$ . Операция над векторами длины  $n$  должна быть разбита на несколько операций с операндами длины  $v_L$  или меньшей. К примеру, для сложения двух  $n$ -векторов  $x$  и  $y$  такое разбиение может быть осуществлено следующим образом:

```

first = 1
while first ≤ n
    last = min {n, first + vL - 1}
    Загрузить вектор x(first:last).
    Загрузить вектор y(first:last).
    Сложить векторы: z(first:last) = x(first:last) + y(first:last).
    Записать вектор: z(first:last).
    first = last + 1
end

```

(1.4.1)

Заметим, что здесь все операции (загрузить, сложить, записать) – это векторные операции. Порядочный компилятор для векторного компьютера должен автоматически генерировать (1.4.1) по оператору  $z = x + y$  программы. Конечно, возможно, что программисту придется задавать сложение векторов в виде фортрановского цикла.

### 1.4.3. Длина вектора

Предположим, что на начальную загрузку конвейера при выполнении векторной операции требуется  $\tau_{\text{op}}$  тактов. Если при заполненном конвейере на каждом такте получается одна компонента результата, то время выполнения  $n$ -мерной векторной операции будет

$$T_{\text{op}}(n) = (\tau_{\text{op}} + n)\mu, \quad n \leq v_L,$$

где  $\mu$  – продолжительность такта,  $v_L$  – длина векторных регистров компьютера.

Если длина векторов, участвующих в операции, превосходит длину векторных регистров, то, как мы видели, векторы должны быть разбиты на умещающиеся в векторные регистры куски. Так, если

$$n = n_1 v_L + n_0, \quad 0 \leq n_0 < v_L,$$

то общее время выполнения  $n$ -мерной векторной операции будет

$$T_{\text{op}}(n) = \begin{cases} n_1(\tau_{\text{op}} + v_L)\mu, & n_0 = 0, \\ (n_1(\tau_{\text{op}} + v_L) + \tau_{\text{op}} n_0)\mu, & n_0 \neq 0. \end{cases}$$

Это можно записать и более просто:

$$T_{\text{op}}(n) = (n + \tau_{\text{op}} \text{ceil}(n/v_L))\mu,$$

где  $\text{ceil}(\alpha)$  – наименьшее целое число, такое что  $\alpha \leq \text{ceil}(\alpha)$ . Если для вычисления каждой компоненты требуется  $\rho$  флопов, то эффективная скорость вычислений дается для произвольного  $n$  формулой:

$$R_{\text{op}}(n) = \frac{\rho n}{T_{\text{op}}(n)} = \frac{\rho}{\mu} \frac{1}{1 + \frac{\tau_{\text{op}}}{n} \text{ceil}\left(\frac{n}{v_L}\right)}.$$

(Если значение  $\mu$  берется в секундах, то  $R_{\text{op}}$  получается в флопах в секунду.) Асимптотическая скорость будет

$$\lim_{n \rightarrow \infty} R_{\text{op}}(n) = \frac{1}{1 + \frac{\tau_{\text{op}}}{v_L} \mu} \frac{\rho}{\mu}.$$

Для оценки относительной роли издержек на начальную загрузку конвейера Хокни и Джесскоуп (1988) ввели величину  $n_{1/2}$  как наименьшее  $n$ , для которого достигается половина пиковой производительности, т. е.

$$\frac{\rho n_{1/2}}{T_{\text{оп}}(n_{1/2})} = \frac{1}{2} \frac{\rho}{\mu}.$$

Для машин с большим коэффициентом  $n_{1/2}$  на операциях с короткими векторами не достигается высокая производительность.

#### 1.4.4. Оптимизация кода с учетом длины векторов

Мы представили упрощенную модель векторных вычислений, в которой производительность является функцией от длины векторов и коэффициента начальной загрузки  $\tau_{\text{оп}}$ . Забыв на время про все прочие факторы (как, скажем, метод доступа к данным), посмотрим, как эта модель советует организовать процедуру перемножения матриц.

Пусть нужно вычислить произведение  $C = AB$ , где  $A \in \mathbf{R}^{m \times r}$ ,  $B \in \mathbf{R}^{r \times n}$ . Вспомним из § 1.1, что стандартный алгоритм может быть организован шестью способами, причем самый внутренний цикл выполняет либо скалярное произведение, либо saxpy. Рассмотрим версию  $ijk$ :

```
for i = 1:m
    for j = 1:n
        for k = 1:r
            C(i, j) = C(i, j) + A(i, k) B(k, j)
        end
    end
end
```

Внутренний цикл вычисляет скалярное произведение векторов длины  $r$ . В условиях нашей простой модели, если мы не учитываем никаких затрат, кроме вычислений скалярных произведений, время выполнения алгоритма будет

$$T_{ijk} = mn[r + \text{ceil}(r/v_L)\tau_{\text{dot}}]\mu.$$

В версиях “ $jki$ ” и “ $kij$ ” самые внутренние циклы выполняют операцию saxpy длины  $m$  и  $n$  соответственно. Следовательно, ожидаемое время выполнения этих двух алгоритмов будет

$$T_{jki} = nr[m + \text{ceil}(m/v_L)\tau_{\text{sax}}]\mu$$

и

$$T_{kij} = mr[n + \text{ceil}(n/v_L)\tau_{\text{sax}}]\mu.$$

(Рассматривать версии “ $jik$ ”, “ $kji$ ” и “ $ikj$ ” нет необходимости, поскольку в нашей модели  $T_{jik} = T_{ijk}$ ,  $T_{kji} = T_{jki}$  и  $T_{ikj} = T_{kij}$ .) Из выписанных трех формул ясно, что наиболее эффективная версия матричного умножения определяется соотношением между размерностями  $m$ ,  $n$  и  $r$  и коэффициентами начальной загрузки  $\tau_{\text{sax}}$  и  $\tau_{\text{dot}}$ . Машинно-зависимая оптимизированная процедура умножения матриц могла бы следующим образом выбирать наиболее эффективную из возможных версий:

```
function: C = matmat.opt(A, B, v_L, tau_dot, tau_sax)
m = rows(A); n = cols(B); r = rows(B)
t_i = n * tau_sax * ceil(m / v_L)
t_j = m * tau_sax * ceil(n / v_L)
```

```

 $t_k = mn \tau_{dot} \text{ceil}(r/v_L)$ 
if  $t_i = \min(t_i, t_j, t_k)$ 
     $C = \text{matmat.jki}(A, B)$ 
elseif  $t_j = \min(t_i, t_j, t_k)$ 
     $C = \text{matmat.kij}(A, B)$ 
elseif  $t_k = \min(t_i, t_j, t_k)$ 
     $C = \text{matmat.ijk}(A, B)$ 
end
end matmat. opt

```

Элементарные прикидки показывают, что если  $\tau_{sax}$  и  $\tau_{dot}$  примерно равны, то в случае, когда  $m$ ,  $n$  и  $r$  меньше, чем  $v_L$ , функция **matmat.opt** выбирает версию с максимальной длиной внутреннего цикла. Если же  $m$ ,  $n$  и  $r$  существенно больше  $v_L$ , то разница между всеми тремя версиями невелика.

Напомним, что эти наблюдения, как и сама **matmat.opt**, основаны на одномерном представлении о векторно-конвейерных вычислениях, учитывающем лишь длину векторов. Для уточнения нашей модели мы должны обратиться к обмену данными между памятью и регистрами, на которых, собственно, и выполняются векторные операции.

#### 1.4.5. Многоуровневая память

Прежде чем векторный компьютер сможет выполнить векторную операцию, операнды должны находиться «в нужном месте», а за доставку в нужное место приходится платить. Проблема в том, что память обычно организована *иерархическим образом*. Между функциональными устройствами, выполняющими арифметические операции, и основной памятью, в которой лежат данные, может располагаться относительно небольших размеров высокоскоростная *кэш-память*. Детали могут меняться от машины к машине, но вот две вещи, о которых необходимо постоянно помнить при работе в условиях многоуровневой памяти:

- Объем памяти на каждом уровне иерархии ограничен по экономическим причинам и уменьшается при повышении уровня иерархии.
- Обмен данными между уровнями иерархии сопряжен с затратами, иногда сравнительно большими.

Выводы из этого очевидны. *Если мы хотим разработать эффективные матричные алгоритмы для типичного векторного компьютера, мы должны постоянно держать*

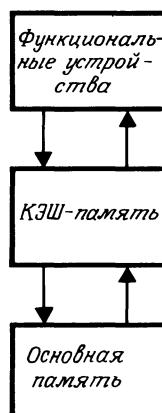


Рис. 1.4.4. Иерархия памяти.

в поле зрения передвижение данных по уровням иерархии памяти. Это остается в силе даже для вычислений с матрицами относительно малых размеров, поскольку современные компиляторы пока не способны отобразить основные матричные алгоритмы на системы с многоуровневой памятью.

Конечно, управление обменами с памятью не является чем-то особенно новым в научных вычислениях. При решении матричных задач на самых первых электронных машинах приходилось продумывать обмен данными с бумажными лентами и другими примитивными устройствами хранения информации. В вычислениях с разреженными матрицами создание эффективных программ всегда требовало внимания к вопросам обменов с памятью.

Многоуровневая память налагает на разработчика алгоритмов специфические требования, побуждая его мыслить в определенном ключе. Этому будет посвящен остаток данного раздела.

#### 1.4.6. Единичный шаг выборки

Очень важно, чтобы при перемещении данных между уровнями иерархии памяти эти данные занимали в памяти непрерывную область. Для векторов это подразумевает *единичный шаг выборки*. Шаг выборки для хранящегося в памяти вектора – это расстояние (измеренное в ячейках памяти) между его компонентами. Доступ к строке в двумерном фортрановском массиве не является операцией с единичным шагом выборки, поскольку массивы хранятся по столбцам.

Векторные операции с неединичным шагом выборки могут помешать конвейеризации. Например, загрузка вектора с неединичным шагом выборки из памяти с чередующимися адресами может потребовать дополнительного количества тактов из-за *конфликтов в банках памяти*.

Теперь учтем влияние шага выборки на производительность нашей оптимизированной процедуры `matmat.opt`. Из трех возможных самых глубоко вложенных циклов

```

for i = 1:m
    C(i, j) = C(i, j) + A(i, k) B(k, j)
end

for j = 1:n
    C(i, j) = C(i, j) + A(i, k) B(k, j)
end

for k = 1:r
    C(i, j) = C(i, j) + A(i, k) B(k, j)
end
```

только первый выбирает элементы массивов  $A$ ,  $B$  и  $C$  с единичным шагом. Ясно, что для реализации процедуры `matmat.opt` на фортране потребуется пересмотреть ее с учетом влияния шага выборки. Насколько сильно уточненная процедура будет тяготеть к избранию версии `matmat.jki`, зависит от того, насколько пагубно векторные операции с неединичным шагом выборкиказываются на производительности. Мы не будем дальше разбирать этот вопрос, так как он особенно сильно зависит от архитектуры конкретного компьютера. Для нас главное зафиксировать следующий момент: требования к длине вектора и к шагу выборки бывает трудно совместить.

Дilemмы такого рода типичны для высокопроизводительных вычислений. Достижение одной из целей (максимальная длина вектора) может противоречить

другой цели (единичный шаг выборки). Иногда наилучший компромисс можно найти, изучая в деталях времена исполнения отдельных операций.

#### 1.4.7. Роль структур данных

Иногда разумный выбор структур данных может помочь разрешить противоречивые требования на длину вектора и шаг выборки. Рассмотрим снова умножение матрицы на вектор  $z = Ax$ , где матрица  $A$  симметрична. Пусть  $A - n \times n$ -матрица, и пусть для простоты  $n \leq v_L$ . Если  $A$  хранится обычным образом, то при использовании процедуры `matvec.ji` основное вычисление состоит из  $n$  операций сахру длины  $n$  с единичным шагом выборки:

```
z(1:n) = 0
for j = 1:n
    z = z + A(:, j)x(j)
end
```

Считая только операции сахру, имеем следующее ожидаемое время исполнения:

$$T_1 = n(\tau_{\text{sax}} + n)\mu.$$

Посмотрим, насколько хуже станет производительность при использовании специальных структур данных, учитывающих симметрию. Если мы будем хранить нижний треугольник матрицы  $A$  по столбцам в массиве  $A.\text{vec}(:)$ , как в § 1.2.7, то соответствующая версия `matvec.ji` запишется в виде:

```
z(1:n) = 0
for j = 1:n
    for i = 1:j - 1
        z(i) = z(i) + A.\text{vec}((i - 1)n - i(i - 1)/2 + j)x(i)
    end
    for i = j:n
        z(i) = z(i) + A.\text{vec}((j - 1)n - j(j - 1)/2 + i)x(i)
    end
end
```

Хотя второй из циклов по  $i$  выполняет операцию сахру длины  $n - j + 1$  с единичным шагом выборки, первый цикл по  $i$  выполняет сахру длины  $j - 1$  уже с неединичным шагом.

Для того чтобы дать количественную оценку ущербу, понесенному вследствие использования операции с неединичным шагом выборки, нам придется сделать некоторые предположения. Допустим, что векторное вычисление с неединичным шагом выборки приводит к обработке со скалярной скоростью  $R_s$  флопов в секунду. Тогда на выполнение первого цикла (это примерно половина всех операций алгоритма) уходит  $n^2/R_s$  секунд. В то же время второй цикл, несмотря на убывание длины обрабатываемых им векторов, может в среднем работать с гораздо более высокой скоростью  $R_v$  флопов в секунду. Полное время вычислений дается формулой

$$T_2 = n^2 \left( \frac{1}{R_s} + \frac{1}{R_v} \right) = \frac{n^2}{R_s} \left( 1 + \frac{R_s}{R_v} \right).$$

Заметим, что даже в случае  $R_v = \infty$  достигается не более чем двукратное ускорение по сравнению со скалярной обработкой. Это одно из проявлений закона Амдала: Если в процедуре доля скалярной обработки составляет  $f \in [0, 1]$ , то

$$\frac{\text{Время вычислений при полностью скалярной обработке}}{\text{Время вычислений при векторной обработке с бесконечной скоростью}} = \frac{1}{f}.$$

Мы можем заключить, что в условиях, когда шаг выборки является критическим параметром, стандартная схема хранения матрицы  $A$  в процедуре `matvec.ji` предпочтительнее, чем хранение по столбцам нижнего треугольника.

Теперь предположим, что матрица  $A$  хранится по диагоналям в массиве  $A.diag(\cdot)$ , как описано в § 1.2.8, и что  $z = Ax$  вычисляется по алгоритму 1.2.4. В этом алгоритме все покомпонентные произведения векторов выполняются с единичным шагом выборки. Поэтому ожидаемое время вычислений зависит от издержек на начальную загрузку конвейера для покомпонентного произведения векторов:

$$T_3 = (\tau_{vm} + n)\mu + \sum_{k=1}^{n-1} [(\tau_{vm} + (n-k)) + (\tau_{vm} + (n-k))] \mu = \\ = ((2n-1)\tau_{vm} + n^2)\mu. \quad (1.4.2)$$

Таким образом, насколько выгодно применять схему хранения по диагоналям, определяется величиной  $T_3 - T_1 = ((2n-1)\tau_{vm} - n\tau_{sax})\mu$ , а также относительной значимостью компактности хранения информации. Но из нашего обсуждения ясно следует, что ценность компактных схем хранения снижается в условиях векторно-конвейерной обработки.

#### 1.4.8. Операции гахру, внешние произведения и векторные обмены

Во многих случаях большим шагом на пути к получению хорошей векторизованной программы является минимизация количества векторных загрузок и записей. Для удобства будем называть векторные загрузки и записи *векторными обменами*.

Многие алгоритмы численной линейной алгебры могут быть организованы таким образом, что доминирующей операцией будет являться либо модификация матрицы внешним произведением векторов, либо операция гахру. (См. § 1.1.15 и § 1.1.16.) Для матриц размера  $m \times n$  обе операции требуют  $2mn$  флопов, но гахру при этом требует в два раза меньше векторных обменов.

Чтобы убедиться в этом, возьмем для простоты случай  $m = m_1 v_L$  и  $n = n_1 v_L$ . Для векторных вычислений модификация внешним произведением запишется следующим образом:

```
for  $i_v = 1:m_1$ 
     $i = (i_v - 1)v_L + 1:i_v v_L$ ;  $w = u(i)$ 
    for  $j_v = 1:n_1$ 
         $j = (j_v - 1)v_L + 1:j_v v_L$ ;  $A(i, j) = A(i, j) + wv(j)^T$ 
    end
end
```

Каждый столбец подматрицы  $A(i, j)$  должен быть загружен, модифицирован и записан обратно в память. Учитывая также обмены для векторов  $i$  и  $v$ , приходим к следующей оценке для общего количества векторных обменов:

$$\sum_{i_v=1}^{m_1} \left( 1 + \sum_{j_v=1}^{n_1} (1 + 2v_L) \right) \approx 2m_1 n.$$

(Заметьте, что мы пренебрегли несущественными для нас слагаемыми низшего порядка.)

Теперь рассмотрим  $m \times n$ -операцию гахру  $u = u + Av$ , где  $u \in \mathbf{R}^m$ ,  $v \in \mathbf{R}^n$ ,  $A \in \mathbf{R}^{m \times n}$ .

Разбивая это вычисление на отрезки длины  $v_L$ , получаем:

```

for  $i = 1:m_1$ 
     $i = (i_v - 1)v_L + 1:i_v v_L; w = u(i)$ 
    for  $j_v = 1:n_1$ 
         $j = (j_v - 1)v_L + 1:j_v v_L$ 
         $w = w + A(i, j)v(j)$ 
    end
     $u(i) = w$ 
end

```

Здесь снова каждый столбец подматрицы  $A(i, j)$  должен быть считан, но запись в память теперь требуется только одна – необходимо записать вектор  $w$ . Таким образом, общее количество векторных обменов будет

$$\sum_{i_v=1}^{m_1} \left( 2 + \sum_{j_v=1}^{n_1} (1 + v_L) \right) \approx m_1 n,$$

т. е. примерно половина используемых при модификации внешним произведением.

*В практике векторных вычислений следует отдавать предпочтение тем схемам матричных алгоритмов, которые интенсивно используют операцию дахру, а не модификацию внешним произведением.*

#### 1.4.9. Блочные алгоритмы, кэш-память и повторное использование данных

Тщательное управление обменами с памятью является ключом к написанию хороших векторизованных программ. Мы проиллюстрировали это выше на примере векторных обменов. Сходные наблюдения могут быть сделаны относительно общего случая передвижения данных вверх и вниз по уровням иерархии памяти в ходе вычислений. В качестве конкретного примера рассмотрим, как может повлиять на разработку алгоритма перемножения матриц присутствие кэш-памяти. Напомним (см. § 1.4.5), что кэш-память можно представлять себе как локальную память, расположенную между основной памятью устройствами обработки. См. рис. 1.4.4. Для этой достаточно типичной ситуации мы будем считать, что операнды для векторной операции должны находиться в кэше и обмен данными между кэшем и основной памятью относительно дорогостоящ. Так что если уж какие-то данные попали из основной памяти в кэш, становится очень важным использовать их максимальным образом.

Рассмотрим умножение  $n \times n$ -матриц  $C = AB$ . Предположим, что кэш может вмещать  $M$  чисел с плавающей точкой, причем  $M \ll n^2$ . Разобъем матрицы  $B$  и  $C$  на блочные столбцы:

$$B = [B_1, \dots, B_N], \quad C = [C_1, \dots, C_N], \\ \alpha \qquad \alpha \qquad \alpha \qquad \alpha$$

где  $n = \alpha N$ . Предположим, что  $\alpha$  было выбрано таким образом, что  $M \approx n(2\alpha + 1) \approx 2n^2/N$ , так что в кэш как раз влезают блочные столбцы матриц  $B$  и  $C$ , а также столбец матрицы  $A$ . Вот процедура матричного умножения, которая добивается многократного использования каждого загружаемого в кэш блочного столбца  $B_j$ :

```

for  $j = 1:N$ 
    Загрузить в кэш  $B_j$  и  $C_j = 0$ .
    for  $k = 1:n$ 

```

```

Загрузить в кэш  $A(:, k)$  и обновить значение  $C_j$ 
end
Записать в основную память вычисленный  $C_j$ .
end

```

По сути дела, цикл по  $k$  выполняет ориентированное на использование схему умножение матриц  $C_j = AB_j$ . Подсчитывая количество  $\Gamma_1$  чисел с плавающей точкой, которые при выполнении нашего алгоритма проходят (и в том, и в другом направлении) по каналу «кэш – основная память», получаем:

$$\Gamma_1 = \sum_{j=1}^N [n(n/N) + n^2 + n(n/N)] = (2 + N)n^2 = 2n^2(1 + n^2/M).$$

Эта формула явственно демонстрирует преимущества большой кэш-памяти.

Теперь мы покажем, что подход, основанный на блочных скалярных произведениях, приводит к более эффективному использованию кэш-памяти. Будем рассматривать  $A(A_{ij})$ ,  $B = (B_{ij})$ ,  $C = (C_{ij})$  как  $N \times N$ -блочные матрицы с одним и тем же размером блоков  $a = n/N$ . Допустим, что  $N$  выбрано таким образом, что  $3a^2 \approx M$ , так что в кэш помещаются три блока. Располагая в кэше рабочий массив  $C_L$ , будем действовать следующим образом:

```

for i = 1:N
    for j = 1:N
         $C_L(1:a, 1:a) = 0$ 
        for k = 1:N
            Загрузить в кэш  $A_{ik}$  и  $B_{kj}$ .
             $C_L = C_L + A_{ik} B_{kj}$ 
        end
        Записать  $C_L$  в основную память на место, отведенное для  $C_{ij}$ .
    end
end

```

В этой версии количество обменов «кэш – основная память» будет

$$\Gamma_2 = \sum_{i=1}^N \sum_{j=1}^N \left( a^2 + \sum_{k=1}^N 2a^2 \right) = n^2(1 + 2N) \approx \frac{2n^3\sqrt{3}}{\sqrt{M}}.$$

Поскольку  $\Gamma_1/\Gamma_2 \approx n/\sqrt{3M}$ , мы видим, что блочные скалярные произведения имеют решительное преимущество. Например, для  $n = 2^{10}$  и  $M = 2^{14}$  наши два метода различаются по количеству обменов «кэш» – основная память» в 4.5 раза.

На этом примере мы видим, что наилучшая производительность для компьютеров, имеющих кэш-память, обычно достигается на блочных алгоритмах.

#### 1.4.10. Резюме

Мы продемонстрировали, что эффективность векторно-конвейерных вычислений с матрицами зависит от длины вектора, шага выборки, количества векторных обменов и способности алгоритма многократно использовать один и тот же блок данных. Проведение оптимизации по всем этим параметрам – дело чрезвычайно трудное и отчасти является искусством. В этом разделе мы обозначили основные моменты, на которые следует обращать внимание в векторно-конвейерных вычислениях. Конечно, хороший компилятор может выполнить за вас часть работы, но лучше не слишком-то на это полагаться!

### Задачи

**1.4.1.** Рассмотрим произведение матриц  $D = ABC$ , где  $A \in \mathbb{R}^{m \times r}$ ,  $B \in \mathbb{R}^{r \times n}$ ,  $C \in \mathbb{R}^{n \times q}$ . Пусть все матрицы хранятся по столбцам и время выполнения операции *saxpy* длины  $k$  с единичным шагом выборки дается формулой  $t(k) = (L + k)\mu$ , где  $L$ —константа,  $\mu$ —продолжительность такта. При каких условиях в этой модели выгоднее вычислять  $D$  как  $D = A(BC)$  вместо  $D = (AB)C$ ? Считайте, что все умножения матриц используют *saxpy*-алгоритм (это версия “*jkl*”).

**1.4.2.** Пусть все матрицы в процедуре *matmat.jki* хранятся по столбцам и время выполнения операции *saxpy* длины  $k$  с единичным шагом выборки дается формулой  $t(k) = (L + k)\mu$ , где  $L$ —константа,  $\mu$ —продолжительность такта. Сколько времени уходит в *matmat.jki* на выполнение операций *saxpy*? Адаптируйте *matmat.jki* так, чтобы она эффективно обрабатывала случай верхних треугольных  $n \times n$ -матриц  $A$  и  $B$ . Получается ли алгоритм для треугольных матриц в шесть раз быстрее, как предполагает подсчет количества флопов?

**1.4.3.** Постройте алгоритм вычисления  $C = A^T BA$ , где  $A$  и  $B$ — $n \times n$ -матрицы, причем  $B$  симметрична. Во всех наиболее глубоко вложенных циклах доступ к массивам должен осуществляться с единичным шагом выборки.

### Замечания и литература к § 1.4

Вот два превосходных обзора по векторным вычислениям:

Dongarra J. J., Gustavson F. G., and Karp A. (1984). “Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine”, SIAM Review 26, 91–112.

Ortega J. M. and Voigt R. G. (1985). “Solution of Partial Differential Equations on Vector and Parallel Computers”, SIAM Review 27, 149–240.

Весьма подробно рассмотрены матричные вычисления в системах с многоуровневой памятью в статье

Gallivan K., Jalby W., Meier U., and Sameh A. H. (1988). “Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design”, Int. J. Supercomputer Applic. 2, 12–48.

Разнообразные модели векторных вычислений предложены в книгах

Hockney R. W. and Jesshope C. R. (1988). Parallel Computers 2, Adam Hilger, Bristol and Philadelphia.

Schönauer W. (1987). Scientific Computing on Vector Computers, North Holland, Amsterdam.

Вот некоторые из многочисленных статей по практическим аспектам векторных вычислений:

Buzbee B. L. (1986). “A Strategy for Vectorization”, Parallel Computing 3, 187–192.

Dongarra J. and Eisenstat S. (1984). “Squeezing the Most Out of an Algorithm in Cray Fortran”, ACM Trans. Math. Soft. 10, 221–230.

Dongarra J. and Hinds A. (1979). “Unrolling Loops in Fortran”, Software Practice and Experience 9, 219–229.

Gallivan K., Jalby W., and Meier U. (1987). “The Use of BLAS3 in Linear Algebra on a Parallel Processor with a Hierarchical Memory”, SIAM J. Sci. and Stat. Comp. 8, 1079–1084.

Kågström B. and Ling P. (1988). “Level 2 and 3 Blas Routines for the IBM 3090 VF/400: Implementation and Experiences”, Report UMINF-154.88, Inst. of Inf. Proc., University of Umeå, S-901 87 Umeå, Sweden.

Madsen N., Roderigue G., and Karush J. (1976). “Matrix Multiplication by Diagonals on a Vector Parallel Processor”, Information Processing Letters 5, 41–45.

## Глава 2

# Матричный анализ

Вывод алгоритмов для матричных вычислений и их анализ требуют хорошего знакомства с некоторыми вопросами линейной алгебры. Обзор ряда основных понятий линейной алгебры дается в § 2.1. В § 2.2 и § 2.3 рассмотрены векторные и матричные нормы. В § 2.4 мы строим модель арифметики с конечной точностью и знакомим читателя с методами матричного анализа, потребными для количественной оценки влияния ошибок округления.

Следующие два раздела посвящены ортогональности, которая играет в матричных вычислениях заметную роль. Две разновидности ортогональных приведений – сингулярное разложение и CS-разложение – дадут нам необходимую теоретическую основу для введения важных понятий ранга и расстояния между подпространствами. Эти вопросы излагаются в § 2.5 и § 2.6.

Наконец, в последнем разделе мы изучаем поведение решения линейной системы  $Ax = b$  при возмущении матрицы  $A$  и вектора  $b$ . При этом вводится важное понятие числа обусловленности матрицы.

## 2.1. Основные сведения из линейной алгебры

Этот раздел представляет собой беглый обзор линейной алгебры. Читателям, которые пожелают ознакомиться с более подробным изложением предмета, мы рекомендуем обратиться к ссылкам в конце этого раздела.

### 2.1.1. Линейная независимость, подпространства, базис и размерность

Набор векторов  $\{a_1, \dots, a_n\}$  из  $\mathbf{R}^m$  линейно независим, если из  $\sum_{j=1}^n \alpha_j a_j = 0$  следует  $\alpha(1:n) = 0$ . В противном случае существует равная нулю нетривиальная линейная комбинация векторов  $a_i$ , и говорят, что набор  $\{a_1, \dots, a_n\}$  линейно зависим.

Подпространство  $\mathbf{R}^m$  – это такое его подмножество, которое также является линейным пространством. Если заданы векторы  $a_1, \dots, a_n \in \mathbf{R}^m$ , то множество всевозможных линейных комбинаций этих векторов является линейным пространством, называемым линейной оболочкой  $\{a_1, \dots, a_n\}$ :

$$\text{span}\{a_1, \dots, a_n\} = \left\{ \sum_{j=1}^n \beta_j a_j : \beta_j \in \mathbf{R} \right\}.$$

Если  $\{a_1, \dots, a_n\}$  – линейно независимый набор векторов и  $b \in \text{span}\{a_1, \dots, a_n\}$ , то  $b$  единственным образом представляется в виде линейной комбинации векторов  $a_j$ .

Если  $S_1, \dots, S_k$  – подпространства в  $\mathbf{R}^m$ , то их сумма определяется как подпространство  $S = \{a_1 + a_2 + \dots + a_k : a_i \in S_i, i = 1:k\}$ . Говорят, что  $S$  является *прямой суммой*, если каждый вектор  $v \in S$  имеет единственное представление  $v = a_1 + \dots + a_k$ , где  $a_i \in S_i$ . В этом случае мы пишем  $S = S_1 \oplus \dots \oplus S_k$ . Пересечение подпространств  $S_i$  также является линейным подпространством:  $S = S_1 \cap S_2 \cap \dots \cap S_k$ .

Подмножество  $\{a_{i_1}, \dots, a_{i_k}\}$  называется *максимальным линейно независимым подмножеством*  $\{a_1, \dots, a_n\}$ , если оно линейно независимо и не является собственным подмножеством никакого другого линейно независимого подмножества  $\{a_1, \dots, a_n\}$ . Если  $\{a_{i_1}, \dots, a_{i_k}\}$  – максимальное линейно независимое подмножество, то  $\text{span}\{a_1, \dots, a_n\} = \text{span}\{a_{i_1}, \dots, a_{i_k}\}$  и  $\{a_{i_1}, \dots, a_{i_k}\}$  является *базисом* для  $\text{span}\{a_1, \dots, a_n\}$ . Если  $S \in \mathbb{R}^m$  – подпространство, то можно найти базисные вектора  $a_1, \dots, a_k \in S$  так, что  $S = \text{span}\{a_1, \dots, a_k\}$ . Все базисы подпространства  $S$  имеют одинаковое количество элементов. Это число называется *размерностью*  $S$  и обозначается  $\dim(S)$ .

### 2.1.2. Область значений, ядро и ранг матрицы

Со всякой  $m \times m$ -матрицей  $A$  связаны два важных подпространства. *Область значений* матрицы  $A$  определяется так:

$$\text{range}(A) = \{y \in \mathbb{R}^m : y = Ax \text{ для некоторого } x \in \mathbb{R}^n\}.$$

*Ядро (нуль-пространство)* матрицы  $A$  задается следующим образом:

$$\text{null}(A) = \{x \in \mathbb{R}^n : Ax = 0\}.$$

Если  $A = [a_1, \dots, a_n]$  есть разбиение по столбцам, то

$$\text{range}(A) = \text{span}\{a_1, \dots, a_n\}.$$

*Ранг* матрицы  $A$  определяется следующим образом:

$$\text{rank}(A) = \dim(\text{range}(A)).$$

Можно показать, что  $\text{rank}(A) = \text{rank}(A^T)$ , и, таким образом, ранг матрицы равен максимальному числу линейно независимых строк или столбцов. Для любой матрицы  $A \in \mathbb{R}^{m \times n}$  мы имеем  $\dim(\text{null}(A)) + \text{rank}(A) = n$ .

### 2.1.3. Обратная матрица

*Единичная*  $n \times n$ -матрица  $I_n$  задается следующим блочно-столбцовыми представлением:

$$I_n = [e_1, \dots, e_n],$$

где  $e_k$  –  $k$ -й канонический вектор

$$e_k = \underbrace{(0, \dots, 0)}_{k-1}, \underbrace{1, \underbrace{0, \dots, 0}_n}_k^T.$$

Канонические векторы часто появляются в матричном анализе, и в тех случаях, когда их размерность не ясна из контекста, мы используем верхние индексы, например  $e_k^{(n)} \in \mathbb{R}^n$ .

Если матрицы  $A$  и  $X$  из  $\mathbb{R}^{n \times n}$  удовлетворяют равенству  $AX = I$ , то  $X$  является *обратной* к матрице  $A$  и для нее используется обозначение  $A^{-1}$ . Если  $A^{-1}$  существует, то матрица  $A$  называется *невырожденной*, в противном случае  $A$  называется *вырожденной*.

Некоторые свойства обратной матрицы играют важную роль в матричных вычислениях. Обратная к произведению матрица является произведением обратных к сомножителям, взятым в обратном порядке:

$$(AB)^{-1} = B^{-1}A^{-1}. \quad (2.1.1)$$

Транспонирование обратной матрицы – это то же самое, что обращение транспо-

нированной:

$$(A^{-1})^T = (A^T)^{-1} \equiv A^{-T}. \quad (2.1.2)$$

Тождество

$$B^{-1} = A^{-1} - B^{-1}(B - A)A^{-1} \quad (2.1.3)$$

показывает, как изменяется обратная матрица при изменении самой матрицы.

*Формула Шермана–Моррисона–Вудбери* дает удобное представление для матрицы, обратной к  $(A + UV^T)$ , где  $A \in \mathbb{R}^{n \times n}$ , а  $U$  и  $V \in n \times k$ -матрицы:

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}V^TA^{-1}. \quad (2.1.4)$$

Таким образом,  $k$ -ранговая модификация матрицы приводит к  $k$ -ранговой модификации ее обратной. В (2.1.4) мы предполагаем, что и  $A$ , и  $(I + V^TA^{-1}U)$  невырожденные.

Каждый из приведенных фактов можно легко проверить, просто показав, что выписанная матрица и в самом деле является обратной. Вот, например, как проверяется (2.1.3):

$$B(A^{-1} - B^{-1}(B - A)A^{-1}) = BA^{-1} - (B - A)A^{-1} = I.$$

#### 2.1.4. Детерминант

Если  $A = (a) \in \mathbb{R}^{1 \times 1}$ , то ее *детерминант* дается равенством  $\det(A) = a$ . Детерминант  $n \times n$ -матрицы определяется через детерминанты  $(n-1) \times (n-1)$ -матриц. Именно, для  $A \in \mathbb{R}^{n \times n}$  имеем:

$$\det(A) = \sum_{j=1}^n (-1)^{j+1} a_{1j} \det(A_{1j}),$$

где  $A_{1j}$  – это  $(n-1) \times (n-1)$ -матрица, получаемая из  $A$  вычеркиванием первой строки и  $j$ -го столбца. Вот несколько полезных свойств детерминанта:

$$\begin{array}{lll} \det(AB) & = \det(A)\det(B), & A, B \in \mathbb{R}^{n \times n}, \\ \det(A^T) & = \det(A), & A \in \mathbb{R}^{n \times n}, \\ \det(cA) & = c^n \det(A), & A \in \mathbb{R}^{n \times n}, c \in \mathbb{R}, \\ \det(A) \neq 0 & \Leftrightarrow A \text{ невырожденная}, & A \in \mathbb{R}^{n \times n}. \end{array}$$

#### 2.1.5. Дифференцирование

Пусть  $a$  – скалярная величина и  $A(a)$  –  $m \times n$ -матрица с элементами  $a_{ij}(a)$ . Если для всех  $i$  и  $j$  функция  $a_{ij}(a)$  дифференцируема по  $a$ , то под  $\dot{A}(a)$  мы подразумеваем матрицу

$$\dot{A}(a) = \frac{d}{da} A(a) = \left( \frac{d}{da} a_{ij}(a) \right) = (\dot{a}_{ij}(a)).$$

Дифференцирование матрицы, зависящей от параметра, оказывается удобным способом исследования чувствительности различных матричных задач.

#### Задачи

**2.1.1.** Покажите, что если матрица  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $p$ , то существуют такие матрицы  $X \in \mathbb{R}^{m \times p}$  и  $Y \in \mathbb{R}^{n \times p}$ , что

$$A = XY^T, \text{ где } \text{rank}(X) = \text{rank}(Y) = p.$$

**2.1.2.** Пусть  $A(a) \in \mathbb{R}^{m \times r}$  и  $B(a) \in \mathbb{R}^{r \times n}$  – матрицы, элементы которых являются дифференци-

руемыми функциями скаляра  $a$ . Покажите, что

$$\frac{d}{da} [A(a)B(a)] = \left[ \frac{d}{da} A(a) \right] B(a) + A(a) \left[ \frac{d}{da} B(a) \right].$$

**2.1.3.** Пусть элементы  $A(a) \in \mathbb{R}^{n \times n}$  – дифференцируемые функции скаляра  $a$ . В предположении что  $A(a)$  всегда невырожденная, покажите, что

$$\frac{d}{da} [A(a)^{-1}] = -A(a)^{-1} \left[ \frac{d}{da} A(a) \right] A(a)^{-1}.$$

**2.1.4.** Пусть  $A \in \mathbb{R}^{n \times n}$ ,  $b \in \mathbb{R}^n$  и отображение  $\psi: \mathbb{R}^n \rightarrow \mathbb{R}^n$  задается формулой  $\psi(x) = \frac{1}{2}x^T Ax - x^T b$ . Покажите, что градиент  $\psi$  выражается как  $\nabla\psi(x) = \frac{1}{2}(A^T + A)x - b$ .

### Замечания и литература к § 2.1

Среди множества вводных курсов линейной алгебры только некоторые обеспечивают начинаящего изучать матричные вычисления необходимым ему материалом. Особенно полезными мы считаем следующие книги:

Halmos P. R. (1958). Finite Dimensional Vector Spaces, 2nd ed., Van Nostrand-Reinhold, Princeton.  
[Имеется русский перевод: Халмос П. Р. Конечномерные векторные пространства.–М.: Физматгиз, 1960.]

Leon S. J. (1980). Linear Algebra with Applications. Macmillan, New York.

Noble B. and Daniel J. W. (1977). Applied Linear Algebra, Prentice-Hall, Englewood Cliffs.

Strang G. (1988). Linear Algebra and Its Applications, 3rd edition, Harcourt, Brace, Jovanovich, San Diego.

Более энциклопедическое изложение можно найти в

Bellman R. (1970). Introduction to Matrix Analysis, 2nd ed., McGraw-Hill, New York. [Имеется русский перевод первого издания: Беллман Р. Введение в теорию матриц.–М.: Наука, 1969.]

Gantmacher F. R. (1959). The Theory of Matrices, vols. 1 and 2, Chelsea, New York. [Имеется русский оригинал: Гантмахер Ф. Р. Теория матриц. 4 изд.–М.: Наука, 1989.]

Householder A. S. (1964). The Theory of Matrices in Numerical Analysis, Ginn (Blaisdell), Boston.

Стюарт (IMC, гл. 1) также дает великолепный обзор матричной алгебры.

## 2.2. Векторные нормы

Нормы в векторных пространствах служат той же цели, что и взятие модуля на вещественной прямой: они позволяют измерять расстояния. Точнее,  $\mathbb{R}^n$  вместе с введенной на  $\mathbb{R}^n$  нормой определяет метрическое пространство. Поэтому при работе с векторами и векторнозначными функциями мы будем располагать привычными понятиями окрестности, открытого множества, сходимости и непрерывности.

### 2.2.1. Определения

Норма вектора на  $\mathbb{R}^n$  – это функция  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ , удовлетворяющая следующим свойствам:

$$\begin{aligned} f(x) &\geq 0, & x \in \mathbb{R}^n & (f(x) = 0 \Leftrightarrow x = 0), \\ f(x+y) &\leq f(x) + f(y), & x, y \in \mathbb{R}^n, \\ f(ax) &= |a| f(x), & a \in \mathbb{R}, & x \in \mathbb{R}^n. \end{aligned}$$

Мы будем обозначать такие функции, окружая аргумент вертикальными двойными чертами:  $f(x) = \|x\|$ . Чтобы отличить одну норму от другой, мы будем использовать индексы при двойной черте.

Полезный класс векторных норм – это *p*-нормы, определяемые как

$$\|x\|_p = (\|x_1\|^p + \dots + \|x_n\|^p)^{1/p}, \quad p \geq 1. \quad (2.2.1)$$

Наиболее важными из *p*-норм являются 1, 2 и  $\infty$ -нормы:

$$\|x\|_1 = |x_1| + \dots + |x_n|,$$

$$\|x\|_2 = (\|x_1\|^2 + \dots + \|x_n\|^2)^{1/2} = (x^T x)^{1/2},$$

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Единичным по отношению к норме  $\|\cdot\|$  называется вектор  $x$ , удовлетворяющий равенству  $\|x\| = 1$ .

## 2.2.2. Некоторые свойства векторных норм

Классический результат о *p*-нормах – неравенство Гёльдера:

$$|x^T y| \leq \|x\|_p \|y\|_q, \quad \frac{1}{p} + \frac{1}{q} = 1. \quad (2.2.2)$$

Очень важным частным случаем этого неравенства является неравенство Коши–Шварца:

$$|x^T y| \leq \|x\|_2 \|y\|_2. \quad (2.2.3)$$

Все нормы в  $\mathbf{R}^n$  эквивалентны, т. е. для двух норм  $\|\cdot\|_\alpha$  и  $\|\cdot\|_\beta$  в  $\mathbf{R}^n$  существуют положительные константы  $c_1$  и  $c_2$ , такие, что

$$c_1 \|x\|_\alpha \leq \|x\|_\beta \leq c_2 \|x\|_\alpha \quad (2.2.4)$$

для всех  $x \in \mathbf{R}^n$ . Например, при  $x \in \mathbf{R}^n$

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n} \|x\|_2, \quad (2.2.5)$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty, \quad (2.2.6)$$

$$\|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty. \quad (2.2.7)$$

## 2.2.3. Абсолютная и относительная погрешности

Пусть  $\hat{x} \in \mathbf{R}^n$  есть аппроксимация к  $x \in \mathbf{R}^n$ . Для заданной векторной нормы  $\|\cdot\|$  мы будем говорить, что

$$\varepsilon_{abs} = \|\hat{x} - x\|$$

есть абсолютная погрешность  $\hat{x}$ , а при  $x \neq 0$  формула

$$\varepsilon_{rel} = \frac{\|\hat{x} - x\|}{\|x\|}$$

задает относительную погрешность  $\hat{x}$ . Для  $\infty$ -нормы относительная погрешность позволяет сформулировать утверждение о количестве верных значащих цифр в  $\hat{x}$ . В частности, если

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \approx 10^{-p},$$

то максимальная компонента вектора  $\hat{x}$  имеет примерно  $p$  верных значащих цифр.

**Пример 2.2.1.** Если  $x = (1.234 \ 0.05674)^T$ ,  $\hat{x} = (1.235 \ 0.05128)^T$ , то  $\|\hat{x} - x\|_\infty / \|x\|_\infty \approx 0.0043 \approx 10^{-3}$ . Обратите внимание, что в  $\hat{x}_1$  верны примерно три значащие цифры, в то время как в  $\hat{x}_2$  только одна.

## 2.2.4. Сходимость

Будем говорить, что последовательность  $\{x^{(k)}\}$   $n$ -векторов сходится к вектору  $x$ , если

$$\lim_{k \rightarrow \infty} \|x^{(k)} - x\| = 0.$$

Отметим, что вследствие (2.2.4) сходимость в  $\alpha$ -норме влечет сходимость в  $\beta$ -норме, и наоборот.

### Задачи

**2.2.1.** Покажите, что при  $x \in \mathbb{R}^n$   $\lim_{p \rightarrow \infty} \|x\|_p = \|x\|_\infty$ .

**2.2.2.** Докажите неравенство Коши–Шварца (2.2.3), рассмотрев неравенство  $0 \leq (ax + by)^T \times (ax + by)$  для подходящих чисел  $a$  и  $b$ .

**2.2.3.** Проверьте, что  $\|\cdot\|_1$ ,  $\|\cdot\|_2$  и  $\|\cdot\|_\infty$  являются векторными нормами.

**2.2.4.** Проверьте (2.2.5)–(2.2.7). Когда в каждом из этих неравенств достигается равенство?

**2.2.5.** Покажите, что в  $\mathbb{R}^n$   $x^{(i)} \rightarrow x$  тогда и только тогда, когда  $x_k^{(i)} \rightarrow x_k$  для  $k = 1:n$ .

**2.2.6.** Покажите, что любая векторная норма в  $\mathbb{R}^n$  равномерно непрерывна, проверив неравенство  $\|\|x\| - \|y\|\| \leq \|x - y\|$ .

**2.2.7.** Пусть  $\|\cdot\|$  – векторная норма в  $\mathbb{R}^m$  и  $A \in \mathbb{R}^{m \times n}$ . Покажите, что если  $\text{rank}(A) = n$ , то  $\|x\|_A = \|Ax\|$  – векторная норма в  $\mathbb{R}^n$ .

**2.2.8.** Пусть  $x$  и  $y$  – векторы из  $\mathbb{R}^n$ . Определим функцию  $\psi: \mathbb{R} \rightarrow \mathbb{R}$  равенством  $\psi(a) = \|x - ay\|$ . Покажите, что минимум этой функции достигается при  $a = x^T y / y^T y$ .

**2.2.9.** (а) Проверьте, что  $\|x\|_p = (|x_1|^p + \dots + |x_n|^p)^{1/p}$  есть векторная норма в  $\mathbb{C}^n$ . (б) Покажите, что при  $x \in \mathbb{C}^n$  имеем  $\|x\|_p \leq c (\|\operatorname{Re}(x)\|_p + \|\operatorname{Im}(x)\|_p)$ . (с) Найдите такую константу  $c_n$ , что  $c_n (\|\operatorname{Re}(x)\|_2 + \|\operatorname{Im}(x)\|_2) \leq \|x\|_2$  для всех  $x \in \mathbb{C}^n$ .

### Замечания и литература к § 2.2

Очень хорошее обсуждение векторных норм можно найти у Стьюарта (IMC). См. также Householder A. S. (1974). The Theory of Matrices in Numerical Analysis, Dover Publications, New York.

Pryce J. D. (1984). “A New Measure of Relative Error for Vectors”, SIAM J. Numer. Anal. 21, 202–221.

## 2.3. Матричные нормы

Матричные нормы часто требуются при анализе матричных алгоритмов. Например, программа решения систем линейных уравнений может давать некачественный результат, если матрица коэффициентов «почти вырожденная». Для количественной характеристики близости к вырожденности нам нужно уметь измерять расстояния в пространстве матриц. Такую возможность дают матричные нормы.

### 2.3.1. Определения

Поскольку  $\mathbb{R}^{m \times n}$  изоморфно  $\mathbb{R}^{mn}$ , определение матричной нормы должно быть эквивалентно определению векторной нормы. Именно,  $f: \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$  является матричной нормой, если

$$f(A) \geq 0, \quad A \in \mathbb{R}^{m \times n} \quad (f(A) = 0 \Leftrightarrow A = 0),$$

$$f(A + B) \leq f(A) + f(B), \quad A, B \in \mathbb{R}^{m \times n},$$

$$f(\alpha A) = |\alpha| f(A), \quad \alpha \in \mathbb{R}, \quad A \in \mathbb{R}^{m \times n}.$$

Мы будем обозначать матричные нормы так же, как и векторные нормы: двойной чертой, добавляя, если нужно, индексы, т. е.  $\|A\| = f(A)$ .

Чаще всего в вычислительной линейной алгебре используются норма Фробениуса

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} \quad (2.3.1)$$

и  $p$ -нормы

$$\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}. \quad (2.3.2)$$

Заметим, что матричные  $p$ -нормы определяются через векторные  $p$ -нормы, рассмотренные в предыдущем разделе. Проверку того, что (2.3.1) и (2.3.2) действительно нормы, мы оставляем читателю в качестве упражнения. Ясно, что  $\|A\|_p$  есть  $p$ -норма наибольшего вектора, полученного действием  $A$  на векторы единичной ( $p$ -норме) длины:

$$\|A\|_p = \sup_{x \neq 0} \left\| A \frac{x}{\|x\|_p} \right\|_p = \max_{\|x\|_p = 1} \|Ax\|_p.$$

Важно понимать, что (2.3.1) и (2.3.2) определяют целые семейства норм: 2-норма в  $\mathbb{R}^{3 \times 2}$  – это другая функция, чем 2-норма в  $\mathbb{R}^{5 \times 6}$ . Таким образом, легко проверяемое неравенство

$$\|AB\|_p \leq \|A\|_p \|B\|_p, \quad A \in \mathbb{R}^{m \times n}, \quad B \in \mathbb{R}^{n \times q} \quad (2.3.3)$$

в действительности описывает соотношение между тремя различными нормами. Формально мы будем говорить, что нормы  $f_1$ ,  $f_2$  и  $f_3$  в пространствах  $\mathbb{R}^{m \times q}$ ,  $\mathbb{R}^{m \times n}$ ,  $\mathbb{R}^{n \times q}$  взаимно согласованы, если для всех  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times q}$  мы имеем  $f_1(AB) \leq f_2(A)f_3(B)$ .

Не все матричные нормы удовлетворяют мультипликативному свойству

$$\|AB\| \leq \|A\| \|B\|. \quad (2.3.4)$$

Например, если  $\|A\|_\Delta = \max |a_{ij}|$  и

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

то  $\|AB\|_\Delta > \|A\|_\Delta \|B\|_\Delta$ . Но в основном мы будем работать с нормами, удовлетворяющими (2.3.4).

$p$ -Нормы обладают тем важным свойством, что для каждой матрицы  $A \in \mathbb{R}^{m \times n}$  и вектора  $x \in \mathbb{R}^n$  мы имеем  $\|Ax\|_p \leq \|A\|_p \|x\|_p$ . Более общо, для любых векторных норм  $\|\cdot\|_\alpha$  в  $\mathbb{R}^n$  и  $\|\cdot\|_\beta$  в  $\mathbb{R}^m$  мы имеем  $\|Ax\|_\beta \leq \|A\|_{\alpha, \beta} \|x\|_\alpha$ , где матричная норма  $\|A\|_{\alpha, \beta}$  определяется как

$$\|A\|_{\alpha, \beta} = \sup_{x \neq 0} \frac{\|Ax\|_\beta}{\|x\|_\alpha}. \quad (2.3.5)$$

Будем говорить, что  $\|\cdot\|_{\alpha, \beta}$  подчинена векторным нормам  $\|\cdot\|_\alpha$  и  $\|\cdot\|_\beta$ . Поскольку множество  $\{x \in \mathbb{R}^n : \|x\|_\alpha = 1\}$  компактно и замкнуто, в силу непрерывности нормы  $\|\cdot\|_\beta$  получаем, что

$$\|A\|_{\alpha, \beta} = \max_{\|x\|_\alpha = 1} \|Ax\|_\beta = \|Ax^*\|_\beta \quad (2.3.6)$$

для некоторого  $x^* \in \mathbb{R}^m$  с единичной  $\alpha$ -нормой.

### 2.3.2. Некоторые свойства матричных норм

Норма Фробениуса и  $p$ -нормы (в особенности при  $p = 1, 2, \infty$ ) удовлетворяют определенным неравенствам, часто используемым в анализе матричных вычислений. Для  $A \in \mathbb{R}^{m \times n}$  имеем:

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n} \|A\|_2, \quad (2.3.7)$$

$$\max_{i,j} |a_{ij}| \leq \|A\|_2 \leq \sqrt{mn} \max_{i,j} |a_{ij}|, \quad (2.3.8)$$

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^m |a_{ij}|, \quad (2.3.9)$$

$$\|A\|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^n |a_{ij}|, \quad (2.3.10)$$

$$\frac{1}{\sqrt{n}} \|A\|_\infty \leq \|A\|_2 \leq \sqrt{m} \|A\|_\infty, \quad (2.3.11)$$

$$\frac{1}{\sqrt{m}} \|A\|_1 \leq \|A\|_2 \leq \sqrt{n} \|A\|_1. \quad (2.3.12)$$

Доказательства нетрудны и оставляются в качестве упражнения.

Последовательность  $\{A^{(k)}\} \in \mathbb{R}^{m \times n}$  сходится, если  $\lim_{k \rightarrow \infty} \|A^{(k)} - A\| = 0$ . Выбор нормы не имеет значения, поскольку все нормы в  $\mathbb{R}^{m \times n}$  эквивалентны.

### 2.3.3. Матричная 2-норма

Прекрасным качеством матричных 1-нормы и  $\infty$ -нормы является то, что их легко вычислить, используя (2.3.9) и (2.3.10). Значительно сложнее оказывается характеристика 2-нормы.

**Теорема 2.3.1.** Если  $A \in \mathbb{R}^{m \times n}$ , то существует единичный в 2-норме  $n$ -вектор  $z$ , такой, что  $A^T A z = \mu^2 z$ , где  $\mu = \|A\|_2$ .

*Доказательство.* Пусть  $z \in \mathbb{R}^n$  – такой единичный вектор, что  $\|Az\|_2 = \|A\|_2$ . Поскольку  $z$  доставляет максимум функции

$$g(x) = \frac{1}{2} \frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2} \frac{x^T A^T Ax}{x^T x},$$

то должно выполняться  $\nabla g(x) = 0$ , где  $\nabla g$  – градиент  $g$ . Утомительное дифференцирование показывает, что для  $i = 1:n$

$$\frac{\partial g(z)}{\partial z_i} = \left[ (z^T z) \sum_{j=1}^n (A^T A)_{ij} z_j - (z^T A^T A z) z \right] / (z^T z)^2.$$

В векторных обозначениях это означает, что  $A^T A z = (z^T A^T A z) z$ . Утверждение теоремы следует отсюда, если положить  $\mu = \|Az\|_2$ .  $\square$

Из теоремы вытекает, что  $\|A\|_2^2$  является нулем многочлена  $p(\lambda) = \det(A^T A - \lambda I)$ . Более конкретно, 2-норма матрицы  $A$  равна квадратному корню из наибольшего собственного значения матрицы  $A^T A$ . О собственных значениях мы будем говорить подробно в гл. 7 и 8, а пока мы ограничиваемся замечанием, что вычисление 2-нормы носит итеративный характер и определенно сложнее, чем вычисление 1-нормы или  $\infty$ -нормы. К счастью, если целью является лишь получение оценки

$\|A\|_2$  по порядку величины, можно использовать (2.3.7), (2.3.11) или (2.3.12).

В качестве еще одного примера «анализа норм» приведем удобный для оценки 2-норм резульятат.

**Следствие 2.3.2.** Для  $A \in \mathbf{R}^{m \times n}$  имеем  $\|A\|_2 \leq \sqrt{\|A\|_1 \|A\|_\infty}$ .

**Доказательство.** Если вектор  $z \neq 0$  таков, что  $A^T A z = \mu^2 z$ , где  $\mu = \|A\|_2$ , то  $\mu^2 \|z\|_1 = \|A^T A z\|_1 \leq \|A^T\|_1 \|A\|_1 \|z\|_1 = \|A\|_\infty \|A\|_1 \|z\|_1$ .  $\square$

### 2.3.4. Возмущения и обратная матрица

Нормы часто используются для количественной оценки влияния возмущений, а также для доказательства того, что последовательность матриц сходится к указанному пределу. Для иллюстрации этих применений норм дадим количественную оценку возмущения  $A^{-1}$  как функцию возмущения  $A$ .

**Лемма 2.3.3.** Пусть  $F \in \mathbf{R}^{n \times n}$  и  $\|\cdot\|$  — норма, обладающая мультипликативным свойством (2.3.4). Если  $\|F\| < 1$ , то матрица  $I - F$  невырождена и

$$(I - F)^{-1} = \sum_{k=0}^{\infty} F^k,$$

причем

$$\|(I - F)^{-1}\| \leq \frac{1}{1 - \|F\|}.$$

**Доказательство.** Допустим, что  $I - F$  вырождена. Тогда для некоторого ненулевого вектора  $x$  имеем  $(I - F)x = 0$ . Но тогда из  $\|x\| = \|Fx\|$  вытекает, что  $\|F\| \geq 1$ , в противоречие с условием леммы. Значит,  $I - F$  невырождена. Чтобы найти выражение для обратной к ней, рассмотрим тождество

$$\left( \sum_{k=0}^N F^k \right) (I - F) = I - F^{N+1}.$$

Поскольку  $\|F\| < 1$ , имеем  $\lim_{k \rightarrow \infty} F^k = 0$ , так как  $\|F^k\| \leq \|F\|^k$ . Таким образом,

$$\left( \lim_{N \rightarrow \infty} \sum_{k=0}^N F^k \right) (I - F) = I.$$

Отсюда следует, что  $(I - F)^{-1} = \lim_{N \rightarrow \infty} \sum_{k=0}^N F^k$ . Теперь легко показать, что

$$\|(I - F)^{-1}\| \leq \sum_{k=0}^{\infty} \|F\|^k = \frac{1}{1 - \|F\|}. \quad \square$$

Заметим, что из леммы следует также  $\|(I - f)^{-1} - I\| \leq \|F\|/(1 - \|F\|)$ . Таким образом, для  $\varepsilon \ll 1$  возмущения порядка  $O(\varepsilon)$  в матрице  $I$  вызывают возмущения порядка  $O(\varepsilon)$  в обратной матрице. Распространим этот результат на матрицы общего вида.

**Теорема 2.3.4.** Если  $A$  — невырожденная матрица и  $\|A^{-1}E\| = r < 1$ , то матрица  $(A + E)$  невырождена и  $\|(A + E)^{-1} - A^{-1}\| \leq \|E\| \|A^{-1}\|^2 / (1 - r)$ .

**Доказательство.** Так как  $A$  невырождена,  $A + E = A(I - F)$ , где  $F = -A^{-1}E$ . Поскольку  $\|F\| = r < 1$ , из леммы 2.3.3 следует, что  $I - F$  невырождена и  $\|(I - F)^{-1}\| < 1/(1 - r)$ . Но  $(A + E)^{-1} = (I - F)^{-1} A^{-1}$ , так что

$$\|(A + E)^{-1}\| \leq \frac{\|A^{-1}\|}{1 - r}.$$

Из (2.1.3) мы получаем, что  $(A + E)^{-1} - A^{-1} = -A^{-1}E(A + E)^{-1}$  и, переходя к нормам, находим:

$$\|(A + E)^{-1} - A^{-1}\| \leq \|A^{-1}\| \|E\| \|(A + E)^{-1}\| \leq \frac{\|A^{-1}\|^2 \|E\|}{1 - r}. \square$$

### Задачи

**2.3.1.** Докажите, что  $\|AB\|_p \leq \|A\|_p \|B\|_p$ , где  $1 \leq p \leq \infty$ .

**2.3.2.** Пусть  $B$  – любая подматрица  $A$ . Покажите, что  $\|B\|_p \leq \|A\|_p$ .

**2.3.3.** Покажите, что если  $D = \text{diag}(\mu_1, \dots, \mu_k) \in \mathbf{R}^{m \times n}$ , где  $k = \min\{m, n\}$ , то  $\|D\|_p = \max|\mu_i|$ .

**2.3.4.** Проверьте (2.3.7) и (2.3.8)

**2.3.5.** Проверьте (2.3.9) и (2.3.10).

**2.3.6.** Проверьте (2.3.11) и (2.3.12).

**2.3.7.** Покажите, что если  $0 \neq s \in \mathbf{R}^n$  и  $E \in \mathbf{R}^{n \times n}$ , то

$$\left\| E \left( I - \frac{ss^T}{s^T s} \right) \right\|_F^2 = \|E\|_F^2 - \frac{\|Es\|_2^2}{s^T s}.$$

**2.3.8.** Пусть  $u \in \mathbf{R}^m$  и  $v \in \mathbf{R}^n$ . Покажите, что если  $E = uv^T$ , то  $\|E\|_F = \|E\|_2 = \|u\|_2 \|v\|_2$  и  $\|E\|_\infty \leq \|u\|_\infty \|v\|_1$ .

**2.3.9.** Пусть  $A \in \mathbf{R}^{m \times n}$ ,  $y \in \mathbf{R}^m$ ,  $0 \neq s \in \mathbf{R}^n$ . Покажите, что матрица  $E = (y - As)s^T/s^T s$  имеет наименьшую 2-норму среди всех  $m \times n$ -матриц, удовлетворяющих  $(A + E)s = y$ .

### Замечания и литература к § 2.3

Полный обзор норм имеется у Стьюарта (IMC, pp. 160–184). Вот еще несколько ссылок:  
 Bauer F. L. and Fike C. T. (1960). “Norms and Exclusion Theorems”, Numer. Math. 2, 137–144.  
 Householder A. S. (1974). The Theory of Matrices in Numerical Analysis, Dover Publications, New York.

Mirsky L. (1960). “Symmetric Gauge Functions and Unitarily. Invariant Norms”, Quart. J. Math. 11, 50–59.

Ortega J. M. (1972). Numerical Analysis: A Second Course, Academic Press, New York.

## 2.4. Матричные вычисления с конечной точностью

Ошибки округления – это одна из тех вещей, которые делают матричные вычисления нетривиальной и интересной областью. В этом разделе мы построим модель арифметики с плавающей точкой и применим ее для получения оценок ошибок при вычислении с плавающей точкой скалярных произведений, операций захру, произведений матрицы на вектор и матрицы на матрицу.

### 2.4.1. Числа с плавающей точкой

При выполнении вычислений на компьютере на результат каждой арифметической операции в общем случае влияет *ошибка округления*. Возникновение этой ошибки обусловлено тем, что аппаратура способна представлять лишь конечное подмножество множества вещественных чисел. Мы будем обозначать это подмножество буквой  $F$  и называть его элементы *числами с плавающей точкой*. Следуя соглашениям, принятым в книге Форсайта, Малькольма и Молера (1977, pp. 10–29), будем характеризовать систему чисел с плавающей точкой для каждой конкретной

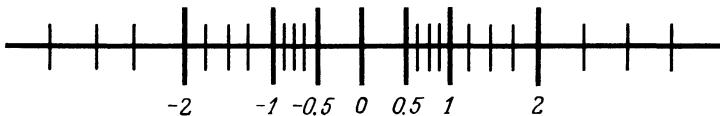


Рис. 2.4.1. Образец системы чисел с плавающей точкой.

машины четырьмя целочисленными параметрами: базой  $\beta$ , точностью  $t$  и интервалом значений показателя  $[L, U]$ . Конкретно,  $\mathbf{F}$  содержит все числа  $f$  вида

$$f = \pm d_1 d_2 \dots d_t \times \beta^e, \quad 0 \leq d_i < \beta, \quad d_1 \neq 0, \quad L \leq e \leq U,$$

а также число нуль. Обратим внимание, что для каждого ненулевого  $f \in \mathbf{F}$  выполняется  $m \leq |f| \leq M$ , где

$$m = \beta^{L-1}, \quad M = \beta^U(1 - \beta^{-t}). \quad (2.4.1)$$

Приведем пример. Для  $\beta = 2$ ,  $t = 3$ ,  $L = 0$ ,  $U = 2$  элементы множества  $\mathbf{F}$  показаны штрихами на рис. 2.4.1. Заметьте, что числа с плавающей точкой отстоят друг от друга на неравные промежутки по оси. Типичным значением для четверки  $(\beta, t, L, U)$  могло бы быть  $(2, 56, -64, 64)$ .

## 2.4.2. Модель арифметики с плавающей точкой

Чтобы делать общие утверждения о влиянии ошибок округления на данный алгоритм, необходимо иметь на множестве  $\mathbf{F}$  модель компьютерной арифметики. Для ее построения определим множество  $G$  как

$$G = \{x \in \mathbf{R} : m \leq |x| \leq M\} \cup \{0\} \quad (2.4.2)$$

и оператор  $fl: G \rightarrow \mathbf{F}$  как

$$fl(x) = \begin{cases} \text{ближайшее к } x \text{ число } c \in \mathbf{F}, \text{ если используется арифметика с округлением.} \\ \text{При наличии двух таких чисел выбирается то из них, которое отстоит} \\ \text{далее от нуля.} \\ \text{ближайшее к } x \text{ число } c \in \mathbf{F}, \text{ удовлетворяющее } |c| \leq |x|, \text{ если используется} \\ \text{арифметика с отбрасыванием разрядов.} \end{cases}$$

Пусть  $a$  и  $b$  – любые два числа с плавающей точкой и  $op$  означает любую из четырех арифметических операций  $+$ ,  $-$ ,  $\times$ ,  $\div$ . Если  $|a \text{ op } b| \notin G$ , то происходит *арифметическая ошибка*, являющаяся либо переполнением ( $|a \text{ op } b| > M$ ), либо обращением в машинный нуль ( $0 < |a \text{ op } b| < m$ ). Если  $|a \text{ op } b| \in G$ , то в нашей модели арифметики с плавающей точкой будем считать, что результатом выполнения  $(a \text{ op } b)$  на компьютере является  $fl(a \text{ op } b)$ .

Арифметическая ошибка обычно влечет за собой остановку программы, хотя некоторые машины и компиляторы продолжают счет при нехватке точности, полагая результат нулем. Обработка арифметических ошибок – это интересный и важный вопрос (см. Demmel (1984)).

Если арифметической ошибки не происходит, то важно ответить на вопрос о точности  $fl(a \text{ op } b)$ . Можно показать, что оператор  $fl$  удовлетворяет соотношению

$$fl(x) = x(1 + \varepsilon), \quad |\varepsilon| \leq u, \quad (2.4.3)$$

где  $u$  – единичная ошибка округления, определяемая как

$$u = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{для арифметики с округлением,} \\ \beta^{1-t} & \text{для арифметики с отбрасыванием разрядов.} \end{cases} \quad (2.4.4)$$

Следовательно,  $fl(a \text{ op } b) = (a \text{ op } b)(1 + \varepsilon)$ , где  $|\varepsilon| \leq u$ . Тогда оценка

$$\frac{|fl(a \text{ op } b) - (a \text{ op } b)|}{|a \text{ op } b|} \leq u, \quad a \text{ op } b \neq 0 \quad (2.4.5)$$

показывает, что отдельные арифметические операции выполняются с малой относительной погрешностью. Важно осознавать, однако, что это уже неизбежно будет так для последовательности операций.

**Пример 2.4.1.** При  $\beta = 10$ ,  $t = 3$  используется арифметика с плавающей точкой с отбрасыванием разрядов; можно показать, что  $fl[fl(10^{-3} + 1) - 1] = 0$ , что влечет за собой относительную ошибку, равную 1. С другой стороны, точный ответ дается выражением  $fl[fl(10^{-3} + 1) + fl(1 - 1)] = 10^{-3}$ . Отсюда видно, что арифметика с плавающей точкой не всегда ассоциативна.

### 2.4.3. Потеря точности

Еще один важный аспект арифметики с конечной точностью – феномен *катастрофической потери точности*. Грубо говоря, этот термин относится к потере многих верных значащих цифр при получении малых чисел в результате сложения/вычитания больших чисел. Хорошо известный пример, описанный в книге Форсайта, Малькольма и Молера (1977, pp. 14–16), – это вычисление  $e^{-a}$ ,  $a > 0$ , через ряд Тейлора. При таком методе ошибка округления примерно равна  $u$ , умноженному на наибольшую частичную сумму. Для больших значений  $a$  эта ошибка может даже превысить верное значение экспоненты, и ответ не будет содержать верных цифр, сколько бы членов ряда ни складывать. С другой стороны, если сложить достаточно членов в ряде Тейлора для  $e^a$  и обратить результат, можно получить оценку для  $e^{-a}$  с полной точностью.

### 2.4.4. Обозначение абсолютной величины

Прежде чем переходить к анализу ошибок округления в основных вычислениях с матрицами, мы введем некоторые полезные обозначения. Пусть  $A \in \mathbb{R}^{m \times n}$  и мы желаем дать количественную оценку ошибкам, возникающим при представлении этой матрицы числами с плавающей точкой. Обозначая через  $fl(A)$  реально хранимую в компьютере матрицу, мы видим, что

$$[fl(A)]_{ij} = fl(a_{ij}) = a_{ij}(1 + \varepsilon_{ij}), \quad |\varepsilon_{ij}| \leq u \quad (2.4.6)$$

для всех  $i$  и  $j$ . Можно дать более удобное представление для этого факта, если принять следующие два соглашения. Пусть  $A$  и  $B$  суть  $m \times n$ -матрицы. Тогда

$$\begin{aligned} B = |A| &\Rightarrow b_{ij} = |a_{ij}|, \quad i = 1:m, j = 1:n, \\ B \leq A &\Rightarrow b_{ij} \leq a_{ij}, \quad i = 1:m, j = 1:n. \end{aligned}$$

С учетом этих обозначений мы видим, что (2.4.6) принимает вид:

$$|fl(A) - A| \leq u |A|.$$

Подобное соотношение легко можно превратить в неравенство для норм, например,  $\|fl(A) - A\|_1 \leq u \|A\|_1$ . Однако для количественной оценки ошибок округления, возникающих при манипуляциях с матрицами, обозначение абсолютной величины может быть намного более информативным, поскольку сообщает нечто о каждом  $(i, j)$ -м элементе.

### 2.4.5. Ошибки округления в скалярных произведениях

Мы начинаем изучать матричные вычисления с конечной точностью с рассмотрения ошибок округления, возникающих в стандартном алгоритме вычисления скалярного произведения:

```
s = 0
for k = 1:n
    s = s + x_k y_k
end
```

(2.4.7)

Здесь  $x$  и  $y$  суть  $n \times 1$ -векторы, составленные из чисел с плавающей точкой.

Пытаясь дать количественную оценку возникающим в этом алгоритме ошибкам округления, мы немедленно сталкиваемся с проблемой обозначений: как различить вычисленные и точные величины? В тех случаях, когда ясно, какие именно вычисления производятся, мы будем использовать оператор  $fI(\cdot)$  для обозначения вычисленных величин. Так,  $fI(x^T y)$  означает реально вычисленный в (2.4.7) результат. Оценим  $|fI(x^T y) - (x^T y)|$ . Если

$$s_p = fI\left(\sum_{k=1}^p x_k y_k\right),$$

то  $s_1 = x_1 y_1 (1 + \delta_1)$ , где  $|\delta_1| \leq u$  и для  $p = 2:n$

$$\begin{aligned} s_p &= fI(s_{p-1} + fI(x_p y_p)) = \\ &= (s_{p-1} + x_p y_p (1 + \delta_p))(1 + \varepsilon_p), \quad |\delta_p|, |\varepsilon_p| \leq u. \end{aligned}$$
(2.4.8)

После несложных преобразований получаем, что

$$fI(x^T y) = s_n = \sum_{k=1}^n x_k y_k (1 + \gamma_k),$$

где

$$(1 + \gamma_k) = (1 + \delta_k) \prod_{j=k}^n (1 + \varepsilon_j)$$

при договоренности, что  $\varepsilon_1 = 0$ . Итак,

$$|fI(x^T y) - (x^T y)| \leq \sum_{k=1}^n |x_k y_k| |\gamma_k|. \quad (2.4.9)$$

Для дальнейшего необходимо оценить величины  $|\gamma_k|$  через  $u$ . На этом пути полезен следующий результат.

**Лемма 2.4.1.** Если  $(1 + \alpha) = \prod_{k=1}^n (1 + \alpha_k)$ , где  $|\alpha_k| \leq u$  и  $nu \leq 0.01$ , то  $|\alpha| \leq 1.01nu$ .

*Доказательство.* См. Форсайт и Молер (SLE, p. 92).  $\square$

Применяя этот результат к (2.4.9) при «разумном» предположении  $nu \leq 0.01$ , получаем

$$|fI(x^T y) - (x^T y)| \leq 1.01nu |x|^T |y|. \quad (2.4.10)$$

Обратите внимание, что если  $|x^T y| \ll |x|^T |y|$ , то относительная погрешность  $fI(x^T y)$  может уже не быть маленькой.

#### 2.4.6. Альтернативные методы количественной оценки ошибок

Более простой, хотя менее точный, способ оценки  $\alpha$  в лемме 2.4.1 дается неравенством  $|\alpha| \leq n\mathbf{u} + O(\mathbf{u}^2)$ . При этом мы имеем

$$|f(x^T y) - (x^T y)| \leq n\mathbf{u}|x|^T |y| + O(\mathbf{u}^2). \quad (2.4.11)$$

По-другому этот же результат можно выразить как

$$|f(x^T y) - (x^T y)| \leq \psi(n)\mathbf{u}|x|^T |y| \quad (2.4.12)$$

или как

$$|f(x^T y) - (x^T y)| \leq c n \mathbf{u} |x|^T |y|, \quad (2.4.13)$$

где в (2.4.12)  $\psi(n)$  – «не очень быстро растущая» функция  $n$ , а  $c$  в (2.4.13) – константа порядка 1.

Мы не будем отдавать предпочтение ни одному из показанных в (2.4.10) – (2.4.13) методов оценки ошибок. Это избавляет нас от необходимости приведения имеющихся в литературе результатов по ошибкам округления к фиксированному формату. Более того, излишнее внимание к деталям отдельных оценок не согласуется с «философией» анализа ошибок. Как пишет Уилкинсон (1971, р. 567):

«Все еще существует тенденция придавать слишком большое значение точным оценкам ошибок, полученным путем априорного анализа. По моему мнению, получаемая в этом процессе оценка обычно менее всего важна сама по себе. Основная цель подобного анализа – выявление в алгоритме потенциальных неустойчивостей (если они есть), с надеждой на то, что добывшие подобным образом знания могут привести к созданию улучшенных алгоритмов. Сама оценка обычно слабее, чем она могла бы быть, в силу необходимости придерживаться разумных пределов детализации, а также ввиду ограничений, налагаемых представлением ошибок в терминах матричных норм. Априорные оценки – это в общем случае не те величины, которые стоит использовать на практике. Для практического определения ошибок, как правило, следует использовать какую-либо разновидность апостериорного анализа, что дает все преимущества учета статистического распределения ошибок, а также других специальных свойств матрицы, как, например, разреженности».

Именно по этим причинам у анализа ошибок округления своя специфика, отличающая его от «чистого матанализа».

#### 2.4.7. Вычисление скалярных произведений с накоплением

В некоторых компьютерах предусматривается вычисление скалярных произведений с *накоплением в двойной точности*. Это означает, что если  $x$  и  $y$  – векторы, составленные из чисел с плавающей точкой с мантиссой длины  $t$ , то текущее значение суммы  $s$  из (2.4.7) накапливается в регистре с мантиссой длины  $2t$ . Поскольку произведение двух чисел с мантиссой длины  $t$  точно укладывается в переменную двойной точности, то ошибка округления возникает лишь в тот момент, когда  $s$  записывается в ячейку памяти с одинарной точностью. В этой ситуации обычно можно утверждать, что вычисленное скалярное произведение имеет хорошую оценку *относительной погрешности*, т. е.  $f(x^T y) = x^T y(1 + \delta)$ , где  $|\delta| \approx \mathbf{u}$ . Таким образом, способность вычислять скалярные произведения с накоплением очень привлекательна для вычислений с матрицами.

### 2.4.8. Ошибки округления в других основных матричных вычислениях

Легко показать, что если  $A$  и  $B$ —матрицы из чисел с плавающей точкой и  $\alpha$ —некоторое число с плавающей точкой, то

$$fl(\alpha A) = \alpha A + E, \quad |E| \leq u |\alpha A| \quad (2.4.14)$$

и

$$fl(A + B) = (A + B) + E, \quad |E| \leq u |A + B|. \quad (2.4.15)$$

Как следствие этих двух последних результатов легко получить, что вычисленные значения  $\alpha x$  и модификации внешним произведением удовлетворяют соотношениям

$$fl(\alpha x + y) = \alpha x + y + z, \quad |z| \leq u(2|\alpha x| + y) + O(u^2), \quad (2.4.16)$$

$$fl(C + uv^T) = C + uv^T + E, \quad |E| \leq u(|C| + 2|uv^T|) + O(u^2). \quad (2.4.17)$$

Используя (2.4.10), легко показать, что умножение двух матриц  $A$  и  $B$  с плавающей точкой, основанное на использовании скалярных произведений, удовлетворяет

$$fl(AB) = AB + E, \quad |E| \leq nu |A| |B| + O(u^2). \quad (2.4.18)$$

Этот же результат остается в силе при использовании процедуры  $\text{axpy}$  или модификации внешним произведением. Обратим внимание, что матричное умножение необязательно дает малую относительную погрешность, поскольку  $|AB|$  может быть гораздо меньше, чем  $|A| |B|$ . Например,

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -0.99 & 0 \end{bmatrix} = \begin{bmatrix} 0.01 & 0 \\ 0 & 0 \end{bmatrix}.$$

Из полученных результатов по ошибкам можно легко извлечь оценки в терминах норм. При рассмотрении 1-нормы в матричном умножении с плавающей точкой легко показать, используя (2.4.18), что

$$\|fl(AB) - AB\|_1 \leq nu \|A\|_1 \|B\|_1 + O(u^2). \quad (2.4.19)$$

### 2.4.9. Прямой и обратный анализ ошибок

Каждая из приведенных выше оценок ошибок округления получена путем *прямого анализа ошибок*. Альтернативный способ характеристики возникающих в алгоритме ошибок округления дает методика, называемая *обратным анализом ошибок*. В ней ошибки округления привязываются не к решению задачи, а к ее данным. Для иллюстрации рассмотрим умножение треугольных матриц при  $n = 2$ . Можно показать, что

$$fl(AB) = \begin{bmatrix} a_{11}b_{11}(1 + \varepsilon_1) & a_{11}b_{12}(1 + \varepsilon_2) + a_{12}b_{12}(1 + \varepsilon_3)(1 + \varepsilon_4) \\ 0 & a_{22}b_{22}(1 + \varepsilon_5) \end{bmatrix},$$

где  $|\varepsilon_i| \leq u$  для  $i = 1 : 5$ . Однако, если мы положим

$$\hat{A} = \begin{bmatrix} a_{11} & a_{12}(1 + \varepsilon_3)(1 + \varepsilon_4) \\ 0 & a_{22}(1 + \varepsilon_5) \end{bmatrix}$$

и

$$\hat{B} = \begin{bmatrix} b_{11}(1 + \varepsilon_1) & b_{12}(1 + \varepsilon_2)(1 + \varepsilon_4) \\ 0 & b_{22} \end{bmatrix},$$

то легко проверить, что  $fl(AB) = \hat{A}\hat{B}$ . Более того,

$$\begin{aligned}\hat{A} &= A + E, \quad |E| = 2\mathbf{u}|A| + O(\mathbf{u}^2), \\ \hat{B} &= B + F, \quad |F| \leq 2\mathbf{u}|B| + O(\mathbf{u}^2).\end{aligned}$$

Иными словами, вычисленное произведение есть точное произведение слегка возмущенных матриц  $A$  и  $B$ .

#### 2.4.10. Ошибки округления в алгоритме Штрассена

В § 1.3.8 мы привели нетрадиционную процедуру умножения матриц, изобретенную Штрассеном в 1969 г. Поучительно сравнить влияние ошибок округления для этого метода с любым из обычных алгоритмов матричного умножения из § 1.1.

Можно показать, что подход Штрассена (алгоритм 1.3.1) приводит к матрице  $\hat{C} = f/(AB)$ , удовлетворяющей неравенству вида (2.4.19). Для многих приложений этого вполне достаточно. Однако вычисляемая в методе Штрассена матрица  $\hat{C}$  не всегда удовлетворяет неравенству вида (2.4.18). Чтобы убедиться в этом, положим

$$A = B = \begin{bmatrix} 0.99 & 0.0010 \\ 0.0010 & 0.99 \end{bmatrix}$$

и пусть алгоритм 1.3.1 исполняется в арифметике с округлением до двух знаков. Среди вычисляемых величин, в частности, будут

$$\begin{aligned}\hat{P}_3 &= f/(0.99(0.001 - 0.99)) = -0.98, \\ \hat{P}_5 &= f/((0.001 + 0.99)0.99) = 0.98, \\ \hat{c}_{12} &= f/(\hat{P}_3 + \hat{P}_5) = 0.0.\end{aligned}$$

Но в точной арифметике  $c_{12} = 2(0.001)(0.99) = 0.00198$ , так что полученное в алгоритме 1.3.1  $\hat{c}_{12}$  не имеет верных значащих цифр. Алгоритм Штрассена в данном примере терпит неудачу по той причине, что малые внедиагональные элементы складываются с большими диагональными. В обычном же матричном умножении ни  $b_{11}$  с  $b_{12}$ , ни  $a_{11}$  с  $a_{12}$  не суммируются, так что потери вклада малых внедиагональных элементов не происходит. Действительно, для вышеприведенных матриц  $A$  и  $B$  стандартное умножение матриц дает  $\hat{c}_{12} = 0.0020$ .

В некоторых приложениях недостоверность отдельных компонент матрицы  $\hat{C}$  может быть серьезным недостатком. Например, в марковских процессах элементы  $a_{ij}$ ,  $b_{ij}$ ,  $c_{ij}$  суть вероятности переходов и поэтому неотрицательны. Если  $c_{ij}$  соответствует какой-нибудь особенно важной вероятности в моделируемом явлении, то его точное вычисление может быть критически важным. Заметим, что если  $A \geq 0$  и  $B \geq 0$ , то стандартное умножение матриц дает матрицу  $C$  с поэлементно малой относительной ошибкой:

$$|\hat{C} - C| \leq \mathbf{n}\mathbf{u}|A||B| + O(\mathbf{u}^2) = \mathbf{n}\mathbf{u}|C| + O(\mathbf{u}^2).$$

Это следует из (2.4.18). Так как аналогичное утверждение не будет справедливым для метода Штрассена, мы заключаем, что алгоритм 1.3.1 непривлекателен для некоторых задач перемножения неотрицательных матриц, если требуется получать относительно точные элементы  $\hat{c}_{ij}$ .

Развивая идеи проведенного здесь обсуждения, мы приходим к двум достаточно очевидным и в то же время важным выводам:

- Различные методы вычисления одной и той же величины могут приводить к существенно различным результатам.
- Насколько удовлетворительными окажутся выданные алгоритмом результаты, зависит от рода решаемой задачи и от целей человека, который эту задачу решает.

Эти утверждения будут разъяснены в последующих главах. Они оказываются тесно связанными с понятиями устойчивости алгоритма и обусловленности задачи.

### Задачи

**2.4.1.** Покажите, что если в (2.4.7)  $y = x$ , то  $f\ell(x^T x) = x^T x(1 + \alpha)$ , где  $|\alpha| \leq n\mathbf{u} + O(\mathbf{u}^2)$ .

**2.4.2.** Докажите (2.4.18).

**2.4.3.** Покажите, что если  $E \in \mathbb{R}^{m \times n}$ , где  $m \geq n$ , то  $\|E\|_2 \leq \sqrt{n} \|E\|_F$ . Этот результат полезен при выводе оценок в терминах норм из оценок в терминах абсолютных величин элементов.

**2.4.4.** Пусть имеется функция извлечения квадратного корня, удовлетворяющая условию  $f\ell(\sqrt{x}) = \sqrt{x}(1 + \varepsilon)$ , где  $|\varepsilon| \leq \mathbf{u}$ . Приведите алгоритм для вычисления  $\|x\|_2$  и оцените ошибки округления.

**2.4.5.** Пусть  $A$  и  $B$  – верхние треугольные  $n \times n$ -матрицы из чисел с плавающей точкой. Если  $\hat{C} = f\ell(AB)$  вычислено по любому из традиционных алгоритмов из § 1.1, верно ли, что  $\hat{C} = \hat{A}\hat{B}$ , где  $\hat{A}$  и  $\hat{B}$  близки к  $A$  и  $B$ ?

**2.4.6.** Пусть  $A$  и  $B$  –  $n \times n$ -матрицы из чисел с плавающей точкой и матрица  $A$  невырожденная, причем  $\|A^{-1}\|A\|_\infty = \tau$ . Покажите, что если  $\hat{C} = f\ell(AB)$  вычислено по любому из алгоритмов из § 1.1, то существует такая  $\hat{B}$ , что  $\hat{C} = A\hat{B}$  и  $\|\hat{B} - B\|_\infty \leq n\mathbf{u}\tau\|B\|_\infty + O(\mathbf{u}^2)$ .

### Замечания и литература к § 2.4

Наиболее полное изложение анализа ошибок округления содержится у Уилкинсона (AEP, гл. 3). Превосходное изложение имеется также у Форсайта и Молера (SLE, pp. 87–97) и Стьюарта (IMC, pp. 69–82). Для общего знакомства с ролью ошибок округления мы рекомендуем гл. 2 книги

Forsythe G. E., Malcolm M. A., and Moler C. B. (1977). Computer Methods for Mathematical Computations, Prentice-Hall, Englewood Cliffs, NJ. [Имеется русский перевод: Форсайт Дж., Мальcolm M., Молер К. Машинные методы математических вычислений.– М.: Мир, 1980.]  
и ее переработанного издания

Kahaner D., Moler C. B., and Nash S. (1988). Numerical Methods and Software, Prentice-Hall, Englewood Cliffs, NJ.

Мы уверенно рекомендуем классическое руководство

Wilkinson J. H. (1963). Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, NJ.

Философско-исторический обзор анализа ошибок округления содержится в фон-неймановской лекции Уилкинсона, опубликованной в

Wilkinson J. H. (1971). “Modern Error Analysis”, SIAM Review 13, 548–568.

В этой статье дано критическое рассмотрение ранних работ по анализу ошибок, принадлежащих фон Нейману и Голдстайну, Тьюрингу и Гивенсу.

Более современные течения в анализе ошибок включают интервальный анализ, построение статистических моделей ошибок округления и автоматизацию самого процесса анализа. Смотрите статью

Hull T. E. and Swensen J. R. (1966). “Tests of Probabilistic Models for Propagation of Roundoff Errors”, Comm. ACM 9, 108–113.

Larson J. and Sameh A. (1978). “Efficient Calculation of the Effects of Roundoff Errors”, ACM Trans. Math. Soft. 4, 228–236.

Miller W. and Spooner D. (1978). “Software for Roundoff Analysis, II”, ACM Trans. Mth. Soft. 4, 369–390.

Yohe J. M. (1979). “Software for Interval Arithmetic: A Reasonable Portable Package”, ACM Trans. Math. Soft. 5, 50–63.

Всякому, кто всерьез занимается разработкой программного обеспечения, необходимо глубокое понимание специфики вычислений с плавающей точкой. Неплохим отправным пунктом для приобретения знаний в этой области станет ознакомление со стандартом IEEE арифметики с плавающей точкой по работе

Stevenson D. (1981). "A Proposed Standard for Binary Floating Point Arithmetic", Computer 14 (March), 51–62.

Желательные свойства системы чисел с плавающей точкой по-прежнему интенсивно разрабатываются. Смотрите статьи

Demmel J. W. (1984). "Underflow and the Reliability of Numerical Software", SIAM J. Sci. and Stat. Comp. 5, 887–919.

Kulish U. W. and Miranker W. L. (1986). "The Arithmetic of the Digital Computer", SIAM Review 28, 1–40.

Даже для простейших задач разработка высококачественного программного обеспечения сопряжена с неимоверным количеством тонкостей и нюансов. Хороший пример – разработка подпрограммы для вычисления 2-норм:

Blue J. M. (1978). "A Portable FORTRAN Program to Find the Euclidian Norm of a Vector", ACM Trans. Math. Soft. 4, 15–23.

Анализ метода Штрассена и других «быстрых» алгоритмов линейной алгебры можно найти в

Brent R. P. (1970). "Error Analysis of Algorithms for Matrix Multiplication and Triangular Decomposition Using Winograd's Identity", Numer. Math. 16, 145–156.

Miller W. (1975). "Computational Complexity and Numerical Stability", SIAM J. Computing 4, 97–107.

## 2.5. Ортогональность и сингулярное разложение

Ортогональность играет в вычислениях с матрицами выдающуюся роль. Введя несколько определений, мы докажем очень полезную теорему о сингулярном разложении (SVD). Помимо всего прочего, SVD позволяет по-умному подойти к проблеме ранга матрицы. Понятие ранга, совершенно ясное в условиях точных вычислений, становится скользким в присутствии ошибок округления и погрешностей в исходных данных. Сингулярное разложение позволяет ввести практическое понятие численного ранга.

### 2.5.1. Ортогональность

Набор векторов  $\{x_1, \dots, x_p\}$  в  $\mathbf{R}^m$  называется *ортогональным*, если  $x_i^T x_j = 0$  при  $i \neq j$ , и *ортонормированным*, если  $x_i^T x_j = \delta_{ij}$ . Интуитивно, ортогональные векторы максимально независимы, поскольку направлены в совершенно разные стороны.

Подпространства  $S_1, \dots, S_p$  в  $\mathbf{R}^m$  попарно ортогональны, если  $x^T y = 0$  для всех  $x \in S_i$  и  $y \in S_j$ , таких что  $i \neq j$ . *Ортогональное дополнение* подпространства  $S \subseteq \mathbf{R}^m$  определяется как

$$S^\perp = \{y \in \mathbf{R}^m : y^T x = 0 \text{ для всех } x \in S\}.$$

Можно показать, что  $\text{range}(A)^\perp = \text{null}(A^T)$ .

Векторы  $v_1, \dots, v_k$  образуют *ортонормированный базис* подпространства  $S \subseteq \mathbf{R}^m$ , если они ортонормированы и их линейная оболочка совпадает с  $S$ . Такой базис всегда можно достроить до ортонормированного базиса  $\{v_1, \dots, v_m\}$  всего пространства  $\mathbf{R}^m$ . Отметим, что в этом случае  $S^\perp = \text{span}\{v_{k+1}, \dots, v_m\}$ .

Говорят, что матрица  $Q \in \mathbf{R}^{m \times m}$  *ортогональна*, если  $Q^T Q = I$ . Если  $Q = [q_1, \dots, q_m]$  ортогональна, то векторы  $q_i$  образуют ортонормированный базис в  $\mathbf{R}^m$ .

### 2.5.2. Нормы и ортогональные преобразования

2-норма инвариантна относительно ортогональных преобразований, поскольку если  $Q^T Q = I$ , то  $\|Qx\|_2^2 = x^T Q^T Qx = x^T x = \|x\|_2^2$ . Этим же свойством обладают

матричная 2-норма и норма Фробениуса. Именно можно показать, что для всех ортогональных матриц  $Q$  и  $Z$  подходящих размеров мы имеем:

$$\|QAZ\|_F = \|A\|_F \quad (2.5.1)$$

и

$$\|QAZ\|_2 = \|A\|_2. \quad (2.5.2)$$

### 2.5.3. Сингулярное разложение

Развитая в двух предыдущих разделах теория норм позволяет получить очень полезную теорему о сингулярном разложении.

**Теорема 2.5.1 (Сингулярное разложение (SVD)).** Если  $A$  – вещественная  $m \times n$ -матрица, то существуют ортогональные матрицы

$$U = [u_1, \dots, u_m] \in \mathbb{R}^{m \times m} \text{ и } V = [v_1, \dots, v_n] \in \mathbb{R}^{n \times n},$$

такие, что

$$U^T AV = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m \times n} \quad p = \min\{m, n\},$$

где  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$ .

*Доказательство.* Пусть  $x \in \mathbb{R}^n$  и  $y \in \mathbb{R}^m$  – единичные в 2-норме векторы, удовлетворяющие равенству  $Ax = \sigma y$ , где  $\sigma = \|A\|_2$ . Поскольку любой ортонормированный набор векторов может быть расширен до ортонормированного базиса всего пространства, то можно найти такие матрицы  $V_1 \in \mathbb{R}^{n \times (n-1)}$  и  $U_1 \in \mathbb{R}^{m \times (m-1)}$ , что  $V = [x \ V_1] \in \mathbb{R}^{n \times n}$  и  $U = [y \ U_1] \in \mathbb{R}^{m \times m}$  ортогональны. Нетрудно показать, что  $U^T AV$  имеет следующую структуру:

$$U^T AV = \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} \equiv A_1.$$

Так как

$$\left\| A_1 \begin{pmatrix} \sigma \\ w \end{pmatrix} \right\|_2^2 \geq (\sigma^2 + w^T w)^2,$$

то мы имеем  $\|A_1\|_2^2 \geq (\sigma^2 + w^T w)$ . Но  $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$  и поэтому необходимо  $w = 0$ . Теперь теорема легко доказывается применением метода математической индукции.  $\square$

Числа  $\sigma_i$  называются *сингулярными значениями* матрицы  $A$ , а векторы  $u_i$  и  $v_i$  – соответственно *левыми и правыми сингулярными векторами*. Сравнивая матричные равенства  $AV = \Sigma U$  и  $A^T U = \Sigma^T A$  по столбцам, легко убедиться, что

$$\left. \begin{aligned} Av_i &= \sigma_i u_i \\ A^T u_i &= \sigma_i v_i \end{aligned} \right\} i = 1 : \min\{m, n\}.$$

Введем следующие обозначения для сингулярных значений:

$\sigma_i(A)$  –  $i$ -е по величине сингулярное значение матрицы  $A$ ,

$\sigma_{\max}(A)$  – максимальное сингулярное значение матрицы  $A$ ,

$\sigma_{\min}(A)$  – минимальное сингулярное значение матрицы  $A$ .

Сингулярные значения матрицы  $A$  – это в точности полуоси гиперэллипсоида  $E$ , задаваемого соотношением  $E = \{Ax : \|x\|_2 = 1\}$ .

**Пример 2.5.1.**

$$A = \begin{bmatrix} 0.96 & 1.72 \\ 2.28 & 0.96 \end{bmatrix} = U \Sigma V^T = \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8 & 0.6 \\ 0.6 & -0.8 \end{bmatrix}^T.$$

Сингулярное разложение дает большую информацию о структуре матрицы. Если SVD матрицы  $A$  задано, как в теореме 2.5.1, то, определяя  $r$  соотношениями

$$\sigma_1 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0,$$

мы имеем

$$\text{rank}(A) = r, \quad (2.5.3)$$

$$\text{null}(A) = \text{span}\{v_{r+1}, \dots, v_n\}, \quad (2.5.4)$$

$$\text{range}(A) = \text{span}\{u_1, \dots, u_r\}. \quad (2.5.5)$$

Более того, обозначив  $U_r = U(:, 1:r)$ ,  $\Sigma_r = \Sigma(1:r, 1:r)$  и  $V_r = V(:, 1:r)$ , мы получим следующее разложение:

$$A = U_r \Sigma_r V_r^T = \sum_{i=1}^r \sigma_i u_i v_i^T. \quad (2.5.6)$$

Наконец, в терминах SVD изящно характеризуются как 2-норма, так и норма Фробениуса:

$$\|A\|_F^2 = \sigma_1^2 + \dots + \sigma_p^2, \quad p = \min\{m, n\}, \quad (2.5.7)$$

$$\|A\| = \sigma_1. \quad (2.5.8)$$

## 2.5.4. Неполнота ранга и SVD

Одно из наиболее ценных качеств сингулярного разложения состоит в том, что оно позволяет нам разумно обращаться с понятием ранга матрицы. Многие теоремы линейной алгебры формулируются примерно так: «Если такая-то матрица имеет полный ранг, то выполняются такие-то и такие-то свойства». Аккуратные и эстетичные результаты в этом духе тем не менее мало помогают в преодолении вычислительных затруднений, часто возникающих при работе с матрицами, близкими к матрицам неполного ранга. Ошибки округления и погрешности в исходных данных превращают определение ранга в нетривиальное упражнение. В действительности нас может интересовать  $\epsilon$ -ранг матрицы, который для некоторого малого  $\epsilon$  определяется по формуле

$$\text{rank}(A, \epsilon) = \min_{\|A - B\|_2 \leq \epsilon} \text{rank}(B).$$

Так, если элементы  $a_{ij}$  матрицы  $A$  получаются в лаборатории с точностью до  $\pm 0.001$ , то, по-видимому, имеет смысл посмотреть на  $\text{rank}(A, 0.001)$ . Аналогично, если  $A$  есть  $m \times n$ -матрица из чисел с плавающей точкой, то разумно считать  $A$  численно неполноранговой, если  $\text{rank}(A, \epsilon) < \min(m, n)$ , где  $\epsilon = \mathbf{u}\|A\|_2$ .

Численная неполнота ранга и  $\epsilon$ -ранг прекрасно характеризуются в терминах SVD, поскольку сингулярные значения показывают, насколько данная матрица близка к какой-нибудь матрице меньшего ранга.

**Теорема 2.5.2.** Пусть SVD матрицы  $A \in \mathbf{R}^{m \times n}$  задано, как в теореме 2.5.1. Если  $k < r = \text{rank}(A)$  и

$$A_k = \sum_{i=1}^k \sigma_i u_i v_i^T, \quad (2.5.9)$$

то

$$\min_{\text{rank}(B) = k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}. \quad (2.5.10)$$

*Доказательство.* Поскольку  $U^T A_k V = \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$ , то  $\text{rank}(A_k) = k$  и  $U^T(A - A_k)V = \text{diag}(0, \dots, 0, \sigma_{k+1}, \dots, \sigma_p)$ , так что  $\|A - A_k\|_2 = \sigma_{k+1}$ .

Теперь предположим, что  $\text{rank}(B) = k$  для некоторого  $B \in \mathbb{R}^{m \times n}$ . Тогда мы можем найти такие ортонормированные векторы  $x_1, \dots, x_{n-k}$ , что  $\text{null}(B) = \text{span} \times \{x_1, \dots, x_{n-k}\}$ . Из соображений размерности ясно, что

$$\text{span} \{x_1, \dots, x_{n-k}\} \cap \text{span} \{v_1, \dots, v_{k+1}\} \neq \{0\}.$$

Пусть  $z$  – единичный в 2-норме вектор, принадлежащий этому пересечению. Поскольку  $Bz = 0$  и

$$Az = \sum_{i=1}^{k+1} \sigma_i (v_i^T z) u_i,$$

мы имеем

$$\|A - B\|_2^2 \geq \| (A - B) z \|_2^2 = \| Az \|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z)^2 \geq \sigma_{k+1}^2,$$

что завершает доказательство нашей теоремы.  $\square$

Теорема 2.5.2 утверждает, что наименьшее сингулярное число матрицы  $A$  равняется измеренному в 2-норме расстоянию от этой матрицы до множества всех матриц неполного ранга. Из нее также следует, что множество матриц полного ранга является открытым и всюду плотным в  $\mathbb{R}^{m \times n}$ .

Наконец, если  $r_\epsilon = \text{rank}(A, \epsilon)$ , то

$$\sigma_1 \geq \dots \geq \sigma_{r_\epsilon} \geq \epsilon \geq \sigma_{r_\epsilon + 1} \geq \dots \geq \sigma_p, \quad p = \min \{m, n\}.$$

Мы вернемся к обсуждению вопросов, связанных с численным рангом матриц, в § 5.5 и § 12.2.

### Задачи

**2.5.1.** Покажите, что если  $S^T = -S$ , то матрица  $I - S$  невырождена, а матрица  $(I - S)^{-1} \times (I + S)$  ортогональна. Это *преобразование Кэли* матрицы  $S$ .

**2.5.2.** Покажите, что если треугольная матрица ортогональна, то она диагональна.

**2.5.3.** Покажите, что если матрица  $Q = Q_1 + iQ_2$ , где  $Q_1, Q_2 \in \mathbb{R}^{n \times n}$ , унитарна, то вещественная  $2n \times 2n$ -матрица

$$Z = \begin{bmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{bmatrix}$$

ортогональна.

**2.5.4.** Докажите свойства (2.5.3) – (2.5.8).

**2.5.5.** Докажите, что

$$\sigma_{\max}(A) = \max_{y \in \mathbb{R}^m, x \in \mathbb{R}^n} \frac{y^T Ax}{\|x\|_2 \|y\|_2}.$$

**2.5.6.** Для  $2 \times 2$ -матрицы  $A = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$  получите выражения для  $\sigma_{\max}(A)$  и  $\sigma_{\min}(A)$  как функции  $w, x, y$  и  $z$ .

**2.5.7.** Покажите, что любая матрица из  $\mathbb{R}^{m \times n}$  является пределом последовательности матриц полного ранга.

**2.5.8.** Покажите, что если матрица  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $n$ , то  $\|A(A^T A)^{-1} A^T\|_2 = 1$ .

**2.5.9.** Какая матрица ранга 1 ближе всего к матрице  $A = \begin{bmatrix} 1 & M \\ 0 & 1 \end{bmatrix}$  по норме Фробениуса?

**2.5.10.** Покажите, что для матрицы  $A \in \mathbb{R}^{m \times n}$   $\|A\|_F \leq \sqrt{\text{rank}(A)} \|A\|_2$ , что улучшает оценку (2.3.7).

### Замечания и литература к § 2.5

Хорошее описание роли SVD в анализе задачи  $Ax = b$  дают Форсайт и Молер (SLE). Их вывод самого разложения использует теорию собственных значений для симметричных матриц и в этом смысле более традиционен, чем наш. Среди ранних работ, относящихся к SVD, упомянем

Beltrami E. (1873). "Sulle Funzioni Bilineari", Giornale di Mathematiche 11, 98–106.

Eckart C. and Young G. (1939). "A Principal Axis Transformation for Non-Hermitian Matrices", Bull. Amer. Math. Soc. 45, 118–121.

Одно из наиболее существенных новшеств в научных вычислениях связано с расширением использования SVD в приложениях, где требуется тонкое обращение с матричным рангом. Диапазон приложений весьма впечатляющ; одно из наиболее интересных описано в

Moler C. B. and Morrison D. (1983). "Singular Value Analysis of Cryptograms", Amer. Math. Monthly 90, 78–87.

Об обобщениях SVD на случай бесконечномерных гильбертовых пространств см. книги

Gohberg I. C. and Krein M. G. (1969). Introduction to the Theory of Linear Non-Self Adjoint Operators, Amer. Math. Soc., Providence, R. I.

Smithies F. (1970). Integral Equations, Cambridge University Press, Cambridge.

Понижение ранга матрицы, как в теореме 2.5.2., при ограничениях на возмущающую матрицу обсуждается в

Demmel J. W. (1987). "The smallest perturbation of a submatrix which lowers the rank and constrained total least squares problems", SIAM J. Numer. Anal. 24, 199–206.

Golub G. H., Hoffman A., and Stewart G. W. (1988). "A Generalization of the Eckart–Young–Mirsky Approximation Theorem", Lin. Alg. and Its. Aplic. 88/89, 317–328.

Watson G. A. (1988). "The Smallest Perturbation of a Submatrix which Lowers the Rank of the Matrix", IMA J. Numer. Anal. 8, 295–304.

## 2.6. Проекции и CS-разложение

Если в результате вычисления должна получиться матрица или вектор, то для оценки точности ответа или для измерения того, насколько успешно продвигается итерационный процесс, удобно применять нормы. Если же необходимо вычислить целое подпространство, то для получения аналогичных оценок нам нужно уметь давать количественное выражение расстояния между подпространствами. Очень важную роль при этом играют ортогональные проекторы. Рассмотрев элементарные понятия, мы переходим к обсуждению CS-разложения. Это похожее на SVD разложение оказывается очень кстати, когда нужно сравнить пару подпространств. Мы начинаем с понятия ортогонального проектора.

### 2.6.1. Ортогональные проекторы

Пусть  $S \subseteq \mathbb{R}^n$  – подпространство. Матрица  $P \in \mathbb{R}^{n \times n}$  называется *ортогональным проектором* на подпространство  $S$ , если  $\text{range}(P) = S$ ,  $P^2 = P$  и  $P^T = P$ . Из этого определения легко усмотреть, что если  $x \in \mathbb{R}^n$ , то  $Px \in S$  и  $(I - P)x \in S^\perp$ .

Если  $P_1$  и  $P_2$  – ортогональные проекторы, то для любого  $z \in \mathbb{R}^n$  мы имеем

$$\|(P_1 - P_2)Z\|_2^2 = (P_1 z)^T (I - P_2)z + (P_2 z)^T (I - P_1)z.$$

Если  $\text{range}(P_1) = \text{range}(P_2) = S$ , то правая часть этого выражения обращается в нуль, показывая, что для данного подпространства ортогональный проектор на него единственен.

Если столбцы  $V = [v_1, \dots, v_k]$  образуют ортонормированный базис подпространства  $S$ , то  $P = VV^T$  – единственный ортогональный проектор на  $S$ . Заметим еще, что если  $v \in \mathbb{R}^n$ , то  $P = vv^T/v^Tv$  есть ортогональный проектор на  $S = \text{span}\{v\}$ .

## 2.6.2. Проекторы, связанные с SVD

С сингулярным разложением ассоциируется несколько важных ортогональных проекторов. Пусть  $A = U\Sigma V^T \in \mathbb{R}^{m \times n}$  есть сингулярное разложение матрицы  $A$ , причем  $\text{rank}(A) = r$ . Если матрицы  $U$  и  $V$  разбиты на блоки как

$$U = \begin{bmatrix} U_r & \tilde{U}_r \\ r & m-r \end{bmatrix}, \quad V = \begin{bmatrix} V_r & \tilde{V}_r \\ r & n-r \end{bmatrix},$$

то

$$\begin{aligned} V_r V_r^T &\text{ – проектор на } \text{null}(A)^\perp = \text{range}(A^T), \\ \tilde{V}_r \tilde{V}_r^T &\text{ – проектор на } \text{null}(A), \\ U_r U_r^T &\text{ – проектор на } \text{range}(A), \\ \tilde{U}_r \tilde{U}_r^T &\text{ – проектор на } \text{range}(A)^\perp = \text{null}(A^T). \end{aligned}$$

## 2.6.3. Расстояние между подпространствами

Взаимно однозначное соответствие между подпространствами и ортогональными проекторами на них позволяет нам выработать понятие расстояния между подпространствами. Предположим, что  $S_1$  и  $S_2$  – подпространства в  $\mathbb{R}^n$  и  $\dim(S_1) = \dim(S_2)$ . Определим *расстояние* между этими двумя подпространствами как

$$\text{dist}(S_1, S_2) = \|P_1 - P_2\|_2, \quad (2.6.1)$$

где  $P_i$  – ортогональные проекторы на  $S_i$ .

Геометрическую интерпретацию  $\text{dist}(\cdot, \cdot)$  можно получить, рассмотрев случай одномерных подпространств в  $\mathbb{R}^2$ . Пусть  $S_1 = \text{span}\{x\}$  и  $S_2 = \text{span}\{y\}$ , где  $x$  и  $y$  – единичные в 2-норме векторы из  $\mathbb{R}^2$ . Поскольку их 2-норма равна 1, мы можем записать

$$x = \begin{bmatrix} \cos(\theta_1) \\ \sin(\theta_1) \end{bmatrix}, \quad y = \begin{bmatrix} \cos(\theta_2) \\ \sin(\theta_2) \end{bmatrix}$$

для некоторых  $\theta_1$  и  $\theta_2$  из интервала  $[0, 2\pi]$ . Если ввести ортогональные матрицы

$$W = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) \end{bmatrix}, \quad Z = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) \end{bmatrix},$$

то

$$W^T y = \begin{bmatrix} \cos(\theta_2 - \theta_1) \\ \sin(\theta_2 - \theta_1) \end{bmatrix}, \quad Z^T x = \begin{bmatrix} \cos(\theta_1 - \theta_2) \\ \sin(\theta_1 - \theta_2) \end{bmatrix}.$$

Вычисление показывает, что

$$W^T(x x^T - y y^T) Z = \begin{bmatrix} 0 & \sin(\theta_1 - \theta_2) \\ \sin(\theta_1 - \theta_2) & 0 \end{bmatrix},$$

так что  $\|xx^T - yy^T\|_2 = \|W^T(xx^T - yy^T)Z\|_2 = |\sin(\theta_1 - \theta_2)|$ . Следовательно,  $\text{dist}(S_1, S_2) = |\sin(\theta_1 - \theta_2)|$ , величине синуса угла между двумя подпространствами.

Эта удачная геометрическая интерпретация функции расстояния применима и для более высоких размерностей, но предварительно необходимо расширить понятие угла между подпространствами. Следующая теорема позволяет это сделать.

**Теорема 2.6.1 (CS-разложение).** Если матрица

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{matrix} k \\ j \end{matrix}$$

ортогональна, причем  $k \geq j$ , то существуют ортогональные матрицы  $U_1, V_1 \in \mathbf{R}^{k \times k}$  и ортогональные матрицы  $U_2, V_2 \in \mathbf{R}^{j \times j}$ , такие что

$$\begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix} = \begin{Bmatrix} I_{k-j} & 0 & 0 \\ 0 & C & S \\ 0 & -S & C \end{Bmatrix}, \quad (2.6.2)$$

где

$$C = \text{diag}(c_1, \dots, c_j) \in \mathbf{R}^{j \times j}, \quad c_i = \cos(\theta_i), \\ S = \text{diag}(s_1, \dots, s_j) \in \mathbf{R}^{j \times j}, \quad s_i = \sin(\theta_i)$$

и  $0 \leq \theta_1 \leq \theta_2 \leq \dots \leq \theta_j \leq \pi/2$ .

*Доказательство.* См. Davis and Kahan (1970) или Stewart (1977). Предположение  $k \geq j$  сделано лишь для удобства обозначений.  $\square$

Грубо говоря, CS-разложение эквивалентно одновременной диагонализации блоков ортогональной матрицы.

**Пример 2.6.1.** Матрицы

$$Q = \begin{bmatrix} -0.716 & 0.548 & 0.433 \\ -0.698 & -0.555 & -0.451 \\ -0.006 & -0.626 & 0.780 \end{bmatrix}, \quad U = \begin{bmatrix} -0.721 & -0.692 & 0.000 \\ -0.692 & 0.721 & 0.000 \\ 0.000 & 0.000 & 1 \end{bmatrix}$$

и

$$V = \begin{bmatrix} 0.999 & -0.010 & 0.000 \\ -0.010 & -0.999 & 0.000 \\ 0.000 & 0.000 & 1 \end{bmatrix}$$

ортогональны с точностью до трех десятичных знаков и удовлетворяют соотношению

$$U^T Q V = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.000 & 0.780 & 0.625 \\ 0.000 & -0.625 & 0.780 \end{bmatrix}.$$

Таким образом, на языке теоремы 2.6.1  $k = 2, j = 1, c_1 = 0.780$  и  $s_1 = 0.625$ .

Теперь мы покажем, как подходящим образом выбранное CS-разложение раскрывает «степень близости» между парой подпространств одинаковой размерности.

**Следствие 2.6.2.** Пусть  $W = [W_1, W_2]$  и  $Z = [Z_1, Z_2]$  – ортогональные матрицы, причем  $W_1, Z_1 \in \mathbf{R}^{n \times k}$ , а  $W_2, Z_2 \in \mathbf{R}^{n \times (n-k)}$ . Если  $S_1 = \text{range}(W_1)$  и  $S_2 = \text{range}(Z_1)$ , то  $\text{dist}(S_1, S_2) = \sqrt{1 - \sigma_{\min}^2(W_1^T Z_1)}$ .

*Доказательство.* Пусть  $Q = W^T Z$ , и предположим, что  $k \geq j = n - k$ . Пусть CS-разложение  $Q$  дается формулой (2.6.2) с  $Q_{ip} = W_i^T Z_p$ . Отсюда следует, что

$$\|W_1^T Z_2\|_2 = \|W_2^T Z_1\|_2 = s_j = \sqrt{1 - c_j^2} = \sqrt{1 - \sigma_{\min}^2(W_1^T Z_1)}.$$

Поскольку  $W_1 W_1^T$  и  $Z_1 Z_1^T$  – ортогональные проекторы на  $S_1$  и  $S_2$  соответственно, мы имеем

$$\text{dist}(S_1, S_2) = \|W_1 W_1^T - Z_1 Z_1^T\|_2 =$$

$$= \|W^T (W_1 W_1^T - Z_1 Z_1^T) Z\|_2 = \left\| \begin{bmatrix} 0 & W_1^T Z_2 \\ -W_2^T Z_1 & 0 \end{bmatrix} \right\|_2 = s_j.$$

Если же  $k < j$ , то наше рассуждение все равно проходит, если просто положить

$$Q = [W_2, W_1]^T [Z_2, Z_1]$$

и заметить, что  $\sigma_{\max}(W_2^T Z_1) = \sigma_{\max}(W_1^T Z_2) = s_j$ .  $\square$

**Пример 2.6.2.** Если  $S_1 = \text{span}\{e_1, e_2\}$  и  $S_2 = \text{span}\{e_2, e_3\}$  – подпространства  $\mathbf{R}^n$ , то матрица  $Q$  из следствия равна

$$Q = [e_1, e_2, e_3, e_4, \dots, e_n]^T [e_2, e_3, e_1, e_4, \dots, e_n] = [e_2, e_3, e_1, e_4, \dots, e_n].$$

Таким образом,  $\text{dist}(S_1, S_2) = 1$ . Этот пример показывает, что факт пересечения подпространств не эквивалентен нулевому расстоянию между ними.

### Задачи

**2.6.1.** Покажите, что если  $P$  – ортогональный проектор, то матрица  $Q = I - 2P$  ортогональна.

**2.6.2.** Чему равны сингулярные числа ортогонального проектора?

**2.6.3.** Пусть в теореме 2.6.1  $k = j$ . (а) Покажите, что

$$\|Q_{11} v\|_2 = \delta_{\max}(Q_{11}) \|v\|_2 \Rightarrow \|Q_{21} v\|_2 = \delta_{\min}(Q_{21}) \|v\|_2.$$

(б) Покажите, что из вырожденности  $Q_{11}$  следует вырожденность  $Q_{22}$ . Получите эти результаты без использования CS-разложения.

### Замечания и литература к § 2.6

Выход CS-разложения можно найти в

Stewart G. W. (1977). “On the Perturbation of Pseudo-Inverses, Projections and Linear Least Squares Problems”, SIAM Review 19, 634–662.

Другие аспекты этого важного разложения исследуются в статьях

Davis C. and Kahan W. (1970). “The Rotation of Eigenvectors by a Perturbation III”, SIAM J. Num. Anal. 7, 1–46.

Paige C. C. and Saunders M. (1981). “Toward a Generalized Singular Value Decomposition”, SIAM J. Numer. Anal. 18, 398–405.

Вычислительные детали см. в § 8.7.

## 2.7. Чувствительность квадратных систем к возмущениям

В этом разделе мы применяем разработанный в предыдущих параграфах инструментарий к анализу линейной системы  $Ax = b$ , где  $A \in \mathbf{R}^{n \times n}$  – невырожденная

матрица и  $b \in \mathbb{R}^n$ . Наша цель – исследовать, как возмущения в  $A$  и  $b$  влияют на решение  $x$ .

### 2.7.1. SVD-анализ

Если

$$A = \sum_{i=1}^n \sigma_i u_i v_i^T = U \Sigma V^T$$

есть сингулярное разложение матрицы  $A$ , то

$$x = A^{-1}b = (U \Sigma V^T)^{-1}b = \sum_{i=1}^n \frac{u_i^T b}{\sigma_i} v_i. \quad (2.7.1)$$

Это разложение показывает, что малые изменения в  $A$  или в  $b$  могут вызывать относительно большие изменения в  $x$ , если  $\sigma_n$  мало. Действительно, если  $\cos(\theta) = |u_n^T b| / \|b\|_2$  и

$$(A - \varepsilon u_n v_n^T)x = b + \varepsilon \operatorname{sign}(u_n^T b) u_n, \quad \sigma_n > \varepsilon \geq 0,$$

то можно показать, что  $\|y - x\|_2 \geq (\varepsilon/\sigma_n) \|x\|_2 \cos(\theta)$ . Таким образом, возмущения порядка  $O(\varepsilon)$  могут изменить решение на  $\varepsilon/\sigma_n$ .

Влияние абсолютной величины  $\sigma_n$  на чувствительность задачи  $Ax = b$  не должно удивлять нас, поскольку мы помним из теоремы 2.5.2, что  $\sigma_n$  – это расстояние от  $A$  до множества вырожденных матриц. Интуитивно ясно, что по мере приближения к этому множеству решение  $x$  должно быть во все большей степени чувствительным к возмущениям.

### 2.7.2. Обусловленность

Точно измерить чувствительность линейной системы мы сможем, рассмотрев параметризованную систему

$$(A + \varepsilon F)x(\varepsilon) = b + \varepsilon f, \quad x(0) = x,$$

где  $F \in \mathbb{R}^{n \times n}$  и  $f \in \mathbb{R}^n$ . Если  $A$  невырождена, то ясно, что функция  $x(\varepsilon)$  дифференцируема в окрестности нуля. Далее,  $\dot{x}(0) = A^{-1}(f - Fx)$ , и поэтому разложение  $x(\varepsilon)$  в ряд Тейлора имеет вид

$$x(\varepsilon) = x + \varepsilon \dot{x}(0) + O(\varepsilon^2).$$

В любой векторной норме и согласованной с ней матричной норме мы получаем:

$$\frac{\|x(\varepsilon) - x\|}{\|x\|} \leq \varepsilon \|A^{-1}\| \left\{ \frac{\|f\|}{\|x\|} + \|F\| \right\} + O(\varepsilon^2). \quad (2.7.2)$$

Определим для квадратных матриц  $A$  число обусловленности  $\alpha(A)$  соотношением

$$\alpha(A) = \|A\| \|A^{-1}\|, \quad (2.7.3)$$

установившись, что  $\alpha(A) = \infty$  для вырожденных матриц  $A$ . Воспользовавшись неравенством  $\|b\| \leq \|A\| \|x\|$ , мы получим из (2.7.2), что

$$\frac{\|x(\varepsilon) - x\|}{\|x\|} \leq \alpha(A) (\rho_A + \rho_b) + O(\varepsilon^2), \quad (2.7.4)$$

где

$$\rho_A = \varepsilon \frac{\|F\|}{\|A\|}, \quad \rho_b = \varepsilon \frac{\|f\|}{\|b\|}$$

представляют соответственно относительные погрешности  $A$  и  $b$ . Таким образом, относительная погрешность  $x$  может в  $\alpha(A)$  раз превышать относительные погрешности  $A$  и  $b$ . В этом смысле число обусловленности  $\alpha(A)$  количественно характеризует чувствительность задачи  $Ax = b$ .

Отметим, что  $\alpha(\cdot)$  зависит от нормы, в которой оно вычисляется. Когда необходимо подчеркнуть использование какой-либо определенной нормы, мы будем использовать индексы, например

$$\alpha_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_1(A)}{\sigma_n(A)}. \quad (2.7.5)$$

Таким образом, обусловленность матрицы  $A$  в 2-норме является мерой вытянутости гиперэллипсоида  $\{Ax : \|x\|_2 = 1\}$ .

Приведем еще два способа характеристики числа обусловленности. Для чисел обусловленности в  $p$ -норме мы имеем

$$\frac{1}{\alpha_p(A)} = \min_{A+E \text{ вырождена}} \frac{\|E\|_p}{\|A\|_p}. \quad (2.7.6)$$

Этот результат можно найти у Кахана (1966). Он показывает, что  $\alpha_p(A)$  является мерой относительного расстояния от  $A$  до множества вырожденных матриц, измеренного в  $p$ -норме. Мы также имеем для любой нормы:

$$\alpha(A) = \lim_{\delta \rightarrow 0} \sup_{\|E\| \leq \delta \|A\|} \frac{\|(A+E)^{-1} - A^{-1}\|}{\delta} \frac{1}{\|A^{-1}\|}. \quad (2.7.7)$$

Этот впечатляющий на вид результат на самом деле просто свидетельствует о том, что число обусловленности – это нормированная производная Фреше отображения  $A \rightarrow A^{-1}$ . Дальнейшие детали см. Rice (1966). Напомним, что первоначально мы пришли к  $\alpha(A)$  через дифференцирование.

В случае если  $\alpha(A)$  велико, говорят, что  $A$  – плохо обусловленная матрица. Заметим, что это свойство зависит от выбора нормы. Однако в  $\mathbb{R}^{n \times n}$  любые два числа обусловленности эквивалентны в том смысле, что найдутся такие константы  $c_1$  и  $c_2$ , что

$$c_1 \alpha_\alpha(A) \leq \alpha_\beta(A) \leq c_2 \alpha_\alpha(A), \quad A \in \mathbb{R}^{n \times n}.$$

Например, в  $\mathbb{R}^{n \times n}$  мы имеем:

$$\begin{aligned} \frac{1}{n} \alpha_2(A) &\leq \alpha_1(A) \leq n \alpha_2(A), \\ \frac{1}{n} \alpha_\infty(A) &\leq \alpha_2(A) \leq n \alpha_\infty(A), \\ \frac{1}{n} \alpha_1(A) &\leq \alpha_\infty(A) \leq n^2 \alpha_1(A). \end{aligned} \quad (2.7.8)$$

Так что если матрица плохо обусловлена в  $\alpha$ -норме, то она будет плохо обусловлена и в  $\beta$ -норме, с поправкой на упомянутые выше константы  $c_1$  и  $c_2$ .

Для любой  $p$ -нормы мы имеем  $\alpha_p(A) \geq 1$ . Матрицы с малыми числами обусловленности называются хорошо обусловленными. В 2-норме ортогональные матрицы обусловлены наилучшим возможным образом, поскольку  $\alpha_2(Q) = 1$ , если  $Q$  ортогональна.

### 2.7.3. Определители и близость к вырожденности

Естественно задаться вопросом, насколько хорошо величина определителя оценивает обусловленность матрицы. Известно, что  $\det(A) = 0$  эквивалентно вырожденности матрицы  $A$ , но верно ли, что  $\det(A) \approx 0$  означает почти вырожденность? К сожалению, величина определителя  $\det(A)$  и обусловленность задачи  $Ax = b$  слабо связаны между собой. К примеру, определитель матрицы

$$B_n = \begin{Bmatrix} 1 & -1 & \dots & -1 \\ 0 & 1 & \ddots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{Bmatrix} \in \mathbb{R}^{n \times n} \quad (2.7.9)$$

равен единице, но  $\alpha_\infty(B_n) = n2^{n-1}$ . С другой стороны, очень хорошо обусловленная матрица может иметь очень маленький определитель. Например, для матрицы

$$D_n = \text{diag}(10^{-1}, \dots, 10^{-1}) \in \mathbb{R}^{n \times n},$$

$\alpha_p(D_n) = 1$ , хотя  $\det(D_n) = 10^{-n}$ .

### 2.7.4. Точная оценка по норме

Вспомним, что ценность вывода (2.7.4) определялась тем, что он отчетливо выявил связь между  $\alpha(A)$  и скоростью изменения  $x(\varepsilon)$  в точке  $\varepsilon = 0$ . Несколько разочаровывает, однако, зависимость от «достаточной малости»  $\varepsilon$ , а также отсутствие каких-либо сведений о величине слагаемого  $O(\varepsilon^2)$ . В этом и следующем подразделах мы получим еще несколько теорем о возмущениях задачи  $Ax = b$ , в которых оценки уже будут совершенно точными.

Сначала мы докажем полезную лемму, которая в терминах  $\alpha(A)$  указывает, когда мы можем ожидать, что возмущенная система окажется невырожденной.

**Лемма 2.7.1.** *Предположим, что*

$$\begin{aligned} Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad 0 \neq b \in \mathbb{R}^n, \\ (A + \Delta A)y = b + \Delta b, \quad \Delta A \in \mathbb{R}^{n \times n}, \quad \Delta b \in \mathbb{R}^n, \end{aligned}$$

где  $\|\Delta A\| \leq \delta \|A\|$ ,  $\|\Delta b\| \leq \delta \|b\|$ . Если  $\delta \alpha(A) = r < 1$ , то  $A + \Delta A$  невырождена, и

$$\frac{\|y\|}{\|x\|} \leq \frac{1+r}{1-r}.$$

**Доказательство.** Поскольку  $\|A^{-1}\Delta A\| < \delta \|A^{-1}\| \|A\| = r < 1$ , из теоремы 2.3.4 следует, что  $(A + \Delta A)$  невырождена.

Учитывая, что  $(I + A^{-1}\Delta A)y = x + A^{-1}\Delta b$ , находим, используя лемму 2.3.3, что

$$\begin{aligned} \|y\| &\leq (I + A^{-1}\Delta A)^{-1} (\|x\| + \delta \|A^{-1}\| \|b\|) \leq \\ &\leq \frac{1}{1-r} (\|x\| + \delta \|A^{-1}\| \|b\|) = \\ &= \frac{1}{1-r} \left( \|x\| + r \frac{\|b\|}{\|A\|} \right). \end{aligned}$$

Поскольку  $\|b\| = \|Ax\| \leq \|A\| \|x\|$ , то

$$\|y\| \leq \frac{1}{1-r} (\|x\| + r \|x\|). \square$$

Теперь мы готовы к получению точной оценки для возмущения  $Ax = b$ .

**Теорема 2.7.2.** *В условиях леммы 2.7.1*

$$\frac{\|x - y\|}{\|x\|} \leq \frac{2\delta}{1-r} \alpha(A). \quad (2.7.10)$$

*Доказательство.* Так как

$$y - x = A^{-1} \Delta b - A^{-1} \Delta A y, \quad (2.7.11)$$

мы имеем  $\|y - x\| \leq \delta \|A^{-1}\| \|b\| + \delta \|A^{-1}\| \|A\| \|y\|$ , так что

$$\begin{aligned} \frac{\|y - x\|}{\|x\|} &\leq \delta \alpha(A) \frac{\|b\|}{\|A\| \|x\|} + \delta \alpha(A) \frac{\|y\|}{\|x\|} \leq \\ &\leq \delta \alpha(A) \left(1 + \frac{1+r}{1-r}\right) = \frac{2\delta}{1-r} \alpha(A). \square \end{aligned}$$

**Пример 2.7.1.** Задача  $Ax = b$

$$\begin{bmatrix} 1 & 0 \\ 0 & 10^{-6} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 10^{-6} \end{bmatrix}.$$

имеет решение  $x = (1, 1)^T$  и обусловленность  $\alpha_\infty(A) = 10^6$ . Если  $\Delta b = (10^{-6}, 0)^T$ ,  $\Delta A = 0$  и  $(A + \Delta A)y = b + \Delta b$ , то неравенство (2.7.10) в  $\infty$ -норме имеет вид

$$\frac{10^{-6}}{10^0} \leq 10^{-6} 10^6.$$

Таким образом, наша оценка сверху для ошибки, вызванной возмущением, может оказаться сильно завышенной. С другой стороны, если  $\Delta b = (0, 10^{-6})^T$ ,  $\Delta A = 0$  и  $(A + \Delta A)y = b + \Delta b$ , то наше неравенство дает

$$\frac{10^0}{10^0} \leq 2 \times 10^{-6} 10^6.$$

Мы видим, что существуют возмущения, для которых оценка (2.7.10) по существу достигается.

## 2.7.5. Несколько точных покомпонентных оценок

В заключение этого раздела мы покажем, что можно построить более точную теорию возмущений, если ввести в рассмотрение покомпонентные оценки возмущений и использовать обозначение абсолютной величины.

**Теорема 2.7.3.** *Предположим, что*

$$\begin{aligned} Ax &= b, \quad A \in \mathbb{R}^{n \times n}, \quad 0 \neq b \in \mathbb{R}^n, \\ (A + \Delta A)y &= b + \Delta b, \quad \Delta A \in \mathbb{R}^{n \times n}, \quad \Delta b \in \mathbb{R}^n \end{aligned}$$

и что  $|\Delta A| \leq \delta |A|$ ,  $|\Delta b| \leq \delta |b|$ . Если  $\delta \alpha_\infty(A) = r < 1$ , то  $(A + \Delta A)$  невырождена, и

$$\frac{\|y - x\|_\infty}{\|x\|_\infty} \leq \frac{2\delta}{1-r} \|A^{-1}\| |A| \|y\|_\infty.$$

*Доказательство.* Поскольку  $\|\Delta A\|_\infty \leq \delta \|A\|_\infty$  и  $\|\Delta b\|_\infty \leq \delta \|b\|_\infty$ , условия леммы 2.7.1 удовлетворены в  $\infty$ -норме. Из этого следует, что  $A + \Delta A$  невырождена и

$$\frac{\|y\|_\infty}{\|x\|_\infty} \leq \frac{1+r}{1-r}.$$

Теперь, используя (2.7.11), находим

$$\begin{aligned} |y - x| &\leq |A^{-1}| |\Delta b| + |A^{-1}| |\Delta A| |y| \leq \\ &\leq \delta |A^{-1}| |b| + \delta |A^{-1}| |A| |y| \leq \delta |A^{-1}| |A| (|x| + |y|). \end{aligned}$$

Переходя к нормам, получаем

$$\|y - x\|_\infty \leq \delta \|A^{-1}\| |A| \|y\|_\infty \left( \|x\|_\infty + \frac{1+r}{1-r} \|x\|_\infty \right).$$

Разделив последнее неравенство на  $\|x\|_\infty$ , получаем утверждение теоремы.  $\square$

Величину  $\|A||A^{-1}||\|_\infty$  мы будем называть *числом обусловленности по Шкеелю*. Это число было эффективно использовано при анализе некоторых важных преобразований линейных систем. См. § 3.5.

Наконец, приведем результаты Oettli, Prager (1964), показывающие, при каких условиях приближенное решение  $\hat{x} \in \mathbb{R}^n$   $n \times n$ -системы  $Ax = b$  удовлетворяет возмущенной системе с заранее предписанной структурой. В частности, пусть  $E \in \mathbb{R}^{n \times n}$  и  $f \in \mathbb{R}^n$  заданы и имеют неотрицательные элементы. Мы ищем такие  $\delta A \in \mathbb{R}^{n \times n}$ ,  $\delta b \in \mathbb{R}^n$  и  $\omega \geq 0$ , что

$$(A + \delta A)\hat{x} = b + \delta b, \quad |\delta A| \leq \omega E, \quad |\delta b| \leq \omega f. \quad (2.7.12)$$

Заметим, что при надлежащем выборе  $E$  и  $f$  возмущенная система может принимать определенные свойства. Например, если  $E = |A|$ ,  $f = |b|$  и  $\omega$  мало, то  $\hat{x}$  удовлетворяет некоторой покомпонентно близкой системе. Oettli, Prager (1964) показывают, что для заданных  $A$ ,  $b$ ,  $\hat{x}$ ,  $E$  и  $f$  минимально возможное  $\omega$  в (2.7.12) выражается как

$$\omega_{\min} = \max_{1 \leq i \leq n} \frac{|A\hat{x} - b|_i}{(E|\hat{x}| + f)_i}.$$

Если  $A\hat{x} = b$ , то  $\omega_{\min} = 0$ . С другой стороны, если  $\omega_{\min} = \infty$ , то  $\hat{x}$  не удовлетворяет никакой системе с предписанной структурой возмущения.

### Задачи

**2.7.1.** Покажите, что если  $\|I\| > 1$ , то  $\alpha(A) \geq 1$ .

**2.7.2.** Покажите, что для данной нормы  $\alpha(AB) \leq \alpha(A)\alpha(B)$  и что  $\alpha(\alpha A) = \alpha(A)$  для всех ненулевых  $\alpha$ .

### Замечания и литература к § 2.7

Понятие обусловленности тщательно исследуется в

Rice J. (1966). "A Theory of Condition", SIAM J. Num. Anal. 3, 287–310.

Kahan W. (1966). "Numerical Linear Algebra", Canadian Math. Bull. 9, 757–801.

В некоторых приложениях возникает необходимость изучить чувствительность решения задачи  $Ax = b$  по отношению к возмущениям определенной структуры. Например, мы можем интересоваться изменением  $x$  при колебании каждого элемента  $b$  в пределах заданного интервала. Такой более тонкий анализ чувствительности проводится в статье

J. E. Cope and B. W. Rust (1979). "Bounds on solutions of systems with accurate data", SIAM J. Numer. Anal. 16, 950–963.

Основные ссылки по теории покомпонентных возмущений включают

Oettli W. and Prager W. (1964). "Compatibility of Approximate Solutions of Linear Equations with Given Error Bounds for Coefficients and Right Hand Sides", Numer. Math. 6, 405–409.

R. D. Skeel (1979). "Scaling for numerical stability in Gaussian Elimination", J. ACM 26, 494–526.

Обратное к числу обусловленности служит мерой близости данной задачи  $Ax = b$  к вырожденности. Во многих случаях важность информации о том, насколько данная задача близка к труднорешаемой или вообще неразрешимой, получила должную оценку. См.

Barlow J. L. (1986). "On the Smallest Possible Singular Value of an  $M$ -matrix with Applications to Ergodic Markov Chains", SIAM J. Alg. and Disc. Struct. 7, 414–424.

Demmel J. W. (1987a). "On the Distance to the Nearest Ill-Posed Problem", Numer. Math. 51, 251–289.

Demmel J. W. (1987b). "A Counterexample for two Conjectures About Stability", IEEE Trans. Auto. Cont. AC-32, 340–342.

Demmel J. W. (1988). "The Probability that a Numerical Analysis Problem is Difficult", Math. Comp. 50, 449–480.

Higham N. J. (1985). "Nearness Problems in Numerical Linear Algebra", PhD Thesis, University of Manchester, England.

- Higham N. J. (1988b). "Computing a Nearest Symmetric Positive Semidefinite Matrix", Lin. Alg. and Its Applic. 103, 103–118.
- Higham N. J. (1988e). "Matrix Nearness Problems and Applications", Numerical Analysis Report 161, University of Manchester, Manchester, England. To appear in The Proceeding of the JIMA Conference on Applications of Matrix Theory, S. Barnett and M. J. C. Cover (eds), Oxford University Press.
- Laub A. (1985). "Numerical Linear Algebra Aspects of Control Design Computations", IEEE Trans. Auto. Cont. AC-30, 97–108.
- Ruhe A. (1987). "Closest Normal Matrix Found!", BIT 27, 585–598.
- Van Loan C. (1985). "How Near is a Stable Matrix to an Unstable Matrix?", Contemporary Mathematics, Vol. 47, 465–477.

## Глава 3

# Линейные системы общего вида

Проблема решения линейной системы  $Ax = b$  является центральной в научных вычислениях. В этой главе мы остановимся на методике исключения Гаусса – методе, который используют, когда матрица  $A$  квадратная, плотная и без специфики. Если  $A$  не удовлетворяет этим условиям, то представляют интерес алгоритмы из гл. 4, 5 и 10. Параллельные методы решения системы  $Ax = b$  обсуждаются в гл. 6.

Мы приходим к методу исключения Гаусса, обсуждая в § 3.1 ту легкость, с которой можно решать треугольные системы. Приведение системы общего вида к треугольной форме при помощи преобразований Гаусса описывается ниже в § 3.2, где вводится «язык» матричных разложений. К несчастью, полученный метод ведет себя очень плохо на нетривиальном классе задач. Наш анализ ошибок округления в § 3.3 выявляет трудности и подготавливает § 3.4, где вводится концепция выбора ведущих элементов. В последнем разделе мы предлагаем некоторые замечания по поводу важных для практики вопросов, связанных с масштабированием, итерационным уточнением и оценкой обусловленности.

## 3.1. Треугольные системы

Традиционные методы разложений для линейных систем включают в себя приведение исходной квадратной системы к треугольной системе, которая имеет такое же решение. Данный раздел посвящен решению треугольных систем.

### 3.1.1. Прямая подстановка

Рассмотрим следующую нижнюю треугольную  $2 \times 2$ -систему:

$$\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Если  $l_{11}l_{22} \neq 0$ , то неизвестные могут быть определены последовательно:

$$\begin{aligned} x_1 &= b_1/l_{11}, \\ x_2 &= (b_2 - l_{21}x_1)/l_{22}. \end{aligned}$$

Это  $2 \times 2$ -версия алгоритма, известного как *прямая подстановка*. Общую процедуру получаем, разрешая  $i$ -е уравнение системы  $Lx = b$  относительно  $x_i$ :

$$x_i = \left( b_i - \sum_{j=1}^{i-1} l_{ij}x_j \right) / l_{ii}.$$

Если вычисления выполнить для  $i = 1:n$ , то будут получены все компоненты  $x$ . Заметим, что на  $i$ -м шаге необходимо скалярное произведение векторов  $L(i, 1:i-1)$

и  $x(1:i-1)$ . Так как  $b_i$  содержится только в формуле для  $x_i$ , мы можем записать  $x_i$  на месте  $b_i$ <sup>1)</sup>.

**Алгоритм 3.1.1 (Прямая подстановка: строчная версия).** Предположим, что  $L \in \mathbf{R}^{n \times n}$  – нижняя треугольная матрица и  $b \in \mathbf{R}^n$ . Алгоритм заменяет  $b$  на решение системы  $Lx = b$ . Матрица  $L$  должна быть невырожденна.

```

 $b(1) = b(1)/L(1, 1)$ 
for  $i = 2:n$ 
     $b(i) = (b(i) - L(i, 1:i-1)b(1:i-1))/L(i, i)$ 
end
```

Алгоритм требует  $n^2$  флопов. Заметим, что доступ к  $L$  осуществляется по строкам. Вычисленное решение  $\hat{x}$  удовлетворяет

$$(L + F)\hat{x} = b, \quad |F| \leq n\mathbf{u}|L| + O(\mathbf{u}^2). \quad (3.1.1)$$

Этот результат доказывается в Forsythe and Moler (SLE, pp. 104–105). Он говорит о том, что вычисленное решение точно удовлетворяет слабо возмущенной системе. Более того, каждый элемент в матрице возмущения  $F$  мал относительно соответствующего элемента  $L$ .

### 3.1.2. Обратная подстановка

Аналогичный алгоритм для верхней треугольной системы  $Ux = b$  называется *обратной подстановкой*. Вот формула для  $x_i$ :

$$x_i = \left( b_i - \sum_{j=i+1}^n u_{ij}x_j \right) / u_{ii}$$

и снова  $x_i$  можно записать на месте  $b_i$ .

**Алгоритм 3.1.2 (Обратная подстановка: строчная версия).** Если матрица  $U \in \mathbf{R}^{n \times n}$  верхняя треугольная и  $b \in \mathbf{R}^n$ , то следующий алгоритм заменяет  $b$  на решение системы  $Ux = b$ .  $U$  должна быть невырожденной.

```

 $b(n) = b(n)/U(n, n)$ 
for  $i = n-1:-1:1$ 
     $b(i) = (b(i) - U(i, i+1:n)b(i+1:n)) / U(i, i)$ 
end
```

Алгоритм требует  $n^2$  флопов и осуществляет доступ к  $U$  по строкам. Можно показать, что вычисленное решение  $\hat{x}$ , полученное с помощью алгоритма, удовлетворяет

$$(U + F)\hat{x} = b \quad |F| \leq n\mathbf{u}|U| + O(\mathbf{u}^2). \quad (3.1.2)$$

### 3.1.3. Столбцовые версии

Столбцовые версии приведенных выше процедур могут быть получены изменением порядка выполнения циклов. Чтобы понять, что это означает с алгебраической точки зрения, рассмотрим прямую подстановку. Как только компонента  $x_1$  найдена, она может быть удалена из всех уравнений от 2 до  $n$ , и мы продолжим процесс с редуцированной системой  $L(2:n, 2:n)x(2:n) = b(2:n) - x(1)L(2:n, 1)$ . Потом мы вычислим  $x_2$  и исключим ее из всех уравнений от 3 до  $n$ , и т. д. Таким образом, если

<sup>1)</sup> Алгоритмы, в которых результат получается на месте, занимаемом исходными данными, обычно называют алгоритмами с замещением.– Прим. перев.

этот подход применить к

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 5 & 0 \\ 7 & 9 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ 5 \end{bmatrix},$$

мы найдем  $x_1 = 3$  и потом будем иметь дело с  $2 \times 2$ -системой

$$\begin{bmatrix} 5 & 0 \\ 9 & 8 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} -1 \\ -16 \end{bmatrix}.$$

Это полная процедура с замещением.

**Алгоритм 3.1.3 (Прямая подстановка: столбцовая версия).** Предположим, что  $L \in \mathbb{R}^{n \times n}$  нижняя треугольная и  $b \in \mathbb{R}^n$ . Этот алгоритм заменяет  $b$  на решение системы  $Lx = b$ .  $L$  должна быть невырожденной

```

for  $j = 1:n - 1$ 
     $b(j) = b(j)/L(j, j)$ 
     $b(j + 1:n) = b(j + 1:n) - b(j)L(j + 1:n, j)$ 
end
 $b(n) = b(n)/L(n, n)$ 
```

Можно получить также и столбцовую процедуру типа saxpy для обратной подстановки.

**Алгоритм 3.1.4 (Обратная подстановка: столбцовая версия).** Предположим, что  $U \in \mathbb{R}^{n \times n}$  – верхняя треугольная матрица и  $b \in \mathbb{R}^n$ . Алгоритм заменяет  $b$  на решение  $Ux = b$ .  $U$  должна быть невырождена

```

for  $j = n:-1:2$ 
     $b(j) = b(j)/U(j, j)$ 
     $b(1:j - 1) = b(1:j - 1) - b(j)U(1:j - 1, j)$ 
end
 $b(1) = b(1)/U(1, 1)$ 
```

Заметим, что главная операция в обоих алгоритмах 3.1.3 и 3.1.4 – это операция saxpy. Ошибки округления в этих версиях с saxpy ведут себя в основном так же, как и для вариантов со скалярным произведением.

Чаще всего точность решения треугольной системы на удивление хорошая. См. Higham (1988d).

### 3.1.4. Случай нескольких правых частей

Рассмотрим задачу вычисления решения  $X \in \mathbb{R}^{n \times q}$  для системы  $LX = B$ , где  $L \in \mathbb{R}^{n \times n}$  – нижняя треугольная матрица и  $B \in \mathbb{R}^{n \times q}$ . Это задача прямой подстановки в случае *нескольких правых частей*. Покажем, что задача такого рода может быть решена с использованием блочного алгоритма, в котором много матричных умножений, при условии, что  $q$  и  $n$  достаточно велики. Это окажется важным в последующих разделах, где обсуждаются различные блочные разложения. Отметим, что, хотя мы рассматриваем здесь только задачу с нижней треугольной матрицей, все, что говорится, применимо также и для верхнего треугольного случая.

Чтобы реализовать блочный алгоритм прямой подстановки, разобьем систему

$LX = B$  на блоки следующим образом:

$$\begin{bmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \dots & L_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}. \quad (3.1.3)$$

Допустим, что диагональные блоки квадратные. По аналогии с выводом алгоритма 3.1.3, мы решаем систему  $L_{11}X_1 = B_1$  относительно  $X_1$  и далее исключаем  $X_1$  из всех блочных уравнений от 2 до  $N$ :

$$\begin{bmatrix} L_{22} & 0 & \dots & 0 \\ L_{32} & L_{33} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N2} & L_{N3} & \dots & L_{NN} \end{bmatrix} \begin{bmatrix} X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_2 - L_{21}X_1 \\ B_3 - L_{31}X_1 \\ \vdots \\ B_N - L_{N1}X_1 \end{bmatrix}.$$

Продолжая таким образом, мы получим следующую схему прямого блочного saxpy-исключения:

```
for j = 1:N
    Решить  $L_{jj}X_j = B_j$ 
    for i = j + 1:N
         $B_i = B_i - L_{ij}X_j$ 
    end
end
```

Заметим, что  $i$ -цикл содержит один пересчет блочного варианта saxpy следующего вида:

$$\begin{bmatrix} B_{j+1} \\ \vdots \\ B_N \end{bmatrix} = \begin{bmatrix} B_{j+1} \\ \vdots \\ B_N \end{bmatrix} - \begin{bmatrix} L_{j+1,j} \\ \vdots \\ L_{Nj} \end{bmatrix} X_j.$$

Для реализации этого алгоритма в виде умножения матриц на данной архитектуре, очевидно, что разбиение на блоки (3.1.3) должно давать достаточно «большие»  $X_j$ . Допустим, что в этом случае  $X_i$  имеет по крайней мере  $r$  строк. Тогда алгоритм может быть выполнен, если  $N = \text{ceil}(n/r)$ ,  $X_1, \dots, X_{N-1} \in \mathbb{R}^{r \times q}$  и  $X_N \in \mathbb{R}^{(n-(N-1)r) \times q}$ .

### 3.1.5. Доля флопов 3 уровня

Удобно взять на вооружение какую-нибудь меру, дающую количественную оценку для матричных умножений в данном алгоритме. С этой целью определим долю *флопов 3 уровня* в алгоритме, как относительное количество флопов, реализующих матричные умножения. Такие флопы мы назовем *флопами 3 уровня*.

Определим долю флопов 3 уровня для (3.1.4) при простом предположении, что  $n = rN$ . (Аналогичные выводы имеют силу и с неравными блоками, описанными выше.) Так как имеется  $N$  применений прямой подстановки для  $r \times r$ -систем (вычисления реализуются на уровне 2) и общее число операций равно  $n^2$ , то доля флопов 3 уровня аппроксимируется выражением

$$1 - \frac{Nr^2}{n^2} = 1 - \frac{1}{N}.$$

Таким образом, для больших  $N$  почти все флопы – это флопы 3 уровня и поэтому разумно выбрать  $N$  настолько большим, насколько это возможно, при том условии, что предполагаемая архитектура может обеспечить высокий уровень производительности при реализации блочных операций saxpy ширины не меньше  $r = n/N$ .

### 3.1.6. Решение неквадратных треугольных систем

Проблема решения неквадратных,  $m \times n$ , треугольных<sup>1)</sup> систем заслуживает некоторого упоминания. Рассмотрим сначала нижний треугольный случай, когда  $m \geq n$ , т. е.

$$\begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad \begin{array}{l} L_{11} \in \mathbb{R}^{n \times n} \\ L_{21} \in \mathbb{R}^{(m-n) \times n} \end{array} \quad \begin{array}{l} b_1 \in \mathbb{R}^n \\ b_2 \in \mathbb{R}^{m-n} \end{array} .$$

Допустим, что матрица  $L_{11}$  нижняя треугольная и невырожденная. Если мы применим прямую подстановку к  $L_{11}x = b$ , то  $x$  является решением системы в том случае, когда  $L_{21}(L_{11}^{-1}b_1) = b_2$ . Другими словами, не существует решения для всей системы. В такой ситуации может быть подходящей минимизация в смысле наименьших квадратов (см. гл. 5).

Далее рассмотрим нижнюю треугольную систему  $Lx = b$ , когда число столбцов  $n$  превосходит число строк  $m$ . В этом случае применим прямую подстановку к квадратной системе  $L(1:m, 1:m)x(1:m, 1:m) = b$ , а для  $x(m+1:n)$  зададим произвольные значения. Чтобы получить добавочные комментарии по системам, имеющим больше неизвестных, чем уравнений, см. § 5.7.

Рассмотрение неквадратных верхних треугольных систем проводится аналогично. Детали оставляем для читателя.

### 3.1.7. Унитреугольные системы

*Унитреугольная* матрица – это треугольная матрица с единицами на главной диагонали. Многие из треугольных матриц, вычисляемых ниже, обладают этим дополнительным свойством. Очевидно, что при этом не затрудняется реализация указанных выше процедур.

### 3.1.8. Алгебраические свойства треугольных матриц

Мы приведем некоторые свойства произведений и обратных матриц для треугольных и унитреугольных матриц, с целью использовать эти свойства в дальнейшем:

- Обратная к верхней (нижней) треугольной матрице является верхней (нижней) треугольной.
- Произведение двух верхних (нижних) треугольных матриц является верхней (нижней) треугольной.
- Обратная к верхней (нижней) унитреугольной матрице является верхней (нижней) унитреугольной.
- Произведение двух верхних (нижних) унитреугольных матриц является верхней (нижней) унитреугольной.

### Задачи

3.1.1. Предложить алгоритм для вычисления ненулевого  $z \in \mathbb{R}^n$ , такого, что  $Uz = 0$ , где  $U \in \mathbb{R}^{n \times n}$  является верхней треугольной, причем  $u_{nn} = 0$ , а  $u_{11} \dots u_{n-1, n-1} \neq 0$ .

<sup>1)</sup> Такие системы иногда называют трапециевидными. – Прим. перев.

**3.1.2.** Выяснить, как может быть вычислен определитель квадратной треугольной матрицы с наименьшим риском переполнения сверху и снизу.

**3.1.3.** Переписать алгоритм 3.1.4, предполагая, что  $U$  хранится по столбцам в массиве  $u.vec$  длины  $n(n+1)/2$ .

**3.1.4.** Написать детально вариант (3.1.4) без предположения, что  $N$  делится на  $n$ .

**3.1.5.** Доказать все свойства треугольных матриц, приведенные в конце раздела.

### Замечания и литература к § 3.1

Фортран-программу для решения треугольных линейных систем можно найти в LINPACK (гл. 6). Точность решения треугольных систем анализируется в работе

Higham N.J. (1988d). "The Accuracy of Solution to Triangular Systems", Report 158, University of Manchester, Department of Mathematics. (To appear, SIAM J. Numer. Anal.)

## 3.2. LU-разложение

Как мы только что видели, треугольные системы решаются «легко». Идея исключения Гаусса – это преобразование данной системы  $Ax = b$  в эквивалентную треугольную систему. Преобразование достигается составлением соответствующих линейных комбинаций уравнений. Например, в системе

$$\begin{aligned} 3x_1 + 5x_2 &= 9, \\ 6x_1 + 7x_2 &= 4, \end{aligned}$$

умножая первую строку на 2 и вычитая ее из второй, мы получим

$$\begin{aligned} 3x_1 + 5x_2 &= 9, \\ -3x_2 &= -14. \end{aligned}$$

Это и есть исключение Гаусса при  $n = 2$ . Наша цель в данном разделе – дать полное описание этой важной процедуры, причем описать ее выполнение на языке матричных разложений. Данный пример показывает, что алгоритм вычисляет нижнюю унитреугольную матрицу  $L$  и верхнюю треугольную матрицу  $U$  так, что  $A = LU$ , т. е.

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

Решение для исходной задачи  $Ax = b$  находится посредством последовательного решения двух треугольных систем:

$$Ly = b, \quad Ux = u \Rightarrow Ax = LUx = Ly = b.$$

LU-разложение – это «высокий уровень» алгебраического описания исключения Гаусса. Представление результата матричного алгоритма на «языке» матричных разложений полезно. Оно облегчает обобщение и проясняет связь между алгоритмами, которые могут казаться очень разными на скалярном уровне.

### 3.2.1. Матрицы преобразования Гаусса

Чтобы получить разложение, описывающее исключение Гаусса, нам нужно иметь некоторое матричное описание процесса обнуления матрицы. Пусть  $n = 2$ , тогда как  $x_1 \neq 0$  и  $\tau = x_2/x_1$ , то

$$\begin{bmatrix} 1 & 0 \\ -\tau & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ 0 \end{bmatrix}.$$

В общем случае предположим, что  $x \in \mathbb{R}^n$  и  $x_k \neq 0$ . Если

$$\tau^T = (\underbrace{0, \dots, 0}_k, \tau_{k+1}, \dots, \tau_n) \quad \tau_i = \frac{x_i}{x_k} \quad i = k+1:n$$

и мы обозначим

$$M_k = I - \tau e_k^T, \quad (3.2.1)$$

то

$$M_k x = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & & 1 & 0 & & 0 \\ 0 & & -\tau_{k+1} & 1 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\tau_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Матрица  $M_k$  – это матрица преобразования Гаусса. Она является нижней унитрэугольной. Компоненты  $\tau(k+1:n)$  – это множители Гаусса. Вектор  $\tau$  называется вектором Гаусса.

Для реализации данных идей имеется функция, которая вычисляет вектор множителей:

**Алгоритм 3.2.1.** Если  $x \in \mathbb{R}^n$  и элемент  $x_1$  ненулевой, функция вычисляет вектор  $t$  длины  $n-1$ , такой, что если  $M$  – матрица преобразования Гаусса, причем  $M(2:n, 1) = -t$  и  $y = Mx$ , то  $y(2:n) = 0$ .

```
function: t = gauss(x)
    n = length(x)
    t = x(2:n)/x(1)
end gauss
```

Алгоритм требует  $n-1$  флоп.

### 3.2.2. Применение матриц преобразования Гаусса

Умножение на матрицу преобразования Гаусса выполняется достаточно просто. Если матрица  $C \in \mathbb{R}^{n \times r}$  и  $M_k = I - \tau e_k^T$ , тогда преобразование вида

$$M_k C = (I - \tau e_k^T) C = C - \tau (e_k^T C)$$

осуществляет одноранговую модификацию. Кроме того, поскольку  $\tau(1:k) = 0$ , то задействованы лишь элементы матрицы  $C(k+1:n, :)$ . Следовательно, мы получаем

**Алгоритм 3.2.2.** Если матрица  $C \in \mathbb{R}^{n \times r}$  и  $M$  задает  $n \times n$ -преобразование Гаусса, причем  $M(2:n, 1) = -t$ , тогда следующая функция заменяет  $C$  на  $MC$ .

```
function: C = gauss.app(C, t)
    n = rows(C)
    C(2:n, :) = C(2:n, :) - tC(1, :)
end gauss.app
```

Этот алгоритм требует  $2(n-1)r$  флопов. Отметим, что если матрица  $M(k+1:n, k) = -t$ , тогда обращение вида  $C = \text{gauss.app}(C(k:n, :), t)$  заменяет  $C$  на  $M_k C$ .

**Пример 3.2.1**

$$C = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix}, \quad t = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \Rightarrow MC = \begin{bmatrix} 1 & 4 & 7 \\ 1 & 1 & 1 \\ 4 & 10 & 17 \end{bmatrix}.$$

**3.2.3. Свойства ошибок округления в преобразованиях Гаусса**

Обозначим через  $\hat{t}$  вычисленный аналог вектора  $t$  в процедуре **gauss**, тогда нетрудно убедиться, что

$$\hat{t} = t + e, \quad |e| \leq \mathbf{u}|t|.$$

Если вектор  $\hat{t}$  используется в процедуре **gauss.app** и **fl**(( $I - \hat{t}e_1^T$ ) $C$ ) обозначает результат реальных вычислений, то

$$\text{fl}((I - \hat{t}e_1^T)C) = (I - \hat{t}e_1^T)C + E,$$

где

$$|E| \leq 3\mathbf{u}(|C| + |t||C(1, :)|) + O(\mathbf{u}^2).$$

Очевидно, что при большом векторе  $t$  ошибки в вычисленных данных могут быть велики относительно  $|C|$ . По этой причине использование преобразований Гаусса требует тщательной проверки, которая будет рассмотрена ниже в § 3.4.

**3.2.4. Приведение к верхнему треугольному виду**

Допустим, что матрица  $A \in \mathbb{R}^{n \times n}$ . Матрицы преобразования Гаусса  $M_1, \dots, M_{n-1}$ , как правило, можно подобрать так, что матрица  $M_{n-1} \dots M_1 A = U$  является верхней треугольной. Чтобы убедиться в этом, сначала рассмотрим случай для  $n = 3$ . Предположим, что

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

Если

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix},$$

тогда

$$M_1 A = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & -6 & -11 \end{bmatrix}.$$

Аналогично,

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \Rightarrow, \quad M_2(M_1 A) = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix}.$$

Обобщение этого примера позволяет нам представить  $k$ -й шаг следующим образом:

- Мы имеем дело с матрицей  $A^{(k-1)} = M_{k-1} \dots M_1 A$ , которая с 1-го по  $(k-1)$ -й столбец является верхней треугольной.
- Множители Гаусса в  $M_k$  определяются по матрице  $A^{(k-1)}(k+1:n, k)$ .
- Особенno важно для продолжения процесса выполнение условия  $a_{kk}^{(k-1)} \neq 0$ .

Замечая, что полное приведение к верхнему треугольному виду достигается через  $n-1$  шаг, мы получаем следующий алгоритм

```

 $k = 1$ 
while  $A(k, k) \neq 0 \wedge k \leq n-1$ 
     $t = \text{gauss}(A(k:n, k))$ 
     $A(k:n, :) = \text{gauss.app}(A(k:n, :), t)$ 
     $k = k + 1$ 
end

```

(3.2.2)

Элемент  $A(k, k)$  необходимо проверять, чтобы избежать деления на нуль в процедуре **gauss**, в которой вычисляются множители вида  $A(i, k)/A(k, k)$ . Элементы  $A(k, k)$  называются *ведущими элементами*, и их относительная величина оказывается крайне важной.

### 3.2.5. LU-разложение

Если алгоритм (3.2.2) завершается при  $k = n$ , то, говоря на матричном языке, он вычисляет матрицы преобразования Гаусса  $M_1 \dots M_{n-1}$ , такие, что матрица  $M_{n-1} \dots M_1 A = U$  является верхней треугольной. Легко убедиться, что если  $M_k = I - \tau^{(k)} e_k^T$ , тогда обратная к ней задается следующим выражением  $M_k^{-1} = I + \tau^{(k)} e_k^T$  и поэтому

$$A = LU, \quad (3.2.3)$$

где

$$L = M_1^{-1} \dots M_{n-1}^{-1}. \quad (3.2.4)$$

Очевидно, что  $L$  – это нижняя унитреугольная матрица, поскольку каждая матрица  $M_k^{-1}$  является нижней треугольной. Разложение (3.2.4) называется *LU-разложением* матрицы  $A$ . Необходимость проверять в алгоритме (3.2.2) ведущие элементы на нуль говорит о том, что LU-разложение может не существовать. Например, невозможно определить  $l_{ij}$  и  $u_{ij}$  в следующем случае:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{11} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Чтобы убедиться в этом, приравняем соответствующие элементы и заметим, что мы должны получить следующее  $u_{11} = 1$ ,  $u_{12} = 2$ ,  $l_{11} = 2$ ,  $u_{22} = 0$  и  $l_{31} = 1$ . Однако если потом рассмотреть элемент (3.2), то мы придем к противоречивому уравнению  $6 = l_{31}u_{12} + l_{32}u_{22} = 4$ .

Как мы только что показали, нулевой ведущий элемент в алгоритме (3.2.2) можно отождествить с наличием вырожденной главной подматрицы.

**Теорема 3.2.1.** Для матрицы  $A \in \mathbb{R}^{n \times n}$  существует LU-разложение, если  $\det(A(1:k, 1:k)) \neq 0$ , для  $k = 1:n - 1$ . Если LU-разложение существует и  $A$  не вырождена, тогда LU-разложение единственно и  $\det A = u_{11} \dots u_{nn}$ .

**Доказательство.** Предположим, что  $(k - 1)$ -й шаг в алгоритме (3.2.2) выполнен. В начале  $k$ -го шага матрица  $A$  будет заменена матрицей  $M_{k-1} \dots M_1 A = A^{(k-1)}$ . Заметим, что  $a_{kk}^{(k-1)}$  является  $k$ -м ведущим элементом. Поскольку матрицы преобразования Гаусса – нижние унитреугольные, то, как следует из вида ведущей  $k \times k$  части полученного уравнения,  $\det(A(1:k, 1:k)) = a_{11}^{(k-1)} \dots a_{kk}^{(k-1)}$ . Поэтому если  $A(1:k, 1:k)$  не вырождена, то  $k$ -й ведущий элемент является ненулевым.

Для доказательства единственности предположим, что  $A = L_1 U_1$  и  $A = L_2 U_2$  – два LU-разложения невырожденной матрицы  $A$ , тогда  $L_2^{-1} L_1 = U_2 U_1^{-1}$ . Так матрица  $L_2^{-1} L_1$  является нижней унитреугольной, а  $U_2 U_1^{-1}$  – это верхняя треугольная матрица, то отсюда следует, что обе матрицы должны быть равны единичной. Следовательно,  $L_1 = L_2$  и  $U_1 = U_2$ .

И наконец, если  $A = LU$ , то  $\det(A) = \det(LU) = \det(L)\det(U) = \det(U) = u_{11} \dots u_{nn}$ .

### 3.2.6. Несколько практических замечаний

С практической точки зрения существует несколько улучшений, которые могут быть реализованы в алгоритме (3.2.2). Во-первых, поскольку мы уже получили нули в столбцах с 1-го до  $(k - 1)$ -го, то преобразование Гаусса нужно применять только к столбцам с  $k$ -го до  $n$ -го. На самом деле нет необходимости применять преобразование Гаусса также и к  $k$ -му столбцу, так как мы знаем результат. Поэтому эффективным способом для вызова процедуры **gauss.app** является следующий

$$A(k:n, k+1:n) = \text{gauss.app}(A(k:n, k+1:n), t).$$

Другое существенное замечание состоит в том, что множители, задающие матрицу  $M_k$ , могут храниться в позициях, в которых получены нули, т. е. в элементах  $A(k+1:n, k)$ . С учетом этих изменений мы получаем следующую версию алгоритма (3.2.2).

**Алгоритм 3.2.3 (Исключение Гаусса с внешним произведением).** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  обладает таким свойством, что подматрицы  $A(1:k, 1:k)$  невырождены для  $k = 1:n - 1$ . Данный алгоритм вычисляет разложение  $M_{n-1} \dots M_1 A = U$ , где матрица  $U$  является верхней треугольной, а каждая матрица  $M_k$  – это матрица преобразования Гаусса. Матрица  $U$  хранится в верхнем треугольнике матрицы  $A$ . Множители, задающие матрицы  $M_k$ , запоминаются в элементах  $A(k+1:n, k)$ , т. е.  $A(k+1:n, k) = -M_k(k+1:n, k)$ .

```

for  $k = 1:n - 1$ 
     $t = \text{gauss}(A(k:n, k))$ 
     $A(k+1:n, k) = t$ 
     $A(k:n, k+1:n) = \text{gauss.app}(A(k:n, k+1:n), t)$ 
end
```

Этот алгоритм требует  $(2/3)n^3$  флопов и является классической формулировкой метода исключения Гаусса. Заметим, что каждое прохождение через  $k$ -цикл реализует внешнее произведение.

### 3.2.7. Где хранить матрицу $L$ ?

Алгоритм 3.2.3 представляет матрицу  $L$  в виде сомножителей. В частности, если  $t^{(k)}$  – это вектор множителей, задающий матрицу  $M_k$ , тогда на выходе алгоритма мы получим, что  $A(k+1:n, k) = t^{(k)}$ . Одно из наиболее благоприятных совпадений в матричных вычислениях состоит в том, что если  $L = M_1^{-1} \dots M_{n-1}^{-1}$ , тогда  $L(k+1:n, k) = t^{(k)}$ . Этот факт очевиден, если внимательно посмотреть на произведение, задающее матрицу  $L$ . Действительно,

$$L = (I + \tau^{(1)} e_1^T) \dots (I + \tau^{(n-1)} e_{n-1}^T) = I + \sum_{k=1}^{n-1} \tau^{(k)} e_k^T.$$

Поскольку столбец  $A(k+1:n, k)$  содержит  $k$ -й вектор сомножителей  $t^{(k)}$ , отсюда следует, что элементы  $A(i, k)$  содержат  $l_{ik}$  для всех  $i > k$ .

### 3.2.8. Решение линейной системы

После разложения матрицы  $A$  посредством алгоритма 3.2.3 в массиве  $A$  будут храниться матрицы  $L$  и  $U$ . Поэтому мы можем решить систему  $Ax = b$ , последовательно решая треугольные системы  $Ly = b$  и  $Ux = y$  с использованием методов из § 3.1.

**Пример 3.2.3.** Если алгоритм 3.2.3 применен к матрице

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix},$$

тогда на выходе

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & -3 & -6 \\ 3 & 2 & 1 \end{bmatrix}.$$

Если вектор  $b = (1, 1, 1)^T$ , то вектор  $y = (1, -1, 0)^T$  разрешает систему  $Ly = b$ , а вектор  $x = (-1/3, 1/3, 0)^T$  разрешает систему  $Ux = y$ .

### 3.2.9. Гахру-версия LU-разложения

Исключение Гаусса, как и умножение матриц, описывается процедурой, содержащей тройной цикл, который может быть упорядочен разными способами. Алгоритм 3.2.3 соответствует “*кij*” версии исключения Гаусса, действительно, если мы распишем подробнее функцию вызова и внесем незначительные изменения, мы получим алгоритм

```

for  $k = 1:n - 1$ 
     $A(k+1:n, k) = A(k+1:n, k) / A(k, k)$ 
    for  $j = k+1:n$ 
        for  $i = k+1:n$ 
             $A(i, j) = A(i, j) - A(i, k) A(k, j)$ 
        end
    end
end

```

Заметим, что в вычислениях преобладает модификация с внешним произведением.

Теперь мы получим *jki*-вариант исключения Гаусса, который преобладает в гахру-

версиях и прямом исключении. При таком подходе матрицы преобразования Гаусса не изменяют непосредственно всю матрицу  $A$ , как это делается в вариантах с внешним произведением. Их применяют постепенно. Столбец исходной матрицы  $A(:, j)$  остается неизменным до  $j$ -го шага. На данном шаге алгоритма столбец  $A(:, j)$  заменяется на столбец  $M_{j-1} \dots M_1 A(:, j)$ . После чего вычисляется  $j$ -матрица преобразования Гаусса.

Чтобы дать строгое описание алгоритма допустим, что для  $1 \leq j \leq n - 1$  нам известны матрицы  $U(1:j-1, 1:j-1)$  и  $L(:, 1:j-1)$ , т. е. мы знаем первые  $j - 1$  столбцов матриц  $L$  и  $U$ . Для вычисления  $j$ -х столбцов матриц  $L$  и  $U$  мы составим уравнение относительно этих столбцов  $A = LU$ :  $A(:, j) = LU(:, j)$ . Отсюда следует, что

$$\begin{aligned} A(1:j-1, j) &= L(1:j-1, 1:j-1) U(1:j-1, j), \\ A(j:n, j) &= \sum_{k=1}^j L(j:n, k) U(k, j). \end{aligned}$$

Первое уравнение – это нижняя треугольная система, которая может быть решена относительно вектора  $U(1:j-1, j)$ . Когда это выполнено, второе уравнение может быть преобразовано так, что получаются формулы для элемента  $U(j, j)$  и столбца  $L(j+1:n, j)$ , действительно, если мы положим

$$\begin{aligned} v(j:n) &= A(j:n, j) - \sum_{k=1}^{j-1} L(j:n, k) U(k, j) = \\ &= A(j:n, j) - L(j:n, 1:j-1) U(1:j-1, j), \end{aligned}$$

то  $U(j, j) = v(j)$  и  $L(j+1:n)/U(j, j)$ . Таким образом,  $L(j+1:n, j)$  – это скалярная операция gaxpy, и поэтому мы получаем алгоритм

```
for  $j = 1:n$ 
    Решить  $L(1:j-1, 1:j-1) U(1:j-1, j) = A(1:j-1, j)$ 
     $v(j:n) = A(j:n, j) - L(j:n, 1:j-1) U(1:j-1, j)$ 
     $U(j, j) = v(j); L(j+1:n, j) = v(j+1:n)/U(j, j)$ 
end
```

Если мы изложим подробнее этот алгоритм и расположим вычисленные матрицы  $L$  и  $U$  на месте матрицы  $A$ , то мы получим

**Алгоритм 3.2.4. (Gaxpy-версия исключения Гаусса).** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  и главные подматрицы  $A(1:k, 1:k)$  невырождены для  $k = 1:n - 1$ . Данный алгоритм вычисляет разложение  $A = LU$ , где  $L$  – нижняя унитреугольная матрица, а  $U$  – верхняя треугольная. Если  $i > j$ , то элементы  $A(i, j)$  содержат элементы  $L(i, j)$ . Если  $i \leq j$ , то элементы  $A(i, j)$  содержат элементы  $U(i, j)$ .

```
for  $j = 1:n$ 
    {Решить  $L(1:j-1, 1:j-1) U(1:j-1, j) = A(1:j-1, j)$ }
    for  $k = 1:j-1$ 
        for  $i = k+1:j-1$ 
             $A(i, j) = A(i, j) - A(i, k) A(k, j)$ 
        end
    end
     $\{v(j:n) = A(j:n, j) - L(j:n, 1:j-1) U(1:j-1, j)\}$ 
    for  $k = 1:j-1$ 
        for  $i = j:n$ 
             $A(i, j) = A(i, j) - A(i, k) A(k, j)$ 
        end
```

```

end
 $\{L(j+1:n, j) = v(j+1:n)/v(j)\}$ 
 $A(j+1:n, j) = A(j+1:n, j)/A(j, j)$ 
end

```

Этот алгоритм требует  $2n^3/3$  флопов.

### 3.2.10. Модификации Краута и Дулитла

Отметим, что существует *jik*-версия описанной выше процедуры, приводящая к методу со скалярным произведением для LU-разложения. Это более или менее соответствует тому, что известно как варианты Краута и Дулитла исключения Гаусса.

### 3.2.11. Блочное LU-разложение

Исключение Гаусса можно организовать так, что умножение матрицы будет основной операцией. Ключом к исследованию этой блочной процедуры является разбиение матрицы  $A \in \mathbb{R}^{n \times n}$  следующим образом

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ r & n-r \end{bmatrix} \quad \begin{matrix} r \\ n-r \end{matrix},$$

где  $r$  – это блочный параметр. Предположим, что мы вычислили LU-разложение  $L_{11}U_{11} = A_{11}$  и далее решаем треугольные системы с несколькими правыми частями  $L_{11}U_{12} = A_{12}$  и  $L_{21}U_{11} = A_{21}$  для  $U_{12}$  и  $L_{21}$  соответственно. Тогда

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} I_r & 0 \\ 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{bmatrix}, \quad (3.2.6)$$

где  $\tilde{A} = A_{22} - L_{21}U_{12}$ . Отметим, что если  $r \ll n$ , то в вычислениях больше всего требуются флопы 3 уровня при  $r$ -ранговой модификации матрицы  $A_{22}$ . Мы можем повторить процесс для редуцированной матрицы  $\tilde{A}$ . Продолжая аналогичным образом, получим

**Алгоритм 3.2.5 (Блочная версия LU-разложения с внешним произведением).** Пусть  $A \in \mathbb{R}^{n \times n}$  и главные подматрицы  $A(1:k, 1:k)$  невырождены для  $k = 1:n-1$ . Предположим, что  $r$  удовлетворяет условию  $1 \leq r \leq n$ . Следующий алгоритм вычисляет разложение  $A = LU$  посредством  $r$ -ранговой модификации. Причем элементы  $A(i, j)$  замещаются на элементы  $L(i, j)$  при  $i > j$  и элементы  $A(i, j)$  замещаются на  $U(i, j)$  при  $j \geq i$ .

```

 $\lambda = 1$ 
while  $\lambda \leq n$ 
   $\mu = \min(n, \lambda + r - 1)$ 
  Используется алгоритм 3.2.3 для замещения  $A(\lambda:\mu, \lambda:\mu) = \tilde{L}\tilde{U}$  на  $\tilde{L}$  и  $\tilde{U}$ .
  Решаем:  $\tilde{L}Z = A(\lambda:\mu, \mu+1:n)$ . Замещаем:  $A(\lambda:\mu, \mu+1, n) \leftarrow Z$ 
  Решаем:  $W\tilde{U} = A(\mu+1:n, \lambda:\mu)$ . Замещаем:  $A(\mu+1:n, \lambda:\mu) \leftarrow W$ 
   $A(\mu+1:n, \mu+1:n) = A(\mu+1:n, \mu+1:n) - WZ$ 
   $\lambda = \mu + 1$ 
end

```

Алгоритм требует  $2n^3/3$  флопов.

В соответствии с результатами § 3.1.5 давайте рассмотрим долю флопов 3-го

уровня в этой процедуре. Если  $r \ll n$  (а это представляется разумным), то мы можем допустить для простоты, что  $n = rN$ . При реализации  $r \times r$  LU-разложения матрицы  $A(\lambda : \mu, \lambda : \mu) = \tilde{L}\tilde{U}$  требуются только такие флопы, которые не являются флопами 3-го уровня. Поскольку в процессе всего вычисления приходится решать  $N$  таких систем, то очевидно, что доля флопов 3-го уровня будет задаваться соотношением

$$1 - \frac{N(2r^3/3)}{2n^3/3} = 1 - \frac{1}{N^2}.$$

Аналогично можно получить алгоритм блочного LU-разложения на основе базовой операции gaxhr. Заметим, что через  $r$  шагов алгоритма 3.2.4 мы получаем матрицы  $L_{11}$ ,  $L_{21}$  и  $U_{11}$  из (3.2.6). После этого решаем систему  $L_{11}U_{12} = A_{12}$  относительно  $U_{12}$  и вычисляем  $\tilde{A} = A_{22} - L_{21}U_{12}$ . Повторяя этот процесс, мы получим

**Алгоритм 3.2.6 (Блочная gaxhr-версия LU-разложения).** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  имеет ненулевые главные миноры  $A(1:k, 1:k)$  для  $k = 1:n-1$ . И пусть целочисленный параметр  $r$  удовлетворяет соотношению  $1 \leq r \leq n$ . Тогда следующий алгоритм вычисляет разложение  $A = LU$  посредством  $r$ -ранговой модификации. На выходе  $A(i, j)$  замещаются на  $L(i, j)$  для  $i > j$  и  $A(i, j)$  замещаются на  $U(i, j)$  для  $j \geq i$ .

```

 $\lambda = 1$ 
while  $\lambda \leq n$ 
     $\mu = \min(n, \lambda + r - 1)$ 
     $r_1 = \mu - \lambda + 1$ 
    Выполняем  $r_1$  шагов алгоритма 3.2.4 для  $A(\lambda:n, \lambda:n)$ 
    Замещаем  $A(\lambda:\mu, \mu+1:n)$  в соответствии с решением
         $A(\lambda:\mu, \lambda:\mu)Z = A(\lambda:\mu, \mu+1:n)$ 
         $A(\mu+1:n, \mu+1:n) = A(\mu+1:n, \mu+1:n)$ 
         $- A(\mu+1:n, \lambda:\mu)A(\lambda:\mu, \mu+1:n)$ 
     $\lambda = \mu + 1$ 
end
```

Алгоритм требует  $2n^3/3$  флопов.

В алгоритме 3.2.6 использование алгоритма 3.2.4 требуется только для тех флопов, которые не являются флопами 3-го уровня. Поэтому не трудно показать, что в алгоритме 3.2.6 доля флопов 3-го уровня задается соотношением  $1 - 3/(2N)$  при выполнении условия  $n = rN$ . Таким образом, доля флопов 3-го уровня в этой процедуре меньше, чем в алгоритме 3.2.5. Тем не менее если  $N$  достаточно велико, то нет большой разницы между этими двумя процедурами с точки зрения использования флопов 3-го уровня.

### 3.2.12. LU-разложение прямоугольной матрицы

LU-разложение прямоугольной матрицы  $A \in \mathbb{R}^{n \times m}$  может быть выполнено аналогичным образом. Случай для  $m > n$  иллюстрируется уравнением

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \\ 5 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix},$$

а уравнение

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

описывает случай для  $m < n$ . LU-разложение матрицы  $A \in \mathbb{R}^{m \times n}$  всегда существует, если главные подматрицы  $A(1:k, 1:k)$  невырождены для  $k = 1 : \min(m, n)$ .

Чтобы распространить алгоритмы квадратного LU-разложения, описанные выше, на случай прямоугольных матриц, их надо незначительно модифицировать. Например, чтобы реализовать случай  $m \geq n$  мы модифицируем алгоритм 3.2.5 следующим образом

```

for  $j = 1 : \min(m - 1, n)$ 
    Решить  $L(1:j, 1:j)U(1:j, j) =$ 
    относительно  $U(1:i, j)$ 
     $v(j + 1:m) = (A(j + 1:m, j) - L(j + 1:m, 1:j - 1)U(1:j, j))$ 
     $L(j + 1:m, j) = v(j + 1:m)/v(j)$ 
end
```

Алгоритм требует  $mn^2 - n^3/3$  флоков.

### 3.2.13. Несостоятельность метода

Как известно, исключение Гаусса требует невырожденности первых  $(n - 1)$  главных подматриц. Это условие может быть нарушено для довольно простых матриц, например

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

Несмотря на то что матрица  $A$  имеет прекрасное спектральное число обусловленности, для нее нельзя выполнить LU-разложение, поскольку ее главная подматрица вырождена.

Поэтому чтобы эффективно использовать исключение Гаусса для решения линейных систем общего вида, необходимо применить модификации метода. Анализ ошибок округления, изложенный в последующих параграфах, позволяет получить такие модификации.

#### Задачи

**3.2.1.** Предположим, что элементы матрицы  $A(\epsilon) \in \mathbb{R}^{n \times n}$  являются непрерывно дифференцируемыми функциями от параметра  $\epsilon$ . Пусть  $A \equiv A(0)$  и все ее главные подматрицы вырождены. Показать, что для достаточно маленьких  $\epsilon$  матрица  $A(\epsilon)$  имеет LU-разложение  $A(\epsilon) = L(\epsilon)U(\epsilon)$  и обе матрицы  $L(\epsilon)$  и  $U(\epsilon)$  непрерывно дифференцируемы.

**3.2.2.** Предположим, что мы разбили матрицу  $A \in \mathbb{R}^{n \times n}$  следующим образом:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

где  $A_{11}$  – это  $(r \times r)$ -матрица. Пусть  $A_{11}$  невырождена. Тогда матрица  $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$  называется *представлением Шура* матрицы  $A_{11}$  для матрицы  $A$ . Показать, что если  $A_{11}$  имеет LU-разложение, то через  $r$  шагов алгоритма (3.2.3) матрица  $A(r+1:n, r+1:n)$  содержит матрицу  $S$ . Как может быть получена матрица  $S$  через  $r$  шагов алгоритма (3.2.4)?

**3.2.3.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  имеет LU-разложение. Показать, каким образом может быть решена система  $Ax = b$ , без хранения сомножителей, с вычислением LU-разложения  $n \times (n + 1)$  матрицы  $[Ab]$ .

**3.2.4.** Описать вариант исключения Гаусса, который получает нули в столбцах матрицы  $A$  в порядке  $n - 1 : 2$  и который получает разложение  $A = UL$ , где  $U$  – верхняя унитреугольная матрица,  $L$  – нижняя треугольная матрица.

**3.2.5.** Матрицы из  $\mathbb{R}^{n \times n}$  вида  $N(y, k) = I - y'e^T$ ,  $y \in \mathbb{R}^n$ , называются матрицами преобразования

ния Гаусса–Жордана. (а) Дать формулу существования матрицы  $N(y, k)^{-1}$ . (б) Для данного вектора  $x \in \mathbb{R}^n$  найти вектор  $y$ , такой, что  $N(y, k) = e_k$ ? (с) Построить алгоритм, использующий матрицы преобразования Гаусса–Жордана, который замещает матрицу  $A$  на матрицу  $A^{-1}$ . Каковы условия на матрицу  $A$ , обеспечивающие успех вашему алгоритму?

**3.2.6.** Описать детально версию алгоритма 3.2.4, применимую для произвольных  $m \times n$  матриц.

### Замечания и литература к § 3.2

Представления Шура (задача 3.2.2) возникают во многих приложениях. Для расширения практического и теоретического кругозора в этом вопросе см.

Cottle R. W. (1974). "Manifestations of the Shur Complement", Lin. Alg. and Its Applic. 8, 189–211.

В некоторых прикладных областях представления Шура известны как «преобразования Гаусса». Использование матриц преобразования Гаусса–Жордана детально изложено в книге

Fox L. (1964). An Introduction to Numerical Linear Algebra, Oxford University Press, Oxford.

Как мы уже упоминали, версии исключения Гаусса со скалярным произведением давно известны и некоторое время использовались. С этими  $ik$ -вариантами связаны имена Краута и Дулитла. Они были популярны во времена настольных вычислителей, поскольку требуют значительно меньше промежуточных результатов, чем исключение Гаусса. Эти методы все еще достаточно притягательны, так как их можно использовать с накоплением скалярного произведения. Дополнительные ссылки по этому вопросу можно найти в гл. 4 Fox (см. выше) или в работе Stewart (IMC, pp. 131–39). АЛГОЛ-процедура может быть найдена в статье

Bowdler H. J., Martin R. S., Peters G., and Wilkinson J. H. (1966). "Solution of Real and Complex Systems of Linear Equations", Numer. Math. 8, 217–34, See also HACLA; pp. 93–110.

См. также

Forsythe G. E. (1960). "Crout with Pivoting", Comm. ACM 3, 507–8.

McKeeman W. M. (1962). "Grout with Equilibration and Iteration," Comm. ACM. 5, 553–55.

Организация циклов в LU-вычислениях обсуждается в работах

Dayde M. J. and Duff I. S. (1988). "Use of Level-3 BLAS in LU Factorization on the Cray-2, the ETA-IOP, and the IBM 3090-200/VF", Report CSS-229, Computer Science and Systems Division, Harwell Laboratory, Oxon Ox 11 ORA, England.

Dongarra J. J., Gustavson F. G., and Karp A. (1984). "Implementing Linear Algebra Algorithms for Dense Matrices on a Victor Pipeline Machine", SIAM Review 26, 91–112.

Ortega J. M. (1988). "The ijk Forms of Factorization Methods I: Vector Computers", Parallel Computers 7, 135–147.

В первой статье исследуются блочная версия со скалярным произведением, блочная Гахру-версия и блочная версия с внешним произведением исключения Гаусса.

Вычисление LU-разложения большой разреженной матрицы – тема многих исследований. См.

Björk A., Plemmons R. J. and Schneider H. (1981). Large-Scale Matrix Problems, North-Holland, New York.

Duff I. S., Erisman A. M. and Reid J. K. (1986). "Direct Methods for Sparse Matrices", Oxford University Press.

Duff I. S. and Stewart G. W., eds. (1979). "Sparse matrix Proceedings", 1978, SIAM Publications, Philadelphia, PA.

### 3.3. Анализ ошибок округления в методе исключения Гаусса

В первых двух разделах мы оценим влияние ошибок округления, когда алгоритмы используются для решения линейной системы  $Ax = b$ . Прежде чем приступить к этому анализу, полезно рассмотреть ситуацию, близкую к идеальной, при которой не возникает ошибок округления в процессе решения, а ошибки возникают при

задании  $A$  и  $b$ . Таким образом, если  $\text{fl}(b) = b + e$ , а хранимая матрица  $\text{fl}(A) = A + E$  невырождена, то мы допускаем, что вычисленное решение  $\hat{x}$  удовлетворяет соотношению

$$(A + E)\hat{x} = (b + l), \quad \|E\|_\infty \leq \mathbf{u}\|A\|_\infty, \quad \|e\|_\infty \leq \mathbf{u}\|b\|_\infty, \quad (3.3.1)$$

т. е.  $\hat{x}$  является решением системы, «близкой» к точной. Более того, если  $\mathbf{u}k_\infty(A) \leq 1/2$  (например), то, используя теорему 2.7.1, можно показать, что

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq 4\mathbf{u}k_\infty(A). \quad (3.3.2)$$

Оценки (3.3.1) и (3.3.2) являются «наилучшими» оценками по норме. В общем случае не существует анализа ошибок округления в  $\infty$ -норме для решения линейной системы, который требовал бы хранения  $A$  и  $b$  и позволял бы получить более тонкую оценку. Как следствие этого, мы не вправе критиковать алгоритм за нахождение неточного решения  $\hat{x}$ , если матрица  $A$  плохо обусловлена относительно машинной точности, например  $\mathbf{u}k_\infty(A) \approx 1$ .

### 3.3.1. Ошибки в LU-разложении

Давайте сравним оценки ошибок для исключения Гаусса с идеальными оценками, приведенными выше. Для удобства будем использовать бесконечную норму и остановимся на алгоритме 3.2.3 (версии с внешним произведением). Оценки ошибок, которые мы получим, также применимы к алгоритму 3.2.4 (гахру-версии).

Наша первая задача – получить оценки ошибок округления, связанные с вычислением треугольных сомножителей.

**Теорема 3.3.1.** Пусть  $A$  – это  $n \times n$  матрица, заданная в арифметике с плавающей запятой. Если в процессе реализации алгоритма 3.2.3 не возникает нулевых ведущих элементов, тогда вычисленные треугольные матрицы  $\hat{L}$  и  $\hat{U}$  удовлетворяют соотношению

$$\hat{L}\hat{U} = A + H, \quad (3.3.3)$$

$$|H| \leq 3(n-1)\mathbf{u}(|A| + |\hat{L}||\hat{U}|) + O(\mathbf{u}^2). \quad (3.3.4)$$

*Доказательство.* Доказательство проведем по индукции относительно  $n$ . Очевидно, что теорема верна при  $n = 1$ . Пусть она верна для всех  $(n-1) \times (n-1)$  матриц, заданных в арифметике с плавающей запятой. Если

$$A = \begin{bmatrix} a & w^T \\ v & B \\ 1 & n-1 \end{bmatrix},$$

тогда на первом шаге алгоритма вычисляются вектор  $\hat{z} = \text{fl}(v/a)$  и матрица  $\hat{A}_1 = \text{fl}(B - \hat{z}w^T)$ . Поэтому мы имеем

$$\hat{z} = \frac{1}{a}v + f, \quad |f| \leq \mathbf{u} \frac{|v|}{a} \quad (3.3.5)$$

и

$$\hat{A}_1 = B - \hat{z}w^T + F, \quad |F| \leq 2\mathbf{u}(|B| + |\hat{z}||w|^T) + O(\mathbf{u}^2) \quad (3.3.6)$$

Используем алгоритм для вычисления LU-разложения матрицы  $\hat{A}_1$ . По индукции  $\hat{A}_1 = \hat{L}_1\hat{U}_1$  и результирующие треугольные матрицы  $\hat{L}_1$  и  $\hat{U}_1$  удовлетворяют

соотношению

$$\hat{L}_1 \hat{U}_1 = \hat{A}_1 + H_1, \quad (3.3.7)$$

$$|H_1| \leq 3(n-2)\mathbf{u}(|\hat{A}_1| + |\hat{L}_1||\hat{U}_1|) + O(\mathbf{u}^2). \quad (3.3.8)$$

Поэтому

$$\begin{aligned} \hat{L}\hat{U} &\equiv \begin{bmatrix} 1 & 0 \\ \hat{z} & \hat{L}_1 \end{bmatrix} \begin{bmatrix} \alpha & w^T \\ 0 & \hat{U}_1 \end{bmatrix} = \\ &= A + \begin{bmatrix} 0 & 0 \\ \alpha f & H_1 + F \end{bmatrix} \equiv A + H. \end{aligned}$$

Из (3.3.6) следует, что

$$|\hat{A}_1| \leq (1+2\mathbf{u})(|B| + |\hat{z}||w|^T) + O(\mathbf{u}^2),$$

и поэтому, используя (3.3.7) и (3.3.8), мы получаем

$$|H_1 + F| \leq 3(n-1)\mathbf{u}(|B| + |\hat{z}||w|^T + |\hat{L}_1||\hat{U}_1|) + O(\mathbf{u}^2).$$

Так как  $|\alpha f| \leq \mathbf{u}|v|$ , то легко убедиться, что

$$|H| \leq 3(n-1)\mathbf{u} \left\{ \left[ \begin{array}{cc} |\alpha| & |w|^T \\ |S| & |B| \end{array} \right] + \left[ \begin{array}{cc} 1 & 0 \\ |\hat{z}| & |\hat{L}_1| \end{array} \right] \left[ \begin{array}{cc} |\alpha| & |w|^T \\ 0 & |\hat{U}_1| \end{array} \right] \right\} + O(\mathbf{u}^2),$$

откуда следует доказательство теоремы.  $\square$

Отметим, что если матрица  $A$  размера  $m \times n$ , тогда теорема применима при условии, что  $n$  из соотношения (3.3.4) заменяется на меньшее из чисел  $n$  и  $m$ .

### 3.3.2. Решение треугольных систем с приближенными треугольными матрицами

Оценим влияние ошибок округления, если решение треугольных систем с матрицами  $\hat{L}$  и  $\hat{U}$  выполняется методами из § 3.1.

**Теорема 3.3.2.** Пусть  $\hat{L}$  и  $\hat{U}$  вычислены в результате LU-разложения  $n \times n$  матрицы  $A$  в режиме с плавающей запятой посредством алгоритма 3.2.3 или 3.2.4. Предположим, что для нахождения решения  $\hat{y}$  системы  $\hat{L}y = b$  и решения  $\hat{x}$  системы  $\hat{U}x = \hat{y}$  используются методы из § 3.1. Тогда  $(A+E)\hat{x} = b$ , где

$$|E| \leq n\mathbf{u}(3|A| + 5|\hat{L}||\hat{U}|) + O(\mathbf{u}^2). \quad (3.3.9)$$

*Доказательство.* Из соотношений (3.1.1) и (3.1.2) мы имеем

$$(\hat{L} + F)\hat{y} = b, \quad |F| \leq n\mathbf{u}|\hat{L}| + O(\mathbf{u}^2),$$

$$(\hat{U} + G)\hat{x} = \hat{y}, \quad |G| \leq n\mathbf{u}|\hat{U}| + O(\mathbf{u}^2),$$

поэтому

$$(\hat{L} + F)(\hat{U} + G)\hat{x} = (\hat{L}\hat{U} + F\hat{U} + \hat{L}G + FG)\hat{x} = b.$$

Как следует из теоремы 3.3.1,

$$\hat{L}\hat{U} = A + H,$$

где  $|H| \leq 3(n-1)\mathbf{u}(|A| + |\hat{L}||\hat{U}|) + O(\mathbf{u}^2)$ , поэтому, задавая матрицу

$$E = H + F\hat{U} + \hat{L}G + FG,$$

мы получаем  $(A+E)\hat{x} = b$ . Причем

$$\begin{aligned}|E| &\leq |H| + |F||\hat{U}| + |\hat{L}||G| + O(u^2) \leq \\&\leq 3nu(|A| + |\hat{L}||\hat{U}|) + 2nu(|\hat{L}||\hat{U}|) + O(u^2).\end{aligned}$$

Если величина  $|\hat{L}||\hat{U}|$  не слишком велика, то оценка (3.3.9) сопоставима с идеальной оценкой (3.3.1). (Множитель  $n$  не имеет значения; ср. с результатом Уилкинсона из § 2.4.6.) Такая ситуация возможна, так как в исключении Гаусса нет никаких средств для управления возникновением маленьких ведущих элементов. Если маленькие ведущие элементы встречаются, то мы можем ожидать больших чисел в представлении матриц  $\hat{L}$  и  $\hat{U}$ .

Подчеркнем, что маленькие ведущие элементы не являются необходимым следствием плохой обусловленности, что подтверждается примером с матрицей  $A = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix}$ .

Таким образом, исключение Гаусса может давать сколь угодно плохие результаты, даже для хорошо обусловленных матриц. Метод является неустойчивым.

Для того чтобы исправить этот недостаток алгоритма, необходимо в процесс исключения ввести перестановки строк и/или столбцов, с целью удержать в подходящих границах числа, возникающие по ходу вычислений. Эта идея развивается в следующем разделе.

**Пример 3.3.1.** Пусть  $\beta = 10$ ,  $t = 3$  и для решения системы используется приближенная арифметика:

$$\begin{bmatrix} 0.001 & 1.00 \\ 1.00 & 2.00 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 3.00 \end{bmatrix}.$$

Применяя исключение Гаусса, мы получим

$$\hat{L} = \begin{bmatrix} 1 & 0 \\ 1000 & 1 \end{bmatrix}, \quad \hat{U} = \begin{bmatrix} 0.001 & 1 \\ 0 & -1000 \end{bmatrix},$$

причем вычисления показывают, что

$$\hat{L}\hat{U} = \begin{bmatrix} 0.001 & 1 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \equiv A + H.$$

Отметим, что матрица  $6 \begin{bmatrix} 10^{-6} & 0.001 \\ 10^{-3} & 1.001 \end{bmatrix}$  является граничной матрицей из соотношения (3.3.4)

и дает верхнюю оценку для  $|H|$ . Если мы продолжим решение задачи, используя методы для решения треугольных систем из § 3.1 и применяя арифметику той же точности, то мы получим вычисленное решение  $\hat{x} = (0,1)^T$ . Это не совпадает с точным решением  $x = (1.002 \dots, 0.998 \dots)^T$ .

### Задачи

**3.3.1.** Показать, что отмена предположения о том, что матрица  $A$  задана в арифметике с плавающей запятой, как в теореме 3.3.1, приведет к замене в соотношении (3.3.3) коэффициента «3» на «4».

**3.3.2.** Пусть  $A$  – матрица размера  $n \times n$ , а матрицы  $\hat{L}$  и  $\hat{U}$  получены при помощи алгоритма 3.2.3. (а) Сколько требуется флопов для вычисления  $\|\hat{L}\|\|\hat{U}\|_\infty$ ? (б) Доказать, что  $fI(|\hat{L}||\hat{U}|) \leq (1 + 2nu)|\hat{L}||\hat{U}| + Q(u^2)$ ?

**3.3.3.** Пусть  $x = A^{-1}b$ . Доказать, что если  $e = x - \hat{x}$  (ошибка) и  $r = b - A\hat{x}$  (невязка), тогда

$$\frac{\|z\|}{\|A\|} \leq \|e\| \leq \|A^{-1}\| \|r\|.$$

Допустить согласованность между матричной и векторной нормами.

**3.3.4.** Используя двухразрядную с десятичным основанием плавающую арифметику, вычислить  $LU$ -разложение матрицы

$$A = \begin{bmatrix} 7 & 6 \\ 9 & 8 \end{bmatrix}.$$

Что представляет матрица  $H$  из соотношения (3.3.3) для этого примера?

### Замечания и литература к § 3.3

Наш подход к обратному анализу ошибок мы позаимствовали в работе de Boor C. and Pinkus A. (1977). "A Backward Error Analysis for Totally Positive Linear Systems", Numer. Math. 27, 485–90.

Первой публикацией, связанной с анализом ошибок округления в исключении Гаусса, является работа

Wilkinson J. H. (1961). "Error Analysis of Direct Methods of Matrix Inversion", J. ACM 8, 281–330.

В последние годы были получены различные улучшенные оценки и был упрощен анализ. См.

Paige C. C (1973). "An Error Analysis of Method for Solving Matrix Equations", Math. Comp. 27, 355–59.

Reid J. K. (1971). "A Note on the Stability of Gaussian Elimination", J. Inst. Math. Applic. 8, 374–75.

Robertson H. H. (1977). "The Accuracy of Error Estimates for Systems of Linear Algebraic Equations", J. Inst. Math. Applic. 20, 409–14.

### Добавления при переводе § 3.3

Вероятностные свойства ошибок округления впервые рассмотрены в работах

Воеводин В. В. (1969). «Ошибки округления и устойчивость в прямых методах линейной алгебры». – М.: Изд-во МГУ.

Воеводин В. В. (1977). «Вычислительные основы линейной алгебры». – М.: Наука.

## 3.4. Выбор ведущего элемента

Приведенный в предыдущем разделе анализ показывает, что мы должны быть уверены в отсутствии больших элементов в вычисленных треугольных множителях  $\hat{L}$  и  $\hat{U}$ .

Пример

$$A = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10000 & 1 \end{bmatrix} \begin{bmatrix} 0.0001 & 1 \\ 0 & -9999 \end{bmatrix} = LU$$

показывает источник возникших неприятностей: относительно малый ведущий элемент. Перестановкой строк от этих сложностей можно избавиться. В нашем примере, если  $P$  – это матрица перестановок вида

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

то

$$PA = \begin{bmatrix} 1 & 1 \\ 0.0001 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0.0001 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0.9999 \end{bmatrix} = LU.$$

Сейчас треугольные множители содержат достаточно малые элементы.

В этом разделе мы покажем, как определить перестановочный вариант матрицы

*A*, чтобы добиться достаточно устойчивого *LU*-разложения. Существует несколько способов достижения цели, и все они соответствуют разным стратегиям выбора ведущего элемента. Мы остановимся на стратегиях частичного и полного выбора ведущего элемента. Мы обсудим эффективные реализации этих стратегий и их свойства. А начнем с обсуждения применения матрицы перестановок.

### 3.4.1. Перестановочные матрицы

Стабилизация процесса исключения Гаусса, которая обсуждается в данном разделе, предполагает переупорядочивание данных в виде перестановки двух матричных строк. В соответствии с нашим желанием описывать все вычисления в «матричных терминах» нам необходимо ввести понятие *перестановочной матрицы*. Перестановочная матрица – это матрица, отличающаяся от единичной лишь перестановкой строк, например

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

Перестановочную матрицу нет необходимости хранить полностью. Гораздо более эффективно перестановочную матрицу можно представить в виде целочисленного вектора *p* длины *n*. Один из возможных способов такого представления это держать в *p(k)* столбцовый индекс единственной «1» *k*-й строки матрицы *P*. Так вектор *p* = (4123) соответствует кодировке приведенной выше матрицы *P*. Также возможно закодировать *P* указанием столбца той позиции, в которой находится элемент «1», например *p* = (2431).

Если *P* – это матрица перестановок, а *A* – некоторая матрица, тогда матрица *AP* является вариантом матрицы *A* с переставленными столбцами, а *PA* вариантом матрицы *A* с переставленными строками.

Перестановочные матрицы ортогональны, и поэтому если *P* перестановочная матрица, тогда  $P^{-1} = P^T$ . Произведение перестановочных матриц также является перестановочной матрицей.

В этом разделе для нас особый интерес представляют *взаимные перестановки*. Такие перестановки осуществляют матрицы, получаемые простой переменой мест двух строк единичной матрицы, например

$$E = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Взаимные перестановки могут использоваться для описания перестановок строк и столбцов матрицы. В приведенном выше примере порядка  $4 \times 4$  матрица *EA* отличается от матрицы *A* перестановкой 1-й и 4-й строк. Аналогично матрица *AE* отличается от матрицы *A* перестановкой 1-го и 4-го столбцов.

Если *P* =  $E_n \dots E_1$  и каждая матрица *E\_k* является единичной с переставленными *k* и *p(k)* строками, тогда вектор *p(1:n)* содержит всю необходимую информацию о матрице *P*. Действительно, вектор  $x \in \mathbb{R}^n$  может быть замещен на вектор *Px* следующим образом:

```
for k = 1:n
    x(k) ↔ x(p(k))
end
```

Здесь символ « $\leftrightarrow$ » обозначает «выполнение перестановки».

$$x(k) \leftrightarrow x(p(k)) \Leftrightarrow r = x(k), \quad x(k) = x(p(k)), \quad x(p(k)) = r.$$

Поскольку каждая матрица  $E_k$  является симметричной и  $P^T = E_1 \dots E_n$ , то также можно выполнить замещение вектора  $x$  на вектор  $P^T x$ :

```
for k = n : -1 : 1
    x(k) ↔ x(p(k))
end
```

Необходимо отметить, что операции перестановок не принадлежат к операциям плавающей арифметики. Тем не менее, операции с матрицами перестановок часто приводят к нерегулярной сортировке данных и могут потребовать значительных вычислительных затрат.

### 3.4.2. Частичный выбор ведущего элемента: общая идея

Мы показали, как можно использовать взаимные перестановки в  $LU$ -разложении, чтобы обеспечить появление множителей Гаусса, по абсолютной величине не превосходящих единицу. Пусть матрица

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}.$$

Чтобы добиться наименьших множителей в первой матрице разложения по Гауссу с помощью взаимных перестановок строк, надо сделать элемент  $a_{11}$  наибольшим в первом столбце. Если  $P_1$  – матрица взаимных перестановок, тогда

$$P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Поэтому

$$P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ -3 & 17 & 10 \end{bmatrix}$$

и

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix} \Rightarrow M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix}.$$

Теперь, чтобы получить наименьший множитель в матрице  $M_2$ , необходимо переставить 2-ю и 3-ю строки. Поэтому если

$$P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ и } M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/4 & 1 \end{bmatrix},$$

то

$$M_2 P_2 M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix}.$$

Пример иллюстрирует общую идею, основанную на перестановке строк. Обобщая эту идею, мы получаем следующую схему:

**for**  $k = 1:n - 1$

Построить матрицу взаимных перестановок  $E_k$ , такую,  
что если  $z$  – это  $k$ -й столбец матрицы  $E_k A$ , то  
 $|z(k)| = \|z(k:n)\|_\infty$ .

$A = E_k A$

Построить матрицу преобразования Гаусса  $M_k$ , такую,  
что если  $v$  – это  $k$ -й столбец матрицы  $M_k A$ , то  
 $v(k+1:n) = 0$ .

$A = M_k A$

**end**

Такая стратегия, основанная на перестановке строк, называется *стратегией частичного выбора*. По окончании у нас образуется верхняя треугольная матрица  $M_{n-1}E_{n-1}\dots M_1E_1A = U$ .

В результате применения стратегии частичного выбора мы получаем множители, не превосходящие по абсолютной величине единицу. Это следует из того, что

$$|(E_k M_{k-1} \dots M_1 E_1 A)_{kk}| = \max_{k \leq i \leq n} |(E_k M_{k-1} \dots M_1 E_1 A)_{ik}|$$

для  $k = 1:n - 1$ . Таким образом, стратегия частичного выбора является надежной защитой от появления чрезмерно больших множителей.

### 3.4.3. Детали стратегии частичного выбора

Давайте рассмотрим полностью метод исключения Гаусса (G.E.) в сочетании со стратегией частичного выбора.

**Алгоритм 3.4.1. (G.E. с частичным выбором: версия с внешним произведением).** Если матрица  $A \in \mathbb{R}^{n \times n}$ , то данный алгоритм вычисляет матрицы преобразования Гаусса  $M_1 \dots M_{n-1}$  и матрицы взаимных перестановок  $E_1 \dots E_{n-1}$ , такие что матрица  $M_{n-1}E_{n-1}\dots M_1E_1A = U$  является верхней треугольной. При этом нет множителей, превосходящих 1 по абсолютной величине. Подматрица  $A(1:k, k)$  замещается на матрицу  $U(1:k, k)$ ,  $k = 1:n$ . Подматрица  $A(k+1:n, k)$  замещается на матрицу  $-M_k(k+1:n, k)$ ,  $k = 1:n - 1$ . Целочисленный вектор  $piv(1:n - 1)$  задает взаимные перестановки. В частности, матрица  $E_k$  представляет строки  $k$  и  $piv(k)$ ,  $k = 1:n - 1$ .

**for**  $k = 1:n - 1$

Зададим  $\mu$ , такое что  $k \leq \mu \leq n$  и  $|A(\mu k)| =$

$\|A(k:n, k)\|_\infty$

$A(k, k:n) \leftrightarrow A(\mu, k:n); piv(k) = \mu$

**if**  $A(k, k) \neq 0$

$t = gauss(A(k:n, k)); A(k+1:n, k) = t$

$A(k:n, k+1:n) = gauss.app(A(k:n, k+1:n), t)$

**end**

**end**

Заметим, что если на  $k$ -м шаге  $\|A(k:n, k)\|_\infty = 0$ , то в условиях точной арифметики первые  $k$  столбцов матрицы  $A$  линейно зависимы. В отличие от алгоритма 3.2.3, здесь это нам не доставляет трудностей. Мы просто обойдем нулевой ведущий элемент.

Дополнительные затраты, вызванные применением стратегии частичного выбора, являются минимальными относительно затрат плавающей арифметики, так как

требуют лишь  $O(n^2)$  операций сравнения для поиска ведущего элемента. В целом алгоритм требует  $2n^3/3$  флопов.

Чтобы решить линейную систему  $Ax = b$  после вызова алгоритма 3.4.1, мы должны

- Вычислить вектор  $y = M_{n-1}E_{n-1}\dots M_1E_1b$ .
- Решить верхнюю треугольную систему  $Ux = y$ .

Вся необходимая для этого информация содержится в массиве  $A$  и целочисленном векторе  $piv$ .

**Пример 3.4.1.** Если алгоритм 3.4.1 применить к матрице  $A$  из § 3.4.2, тогда на выходе мы получим

$$A = \begin{bmatrix} 6 & 18 & -12 \\ 1/3 & 8 & 16 \\ -1/2 & 1/4 & 6 \end{bmatrix}$$

и  $piv = (3, 3)$ .

### 3.4.4. Где хранить матрицу $L$ ?

Метод исключения Гаусса вычисляет  $LU$ -разложение матрицы  $A$  с переставленными строками. Доказательством являются следующие рассуждения.

**Теорема 3.4.1.** Если метод Гаусса с частичным выбором используется для вычисления верхней триангуляризации вида

$$M_{n-1}E_{n-1}\dots M_1E_1A = U \quad (3.4.2)$$

посредством алгоритма 3.4.1, то

$$PA = LU,$$

где  $P = E_{n-1}\dots E_1$  и  $L$  – это нижняя унитреугольная матрица с элементами  $|l_{ij}| < 1$ .  $k$ -й столбец матрицы  $L$  ниже диагонали является перестановочной версией  $k$ -го вектора Гаусса. В частности, если  $M_k = I - t^{(k)}l_k^t$ , то  $L(k+1:n, k) = g(k+1:n)$ , где  $g = E_{n-1}\dots E_{k+1}t^{(k)}$ .

**Доказательство.** Преобразование формулы (3.4.2) позволяет получить соотношение вида  $M_{n-1}\dots \tilde{M}_1PA = U$ , где  $\tilde{M}_{n-1} = M_{n-1}$  и

$$\tilde{M}_k = E_{n-1}\dots E_{k+1}M_kE_{k+1}\dots E_{n-1}, \quad k \leq n-2.$$

Так как каждая матрица  $E_j$  является матрицей взаимных перестановок, меняющей  $j$ -ю строку с  $\mu$ -й строкой, где  $\mu \geq j$ , то матрица  $E_j(1:j-1, 1:j-1) = I_{j-1}$ . Отсюда следует, что каждая матрица  $\tilde{M}_k$  – это матрица преобразования Гаусса с вектором Гаусса  $t^{(k)} = E_{n-1}\dots E_{k+1}t^{(k)}$ . Опираясь на результат теоремы, легко показать, как можно изменить алгоритм 3.4.1, чтобы на выходе в матрице  $A(i, j)$  содержалась матрица  $L(i, j)$  для всех  $i > j$ . Мы просто применим преобразование, задаваемое матрицей  $E_k$ , ко всем ранее вычисленным векторам Гаусса. Это достигается изменением записи “ $A(k, k:n) \leftrightarrow A(\mu, k:n)$ ” в алгоритме 3.4.1 на запись “ $A(k, 1:n) \leftrightarrow A(\mu, 1:n)$ ”.

### 3.4.5. Гахру-версия

В § 3.2 мы рассмотрели схему с внешним произведением и гахру-версию для вычисления  $LU$ -разложения. Объединив выбор ведущего элемента и схему с внеш-

ним произведением, естественно было бы это же выполнить и для гахру-версии. Возьмем из алгоритма (3.2.5) общую структуру LU-процесса для гахру-версии.

```

for  $j = 1:n$ 
    Решить  $L(1:j-1, 1:j-1)U(1:j-1, j) = A(1:j-1, j)$ 
     $v(j:n) = A(j:n, j) - L(j:n, 1:j-1)U(1:j-1, j)$ 
     $U(j, j) = v(j)$ 
     $L(j+1:n, j) = v(j+1:n)/U(j, j)$ 
end
```

В сочетании со стратегией частичного выбора мы получаем следующий алгоритм

```

for  $j = 1:n$ 
     $A(:, j) = E_{j-1} \dots E_1 A(:, j)$ 
    Решить  $L(1:j-1, 1:j-1)U(1:j-1, j) = A(1:j-1, j)$ 
     $v(j:n) = A(j:n) - L(j:n, 1:j-1)U(1:j-1, j)$ 
    Определить  $\mu$ ,  $j \leq \mu \leq n$ , такое что  $|v(\mu)| = \|v(j:n)\|_\infty$ 
    Пусть  $E_j$  меняет строки  $j$  и  $\mu$  и переставляет:  $(j) \leftrightarrow (\mu)$ .
    if  $v(j) \neq 0$ 
         $v(j+1:n) = v(j+1:n)/(j)$ 
    end
     $U(j, j) = v(j)$ 
     $L(j+1:n, j) = v(j+1:n)$ 
     $L(:, 1:j-1) = E_j L(:, 1:j-1)$ 
end
```

Если наполнить схему деталями и учесть замещение матрицы  $A$ , то получим

**Алгоритм 3.4.3. (G.E. с частичным выбором: Гахру-версия).** Алгоритм вычисляет нижнюю унитреугольную матрицу  $L$ , верхнюю треугольную матрицу  $U$  и матрицу перестановок  $P$ , такие, что  $PA = LU$ . Матрица  $A(i, j)$  замещается на матрицу  $L(i, j)$ , если  $i > j$ , и на матрицу  $U(i, j)$  в противном случае. Матрица  $P = E_{n-1} \dots E_1$  задается целочисленным вектором  $piv(1:n-1)$ . В частности, матрица  $E_j$  выполняет взаимные перестановки строк  $j$  и  $piv(j)$ .

```

for  $j = 1:n$ 
     $\{A(:, j) = E_{j-1} \dots E_1 A(:, j)\}$ 
    for  $k = 1:j-1$ 
         $A(k, j) \leftrightarrow A(piv(k), j)$ 
    end
     $\{\text{Решить } L(1:j-1, 1:j-1)U(1:j-1, j) = A(1:j-1, j)\}$ 
    for  $k = 1:j-1$ 
        for  $i = k+1:j-1$ 
             $A(i, j) = A(i, j) - A(i, k)A(k, j)$ 
        end
    end
     $\{v(j:n) = A(j:n, j) - L(j:n, 1:j-1)U(1:j-1, j)\}$ 
    for  $k = 1:j-1$ 
        for  $i = j:n$ 
             $A(i, j) = A(i, j) - A(i, k)A(k, j)$ 
        end
    end
    end
     $\{\text{Определить } E_j \text{ и применить к } v \text{ и } L(:, 1:j-1)\}$ 
    Найти  $\mu$ , такое, что  $j \leq \mu \leq n$  и  $|A(\mu, j)| = \|A(j:n, j)\|_\infty$ 
     $piv(j) = \mu$ 
```

```

for  $k = 1:j$ 
     $A(j, k) \leftrightarrow A(\mu, k)$ 
end
 $\{L(j+1:n, j) = v(j+1:n)/v(j)\}$ 
if  $A(j, j) \neq 0$ 
    for  $i = j+1:n$ 
         $A(i, j) = A(i, j)/A(j, j)$ 
    end
end
end

```

Как и версия с внешним произведением, эта процедура требует  $2n^3/3$  флопов и  $O(n^2)$  сравнений. Чтобы решить систему  $Ax = b$  после применения алгоритма 3.4.3, мы должны (a) положить  $c = Pb$  (выполнить перестановки компонент вектора  $b$ ), (b) решить систему  $Ly = c$  (нижняя треугольная система) и (c) решить систему  $Ux = y$  (верхняя треугольная система).

**Пример 3.4.2.** Если алгоритм 3.4.2 применить к матрице  $A$  из примера 3.4.1, то на выходе мы получим

$$A = \begin{bmatrix} 0 & 18 & -12 \\ 1/3 & 8 & 16 \\ 1/2 & 1/4 & 6 \end{bmatrix}$$

и  $piv = (3, 3)$ .

### 3.4.6. Анализ ошибок округления

Давайте проверим на устойчивость метод со стратегией частичного выбора. Для этого надо подсчитать ошибки округления, которые появляются в процессе исключения и в процессе решения треугольных систем. Учитывая, что перестановки не дают ошибок округления, несложно показать, с учетом теоремы 3.3.2, что вычисленное решение  $\hat{x}$  удовлетворяет соотношению  $(A + E)\hat{x} = b$ , где

$$|E| \leq n\mathbf{u}(3|A| + 5\hat{P}^T|\hat{L}||\hat{U}|) + O(u^2). \quad (3.4.3)$$

При этом мы считаем, что матрицы  $\hat{P}$ ,  $\hat{L}$  и  $\hat{U}$  вычислены аналогично матрицам  $P$ ,  $L$  и  $U$ , в соответствии с приведенным выше алгоритмом. Элементы матрицы  $\hat{L}$  ограничены сверху единицей вследствие выбора ведущего элемента. Поэтому  $\|\hat{L}\|_\infty \leq n$ , и мы имеем оценку

$$\|E\|_\infty \leq n\mathbf{u}(3\|A\|_\infty + 5n\|\hat{U}\|_\infty) + O(\mathbf{u}^2). \quad (3.4.4)$$

Наша задача оценить  $\|\hat{U}\|_\infty$ . Определим понятие *фактор роста*  $\rho$  следующим образом

$$\rho = \max_{i, j, k} \frac{|\hat{a}_{ij}^{(k)}|}{\|A\|_\infty}, \quad (3.4.5)$$

где  $\hat{A}^{(k)}$  – вычисленный аналог матрицы  $A^{(k)} = M_k E_k \dots M_1 E_1 A$ . Отсюда следует, что

$$\|E\|_\infty \leq 8n^3\rho\|A\|_\infty\mathbf{u} + O(\mathbf{u}^2) \quad (3.4.6)$$

Может ли эта оценка быть сопоставимой с идеальной оценкой (3.3.1), зависит от величины фактора роста  $\rho$ . (Множитель  $n^3$  на практике не является единственным и в наших суждениях может быть опущен.) Фактор роста является мерой величины чисел, появившихся в процессе исключения. На практике,  $\rho$  обычно бывает порядка

10, но оно может быть и таким же большим как  $2^{n-1}$ . Несмотря на это, большинство специалистов по численному анализу считает, что случайная последовательность растущих элементов в исключении Гаусса с частичным выбором ведет себя близко к практике. Метод является надежным для использования.

**Пример 3.4.3.** Если метод исключения Гаусса с частичным выбором использовать при решении задачи

$$\begin{bmatrix} 0.001 & 1.00 \\ 1.00 & 2.00 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 3.00 \end{bmatrix}$$

при  $\beta = 10$ ,  $t = 3$  в приближенной арифметике, то

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \hat{L} = \begin{bmatrix} 1.00 & 0 \\ 0.001 & 1.00 \end{bmatrix}, \quad \hat{V} = \begin{bmatrix} 1.00 & 2.00 \\ 0 & 1.00 \end{bmatrix}$$

и  $\hat{x} = (1.00, 0.998)$  имеет правильные три цифры.

**Пример 3.4.4.** Если матрица  $A \in \mathbb{R}^{n \times n}$  задается формулами

$$a_{ij} = \begin{cases} 1, & \text{если } i = j \text{ или } j = n, \\ -1, & \text{если } i > j, \\ 0 & \text{для остальных } i, j, \end{cases}$$

тогда в LU-разложении матрицы  $A$  элементы  $|e_{ij}| \leq 1$ , а  $u_{nn} = 2^{n-1}$

### 3.4.7. Метод блочного исключения Гаусса

Исключения Гаусса с частичным выбором можно организовать так, что в нем будет много операций 3-го уровня. Мы детально исследуем блочную процедуру с внешним произведением, но отметим, что также возможно блочное представление гахру-версии и версии со скалярным произведением. См. Dayde и Duff (1988).

Пусть матрица  $A \in \mathbb{R}^{n \times n}$  и для простоты  $n = rN$ . Разобьем матрицу  $A$  следующим образом:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ r & n-r \end{bmatrix}_{n-r}.$$

Первый шаг блочного преобразования является типичным и выполняется следующим образом:

- Используем скалярный метод исключения Гаусса с частичным выбором (например, прямоугольные версии алгоритмов 3.4.1 или 3.4.2), чтобы вычислить матрицу перестановок  $P_1 \in \mathbb{R}^{n \times n}$ , нижнюю унитреугольную матрицу  $L_{11} \in \mathbb{R}^{n \times n}$  и верхнюю треугольную матрицу  $U_{11} \in \mathbb{R}^{n \times r}$ , так чтобы

$$P_1 \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}.$$

- Преобразуем посредством матрицы  $P_1$  остальную часть матрицы  $A$ :

$$\begin{bmatrix} \tilde{A}_{12} \\ \tilde{A}_{22} \end{bmatrix} = P_1 \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}.$$

- Решаем систему с нижней треугольной матрицей и многими правыми частями

$$L_{11} U_{12} = \tilde{A}_{12}.$$

- Выполняем преобразование 3-го уровня

$$\tilde{A} = \tilde{A}_{22} - L_{21}U_{12}.$$

В результате этих вычислений мы получаем разложение

$$P_1 A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} I_r & 0 \\ 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{bmatrix}.$$

Далее процесс повторяется для первых  $r$  столбцов матрицы  $A$ .

В целом в течение  $k$  шагов ( $1 \leq k \leq N-1$ ) блочного алгоритма мы применяем скалярный метод исключения Гаусса к матрице порядка  $(n-(k-1)r) \times r$ . Решаем  $r \times (n-kr)$  систему со многими правыми частями и выполняем преобразование 3-го уровня размера  $(n-kr) \times (n-kr)$ . Вклад операции 3-го уровня в общий процесс аппроксимируется соотношением  $1 - 3/(2N)$ . Поэтому для больших  $N$  процедура содержит много операций 3-го уровня.

### 3.4.8. Полный выбор ведущего элемента

Другая стратегия выбора ведущего элемента, называемая *полным выбором*, обладает тем свойством, что возможная оценка фактора роста значительно меньше, чем  $2^{n-1}$ . Заметим, что при частичном выборе  $k$ -й ведущий элемент определяется исследованием текущего подстолбца  $A(k:n, k)$ . При полном выборе в позицию  $(k, k)$  помещается наибольший элемент текущей подматрицы  $A(k:n, k:n)$ . Следовательно, мы вычисляем верхнюю триангуляризацию  $M_{n-1}E_{n-1}\dots M_1E_1AF_1\dots F_{n-1} = U$  таким образом, что на  $k$ -м шаге имеем дело с матрицей

$$A^{(k-1)} = M_{k-1}E_{k-1}\dots M_1E_1AF_1\dots F_{k-1}$$

и определяем матрицы перестановок  $E_k$  и  $F_k$  из условий

$$|(E_k A^{(k-1)} F_k)_{kk}| = \max_{k \leq i, j \leq n} |(E_k A^{(k-1)} F_k)_{ij}|.$$

Имеет место аналог теоремы 3.4.1.

**Теорема 3.4.2.** Если исключение Гаусса с полным выбором ведущего элемента используется для вычисления верхней триангуляризации

$$M_{n-1}E_{n-1}\dots M_1E_1AF_1\dots F_{n-1} = U, \quad (3.4.7)$$

то

$$PAQ = LU,$$

где  $P = E_{n-1}\dots E_1$ ,  $Q = F_1\dots F_{n-1}$ ; а  $L$  – нижняя унитреугольная матрица с элементами  $|l_{ij}| \leq 1$ ;  $k$ -й столбец матрицы  $L$  ниже диагонали – это переставленный вариант  $k$ -го вектора Гаусса. В частности, если  $M_k = I - t^{(k)}l_k^T$ , то  $L(k+1:n, k) = g(k+1:n)$ , где  $g = E_{n-1}\dots E_{k+1}t^{(k)}$ .

**Доказательство.** Доказательство аналогично доказательству теоремы 3.4.1. Детали оставляем для читателя.

Вот подробный алгоритм исключения Гаусса с полным выбором:

**Алгоритм 3.4.3 (Г.Е. с полным выбором ведущего элемента).** Этот алгоритм вычисляет разложение с полным выбором  $PAQ = LU$ , где  $L$  – нижняя унитреугольная матрица, а  $U$  – верхняя треугольная. Матрицы  $P = E_{n-1}\dots E_1$  и  $Q = F_1\dots F_{n-1}$  осуществляют взаимные перестановки. Подматрица  $A(1:k, k)$  замещается матрицей  $U(1:k, k)$ ,  $k = 1:n$ . Подматрица  $A(k+1:n, k)$  замещается матрицей  $L(k+1:n, k)$ ,

$k = 1, n - 1$ . Матрица  $E_k$  переставляет строки  $k$  и  $p(k)$ . Матрица  $F_k$  переставляет столбцы  $k$  и  $q(k)$ .

**for**  $k = 1:n - 1$

    Определить  $\mu$  и  $\lambda$ , такие, что

$$|A(\mu, \lambda)| = \max \{|A(i, j)| : i = k:n, j = k:n\}$$

$$A(k, 1:n) \leftrightarrow A(\mu, 1:n); A(1:n, k) \leftrightarrow A(1:n, \lambda)$$

$$p(k) = \mu; q(k) = \lambda$$

**if**  $A(k, k) \neq 0$

$$t = \text{gauss}(A(k:n, k))$$

$$A(k + 1:n, k) = t$$

$$A(k:n, k + 1:n) = \text{gauss.app}(A(k:n, k + 1:n), t)$$

**end**

**end**

Алгоритм требует  $2n^3/3$  флопов и такое же количество сравнений. В отличие от частичного, полный выбор ведущего элемента приводит к значительному увеличению вычислительных затрат, поскольку на каждом этапе выполняется двумерный поиск.

### 3.4.9. Замечания по стратегии с полным выбором ведущего элемента

Пусть  $\text{rank}(A) = r < n$ . Из этого следует, что в начале  $(r + 1)$ -го шага подматрица  $A(r + 1:n, r + 1:n) = 0$ . Это означает, что  $E_k = F_k = M_k = I$  для  $k = r + 1:n$  и поэтому алгоритм может быть завершен через  $r$  шагов в виде разложения такого рода

$$PAQ = LU = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix}.$$

Здесь  $L_{11}$  и  $U_{11}$  – это  $r \times r$  матрицы, а  $L_{21}$  и  $U_{12}$  – матрицы  $(n - r) \times r$ . Таким образом, исключение Гаусса с полным выбором ведущего элемента может быть в принципе использовано для определения ранга матрицы. Однако ошибки округления создают возможность изменения точного нулевого значения ведущего элемента. На практике можно «объявить», что матрица  $A$  имеет ранг  $k$ , если ведущий элемент на  $k + 1$  шаге достаточно мал. Подробно задача численного определения ранга матрицы обсуждается в § 5.4.

Уилкинсон (1961) показал, что в условиях точной арифметики элементы матрицы  $A^{(k)} = M_k E_k \dots M_1 E_1 A F_1 \dots F_k$  удовлетворяют соотношению

$$|a_{ij}^{(k)}| \leq k^{1/2} (2 \cdot 3^{1/2} \dots k^{1/(k-1)})^{1/2} \max |a_{ij}|. \quad (3.4.8)$$

Верхняя граница этого неравенства является медленно растущей функцией от  $k$ . Этот факт в сочетании с обширным эмпирическим обоснованием позволяет предположить, что величина  $\rho$  всегда умеренна (например,  $\rho = 10$ ), и это позволяет нам сделать вывод об устойчивости исключения Гаусса с полным выбором ведущего элемента. Этот метод решает слабо возмущенную линейную систему  $(A + E)\hat{x} = b$  точно в смысле соотношения (3.3.1). Тем не менее не имеется практических обоснований для предпочтения стратегии полного выбора перед частичным выбором, исключая случай, когда стоит вопрос об определении ранга матрицы.

**Пример 3.4.5.** Если исключение Гаусса с полным выбором ведущего элемента применяется при решении задачи

$$\begin{bmatrix} 0.001 & 1.00 \\ 1.00 & 2.00 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.00 \\ 3.00 \end{bmatrix}$$

при  $\beta = 10$ ,  $t = 3$  в условиях приближенной арифметики, то

$$P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \Pi = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \hat{L} = \begin{bmatrix} 1.00 & 0.00 \\ 0.500 & 1.00 \end{bmatrix}, \quad \hat{U} = \begin{bmatrix} 2.00 & 1.00 \\ 0.00 & 0.499 \end{bmatrix}$$

и  $\hat{x} = (1.00, 1.00)$ . Это решение верно до двух значащих цифр в каждой компоненте. Ср. с примером 3.4.3.

### 3.4.10. Отказ от выбора ведущего элемента

Для некоторых классов матриц нет необходимости производить выбор ведущего элемента. Выявить такие классы важно, поскольку выбор ведущего элемента усложняет процесс. Чтобы проиллюстрировать, какого рода анализ требуется для доказательства безопасного отказа от выбора ведущего элемента, рассмотрим случай с диагонально доминирующими матрицами. Будем говорить, что матрица  $A \in \mathbb{R}^{n \times n}$  является *строго диагонально доминирующей*, если

$$|a_{ij}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|, \quad i = 1:n.$$

Следующая теорема показывает, как это свойство может прекрасно гарантировать выполнение LU-разложения без выбора ведущего элемента.

**Теорема 3.4.3.** Если матрица  $A^T$  является диагонально доминирующей, тогда LU-разложение матрицы  $A$  существует,  $|l_{ij}| < 1$ . Другими словами, если применять алгоритм 3.4.1, то  $P = I$ .

*Доказательство.* Разобьем матрицу  $A$  следующим образом

$$A = \begin{bmatrix} a & w^T \\ v & C \end{bmatrix},$$

где  $a$  порядка  $1 \times 1$ , и заметим, что после одного шага LU-процесса с внешним произведением мы получим разложение вида

$$\begin{bmatrix} a & w^T \\ v & C \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C - vw^T/a \end{bmatrix} \begin{bmatrix} a & w^T \\ 0 & I \end{bmatrix}.$$

Доказательство теоремы будет получено индукцией по  $n$ , если мы сможем показать, что матрица, транспонированная к  $B = C - vw^T/a$ , является строго диагонально доминирующей, поскольку из разложения  $B = L_1 U_1$  следует, что

$$A = \begin{bmatrix} 1 & 0 \\ v/a & L_1 \end{bmatrix} \begin{bmatrix} a & w^T \\ 0 & U_1 \end{bmatrix} \equiv LU.$$

Но доказательство факта, что  $B^T$ -матрица с диагональным доминированием, получается без всяких осложнений. Из определения мы имеем

$$\begin{aligned} \sum_{\substack{i=1 \\ i \neq j}}^{n-1} |b_{ij}| &= \sum_{\substack{i=1 \\ i \neq j}}^{n-1} |c_{ij} - v_i w_j/a| \leq \sum_{\substack{i=1 \\ i \neq j}}^{n-1} |c_{ij}| + \frac{|w_j|}{|a|} \sum_{\substack{i=1 \\ i \neq j}}^{n-1} |v_i| \leq \\ &\leq (|c_{jj}| - |w_j|) + \frac{|w_j|}{|a|} (|a| - |v_j|) \leq \left| c_{jj} - \frac{w_j v_j}{a} \right| = |b_{jj}|. \quad \square \end{aligned}$$

### 3.4.11. Некоторые приложения

Мы сделаем выводы и приведем примеры, иллюстрирующие, как нужно рассуждать на языке матричных разложений, когда сталкиваешься с различными видами линейных уравнений.

Пусть  $A$  – невырожденная  $n \times n$  матрица, а  $B$  – матрица  $n \times p$ . Рассмотрим задачу нахождения матрицы  $X (n \times p)$ , такой, что  $AX = B$ , т. е. задачу со многими правыми частями. Если имеются столбцовые разбиения матриц  $X = [x_1, \dots, x_p]$  и  $B = [b_1, \dots, b_p]$ , тогда имеем алгоритм

Вычислить  $PA = LU$ .

**for**  $k = 1:p$

Решить  $Ly = Pb_k$

Решить  $Ux_k = y$

**end**

(3.4.9)

Заметим, что разложение матрицы  $A$  выполняется однажды. Если  $B = I_n$ , то у нас возникает задача вычисления  $A^{-1}$ .

Другой пример применения  $LU$ -разложения является «нетрадиционным»; предположим, мы хотим решить линейную систему  $A^k x = b$ , где  $A \in \mathbb{R}^{n \times n}$ ,  $b \in \mathbb{R}^n$  и  $k$  – целое положительное число. Один подход состоит в вычислении матрицы  $C = A^k$  и далее в решении системы  $Cx = b$ . Он требует  $O(kn^3)$  флопов. Намного более эффективный алгоритм следующий:

Вычислить  $PA = LU$

**for**  $j = 1:k$

Вектор  $b$  замещается решением  $Ly = Pb$

Вектор  $b$  замещается решением  $Ux = b$

**end**

(3.4.10)

В качестве последнего примера мы покажем, как избегать ловушки, связанный с явным вычислением обратной матрицы. Пусть заданы матрица  $A \in \mathbb{R}^{n \times n}$  и векторы  $d \in \mathbb{R}^n$  и  $c \in \mathbb{R}^n$ , мы хотим вычислить  $s = c^T A^{-1} d$ . Один подход состоит в вычислении матрицы  $X = A^{-1}$ , как предлагается в задаче, и потом в нахождении  $s = c^T X d$ . Более экономная процедура состоит в выполнении  $PA = LU$  и потом в решении треугольных систем  $Ly = Pd$  и  $Ux = y$ . Далее следует, что  $s = c^T x$ . Цель данного примера – подчеркнуть, что когда в формуле встречается матричное обращение, то мы должны рассуждать на языке решения систем, а не на языке формирования точной обратной матрицы.

#### Задачи

**3.4.1.** Пусть  $A = LU$  будет  $LU$ -разложением  $n \times n$ -матрицы  $A$ , причем  $|e_{ij}| \leq 1$ . Обозначим через  $a_i^T$  и  $u_i^T$   $i$ -е строки матриц  $A$  и  $U$  соответственно. Проверить уравнение

$$u_i^T = a_i^T - \sum_{j=1}^{i-1} l_{ij} u_j^T$$

и, используя его, показать, что  $\|U\|_\infty \leq 2^{n-1} \|A\|_\infty$ .

(Подсказка: возьмите нормы и используйте индукцию.)

**3.4.2.** Показать, что если разложение  $PAQ = LU$  получается методом Гаусса с полным выбором, то не существует элемента в  $U(i, i:n)$  большего по абсолютной величине, чем  $|u_{ii}|$ .

**3.4.3.** Пусть для матрицы  $A \in \mathbb{R}^{n \times n}$  существует  $LU$ -разложение, и пусть  $L$  и  $U$  известны. Дать алгоритм вычисления  $(i, j)$  элемента матрицы  $A^{-1}$  сложности  $(n-j)^2 + (n-i)^2$  флопов.

**3.4.4.** Показать, что если  $\hat{X}$  – это обратная матрица, вычисленная посредством алгоритма (3.4.9), то  $A\hat{X} = I + F$ , где  $\|F\|_1 \leq \text{nu}(3 + 5n^2\rho) \|A\|_\infty \|\hat{X}\|_1$ .

**3.4.5.** Доказать теорему 3.4.2.

**3.4.6.** Обобщить алгоритм 3.4.3 так, чтобы с его помощью можно было разложить на множители произвольную прямоугольную матрицу.

**3.4.7.** Написать подробно вариант алгоритма блочного исключения, набросок которого дан в § 3.4.7.

### Замечания и литература к § 3.4

Фортран-программа для решения линейных систем общего вида может быть найдена в Linpack (Часть 1), а Алгол-версии в

Bowdler H. J., Martin R. S., Peters G. and Wilkinson J. H. (1966). "Solution of Real and Complex Systems of Linear Equations", Numer. Math. 8, 217–34. See also HACLA, pp. 93–110.

Предположение, что  $|a_{ij}^{(k)}| \leq n \max |a_{ij}|$ , когда используется полный выбор ведущего элемента, было доказано для вещественного случая при  $n = 4$  в работе

Cryer C. W. (1968). "Pivot Size in Gaussian Elimination", Numer. Math. 12, 335–45.

Другие работы, связанные с ростом и выбором ведущего элемента, включены в следующий список

Businger P. A. (1971). "Monitoring the Numerical Stability of Gaussian Elimination", Numer. Math. 16, 360–61.

Cohen A. M. (1974). "A Note on Pivot Size in Gaussian Elimination", Lin. Alg. and Its Applic. 8, 361–68.

Day J. and Peterson B. (1988). "Growth in Gaussian Elimination", Amer. Math. Monthly 95, 489–513.

Erisman A. M. and Reid J. K. (1974). "Monitoring the Stability of the Triangular Factorization of a Sparse Matrix", Numer. Math. 22, 183–86.

Higham N. J. and Higham D. J. (1988). "Large Growth Factors in Gaussian Elimination with Pivoting", Report 152, Department of Mathematics, University of Manchester, M13 9PL, England, to appear SIAM J. Matrix Analysis and Applications.

Reid J. K. (1971). "A Note on the Stability of Gaussian Elimination", J. Inst. Math. Applics. 8, 374–75.

Trefethen L. N. and Schreiber R. S. (1987). "Average-Case Stability of Gaussian Elimination", Numer. Anal. Report 88–3, Department of Mathematics, MIT. (To appear SIAM J. Matrix Anal. & Applic.)

Wilkinson J. H. (1961). "Error Analysis of Direct Methods of Matrix Inversion", J. ACM 8, 281–330.

Составление программ для разреженного исключения Гаусса – особенно интересная тема с точки зрения роста элементов, поскольку ради минимизации заполнения иногда допускаются множители больше единицы.

Duff I. S., Erisman A. M., and Reid J. K. (1986). Direct Methods for Sparse Matrices. Oxford University Press.

Связь между малыми ведущими элементами и близостью к вырожденности рассматривается в работе

Chan T. F. (1985). "On the Existence and Computation of LU Factorizations with small pivots", Math. Comp. 42, 535–548.

Стратегия выбора ведущего элемента, которую мы здесь не обсуждаем, это попарный выбор. В этом подходе для обнуления нижнего треугольника матрицы  $A$  используются  $2 \times 2$ -матрицы преобразования Гаусса. Эта техника представляет интерес при определенном многопроцессорном обеспечении, поскольку на каждом шаге взаимодействуют только соседние строки. См.

Sorensen D. (1985). "Analysis of Pairwise Pivoting in Gaussian Elimination", IEEE Trans. on Computers C-34, 274–278.

Как образец статьи, в которой описывается класс матриц, не требующий выбора ведущего элемента, см.

Serbin S. (1980). "On Factoring a Class of Complex Symmetric Matrices Without Pivoting", Math. Comp. 35, 1231–1234.

Точно так же, как существует шесть «условных» вариантов скалярного метода исключения

Гаусса, существует и шесть условных описаний блочного исключения Гаусса. Для ознакомления с этими процедурами и их реализациями см.

Dayde M. J. and Duff I. S. (1988). "Use of Level-3 BLAS in LU Factorization on the Cray-2, the ETA-10P, and the IBM 3090-200/VF", Report CSS-229, Computer Science and Systems Division, Harwell Laboratory, Oxon OX11 ORA, England.

Дополнительные практические и теоретические подробности о блочном LU-разложении могут быть найдены в

Calahan D. A. (1986). "Block-Oriented, Local-Memory-Based Linear Equation Solution on the Cray-2: Uniprocessor Algorithms", Proceedings of the 1986 Conference on Parallel Processing, pp. 375–378.

Gallivan K., Jalby W., Meier U. and Sameh A. H. (1988). "Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design", Int'l J. Supercomputer Applic. 2, 12–48.

## 3.5. Уточнение и оценивание точности

Пусть для решения  $n \times n$  системы  $Ax = b$  используется метод исключения Гаусса с частичным выбором ведущего элемента. Предположим, применяется  $t$ -разрядная плавающая арифметика с основанием  $\beta$ . Уравнение (3.4.4) гарантирует, что если фактор роста умеренный, то вычисленное решение  $\hat{x}$  удовлетворяет соотношению

$$(A + E)\hat{x} = b, \quad \|E\|_\infty \approx u\|A\|_\infty, \quad u = \beta^{-t}. \quad (3.5.1)$$

В этом разделе мы исследуем практические разветвления этого результата. Сначала сделаем акцент на различий, которое следует делать между величиной невязки и точностью. Потом продолжим обсуждение масштабирования, итерационного уточнения и оценки обусловленности.

Прежде чем начать, мы сделаем два замечания по поводу обозначений. Всюду используется бесконечная норма, так как это удобно для анализа ошибок округления и для практической оценки ошибки. Кроме того, где бы в этом разделе мы ни употребили выражение «исключение Гаусса», мы всегда будем иметь в виду исключение Гаусса с некоторой устойчивой стратегией выбора ведущего элемента, такой, как частичный выбор.

### 3.5.1. Большая невязка дает плохую точность

Невязкой вычисленного решения  $\hat{x}$  для линейной системы  $Ax = b$  является вектор  $b - A\hat{x}$ . Маленькая невязка обозначает, что вектор  $A\hat{x}$  эффективно «приближает» правую часть  $b$ . Из соотношения (3.5.1) мы имеем  $\|b - A\hat{x}\|_\infty \approx u\|A\|_\infty\|\hat{x}\|_\infty$  и поэтому получаем

**Вывод 1.** Метод исключения Гаусса дает решение  $\hat{x}$  с малой относительной невязкой.

Из маленьких невязок не следует высокой точности. Сопоставляя (3.5.1) с теоремой 2.7.1, мы видим, что

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \approx ux_\infty(A). \quad (3.5.2)$$

Это подтверждает второй ведущий принцип.

**Вывод 2.** Если машинная точность и обусловленность удовлетворяют соотношениям  $u \approx 10^{-d}$  и  $u_\infty(A) \approx 10^q$ , тогда исключение Гаусса дает решение с  $d - q$  правильными десятичными знаками.

Для иллюстрации выводов 1 и 2 рассмотрим систему

$$\begin{bmatrix} 0.986 & 0.579 \\ 0.409 & 0.237 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.235 \\ 0.107 \end{bmatrix},$$

в которой  $\kappa_\infty(A) \approx 700$  и  $x = (2 - 3)^T$ . Вот что мы находим для различных вариантов машинной точности:

$\beta$	$t$	$\hat{x}_1$	$\hat{x}_2$	$\frac{\ \hat{x} - x\ _\infty}{\ x\ _\infty}$	$\frac{\ b - A\hat{x}\ _\infty}{\ A\ _\infty \ \hat{x}\ _\infty}$
10	3	2.11	-3.17	$5 \cdot 10^{-2}$	$2.0 \cdot 10^{-3}$
10	4	1.986	-2.975	$8 \cdot 10^{-3}$	$1.5 \cdot 10^{-4}$
10	5	2.0019	-3.0032	$1 \cdot 10^{-3}$	$2.1 \cdot 10^{-6}$
10	6	2.00025	-3.00094	$3 \cdot 10^{-4}$	$4.2 \cdot 10^{-7}$

Можно ли признать вычисленное решение  $\hat{x}$  удовлетворительным – зависит от требований, заложенных в исходной задаче. Во многих приложениях точность не так важна, но важна малость невязки. В этом случае бывает достаточно вычислить  $\hat{x}$  при помощи исключения Гаусса. С другой стороны, если наличие правильных цифр в  $\hat{x}$  спорно, то ситуация является не такой простой, и в оставшейся части мы ее, безусловно, обсудим.

### 3.5.2. Масштабирование

Пусть  $\beta$  – это основание системы счисления; обозначим через  $D_1$  и  $D_2$  следующие диагональные матрицы

$$\begin{aligned} D_1 &= \text{diag}(\beta^{r_1} \dots \beta^{r_n}), \\ D_2 &= \text{diag}(\beta^{c_1} \dots \beta^{c_n}). \end{aligned}$$

Решение  $n \times n$  линейной системы  $Ax = b$  может быть найдено решением *масштабированной системы*  $(D_1^{-1}AD_2)y = D_1^{-1}b$  исключением Гаусса с учетом  $x = D_2y$ . Масштабирование матрицы  $A$  и векторов  $b$  и  $y$  требует только  $O(n^2)$  флопов и может быть выполнено без ошибок округления. Отметим, что матрицы масштабирования  $D_1$  и  $D_2$  неизвестны.

Из вывода 2 следует, что если  $\hat{x}$  и  $\hat{y}$  – это вычисленные аналоги векторов  $x$  и  $y$ , то

$$\frac{\|D_2^{-1}(\hat{x} - x)\|_\infty}{\|D_2^{-1}x\|_\infty} = \frac{\|\hat{y} - y\|_\infty}{\|y\|_\infty} = \kappa_\infty(D_1^{-1}AD_2). \quad (3.5.3)$$

Поэтому, если  $\kappa_\infty(D_1^{-1}AD_2)$  может быть сделано значительно меньшим, чем  $\kappa_\infty(A)$ , то можно ожидать соответственно более точного решения  $\hat{x}$ , при условии, что ошибки оцениваются в  $D_2$ -норме, которая определена следующим образом:  $\|z\|_{D_2} = \|D_2^{-1}z\|_\infty$ . Это и есть цель масштабирования. Заметим, что оно сопряжено с двумя вопросами: обусловленность масштабированной задачи и оценка для ошибки в  $D_2$ -норме.

Поиск точного минимума функции  $\kappa_p(D_1^{-1}AD_2)$  для произвольных диагональных матриц  $D_i$  и различных  $p$  – это интересная, но очень трудная математическая задача. Различные результаты, полученные в этом направлении, не пригодны для практики. Однако это не слишком обескураживает, когда мы вспоминаем, что оценка (3.5.3) является эвристической, и не имеет смысла искать точный минимум этой эвристиче-

ской оценки. То, что мы ищем,—это быстрый приближенный метод для улучшения качества вычисленного решения.

Среди множества подходов одним является простое *строчное масштабирование*. В этой схеме матрица  $D_2$  будет единичной, а матрица  $D_1$  выбирается так, что каждая строка в матрице  $D_1^{-1}A$  имеет приблизительно одинаковую  $\infty$ -норму. Строчное масштабирование уменьшает вероятность сложения очень маленького числа с очень большим числом в ходе исключения, что в результате значительно уменьшило бы точность.

Несколько более сложным, чем простое строчное масштабирование, является *строчно-столбцовое уравновешивание*. В данном случае наша цель — выбрать матрицы  $D_1$  и  $D_2$  так, чтобы  $\infty$ -норма каждой строки и столбца матрицы  $D_1^{-1}AD_2$  принадлежала интервалу  $[1/\beta, 1]$ , где  $\beta$  — это основание плавающей системы счисления. Среди работ в этом направлении смотри McKeeman (1962).

Не будет излишним подчеркнуть, что ни простое строчное масштабирование, ни строчно-столбцовое уравновешивание не могут «решить» задачу масштабирования. Действительно, каждый из этих методов может дать худший вектор  $\hat{x}$ , чем какой-либо другой, не использующий масштабирование. Развитие этого взгляда подробно обсуждается в работе Forsythe and Moler (SLE, гл. 11). Основная рекомендация состоит в том, что масштабирование уравнений и неизвестных должно производиться на уровне исходной задачи. Обычная стратегия масштабирования ненадежна. Лучше масштабировать (если вообще масштабировать) базис, по которому исходная задача задает значение каждого элемента  $a_{ij}$ . При этом можно принять во внимание единицы измерения и ошибки данных.

**Пример 3.5.1** (Forsythe and Moler (SLE стр. 34, 40)). Если система

$$\begin{bmatrix} 10 & 100000 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 100000 \\ 2 \end{bmatrix}$$

и эквивалентная строчно-масштабированная задача

$$\begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

решаются в арифметике с основанием  $\beta = 10$  и  $t = 3$ , тогда вычисляются, соответственно, решения  $\hat{x} = (0.00, 1.00)$  и  $\hat{x} = (1.00, 1.00)$ . Заметим, что точным решением является  $x = (1.0001 \dots, 0.9999 \dots)$ .

### 3.5.3. Итерационное уточнение

Пусть система  $Ax = b$  решена с использованием разложения с частичным выбором  $PA = LU$  и пусть мы хотим улучшить точность вычисленного решения  $\hat{x}$ . Если мы выполним алгоритм

$$\begin{aligned} r &= b - A\hat{x} \\ \text{Решить } Ly &= Pr, \\ \text{Решить } Uz &= y, \\ x_{new} &= \hat{x} + z, \end{aligned} \tag{3.5.4}$$

то в точной арифметике получим  $Ax_{new} = A\hat{x} + Az = (b - r) + r = b$ . К несчастью, непосредственное выполнение этих формул в плавающей арифметике приводит к вектору  $x_{new}$ , который не более точен, чем  $\hat{x}$ . Этого можно ожидать, так как  $\hat{r} = f1(b - A\hat{x})$  имеет мало, если вообще имеет, правильных значащих цифр. (Вспомним Вывод 1.) Следовательно,  $\hat{z} = f1(A^{-1}r) \approx A^{-1}$ . «Погрешность  $\approx$  погрешность»

является очень плохой поправкой с точки зрения улучшения точности  $\hat{x}$ . Тем не менее Skeel (1980) провел анализ ошибок, который показывает, когда алгоритм (3.5.4) дает улучшенное решение  $x_{new}$  с точки зрения обратного анализа ошибок. В частности, если величина

$$r = (\| |A| |A^{-1}| \|_\infty) \left( \max_i (|A| |x|)_i / \min_i (|A| |x|)_i \right)$$

не слишком велика, то алгоритм (3.5.4) вычисляет вектор  $x_{new}$ , такой, что  $(A + E)x_{new} = b$  при очень малой матрице  $E$ . Конечно, если использовать исключение Гаусса с частичным выбором, то вычисленное решение  $\hat{x}$  уже разрешает систему, близкую к исходной. Однако этого может не быть в случае применения стратегий выбора, которые используются для сохранения разреженности. В этом случае шаг итерационного уточнения с фиксированной точностью (3.5.4) может быть очень дешевым и оправданным. См. Arioly, Demmel и Duff (1988).

Для того чтобы алгоритм (3.5.4) получил более точный вектор  $\hat{x}$ , необходимо вычислить невязку  $b - A\hat{x}$  в плавающей арифметике с повышенной точностью. Обычно это означает, что если для вычисления разложения  $PA = LU$  и векторов  $x$ ,  $y$  использовалась  $t$ -разрядная арифметика, то для вычисления невязки  $b - A\hat{x}$  используется  $2t$ -разрядная арифметика, т. е. двойная точность. Процесс может быть итерационным. В частности, если мы вычислим разложение  $PA = LU$  и положим начальный вектор  $x = 0$ , то мы повторяем следующий процесс:

$$\begin{aligned} r &= b - Ax \text{ (Двойная точность)} \\ \text{Решить } Ly &= Pr \text{ относительно } y. \\ \text{Решить } Uz &= y \text{ относительно } z. \\ x &= x + z. \end{aligned} \tag{3.5.5}$$

Мы назовем этот процесс *итерационным уточнением со смешанной точностью*. Исходная матрица  $A$  должна быть использована в вычислениях вектора  $r$  с двойной точностью. Основной результат работы алгоритма (3.5.5) содержится в следующем выводе.

**Вывод 3.** Если машинная точность  $u$  и обусловленность удовлетворяют соотношениям  $u = 10^{-4}$  и  $\kappa_\infty(A) \approx 10^6$ , то через  $k$  выполнений алгоритма (3.5.5) вектор  $x$  имеет приблизительно  $\min(d, k(d - q))$  правильных знаков.

Грубо говоря, если  $\kappa_\infty(A) \leq 1$ , то с помощью итерационного уточнения можно в конечном счете получить решение, которое является предельно точным (для простой точности). Заметим, что этот процесс относительно дешевый. Каждое уточнение стоит  $O(n^2)$  флопов, для сравнения: исходный вклад разложения  $PA = LU$  составляет  $O(n^3)$  флопов. Разумеется, результат не может быть улучшен, если матрица  $A$  достаточно плохо обусловлена относительно точности ЭВМ.

Основной недостаток итерационного уточнения со смешанной точностью состоит в том, что его выполнение в некоторой степени является машинно-зависимым. Это осложняет его использование в программном обеспечении, предназначенном для широкого распространения. Необходимость хранить исходную копию матрицы  $A$  является вторым недостатком, связанным с этим методом.

С другой стороны, итерационное уточнение со смешанной точностью обычно очень легко реализовать на данной ЭВМ, которая имеет возможность накапливать скалярные произведения, т. е. возможность для вычисления скалярных произведений для строк матрицы  $A$  и вектора  $x$  с двойной точностью. В условиях вычислений с короткой мантиссой наличие программы итерационного уточнения может существенно расширить класс разрешимых задач  $Ax = b$ .

**Пример 3.5.2.** Если алгоритм (3.5.5) применяется к системе

$$\begin{bmatrix} 0.986 & 0.579 \\ 0.409 & 0.237 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.235 \\ 0.107 \end{bmatrix},$$

причем  $\beta = 10$  и  $t = 3$ , то итерационное уточнение дает следующую последовательность вычисленных решений

$$\hat{x} = \begin{bmatrix} 2.11 \\ -3.17 \end{bmatrix}, \quad \begin{bmatrix} 1.99 \\ -2.99 \end{bmatrix}, \quad \begin{bmatrix} 2.00 \\ -3.00 \end{bmatrix}, \dots$$

Точное решение:  $x = (2, -3)$ .

**Пример 3.5.3.** Если алгоритм 3.5.1 применяется к матрице  $B_n$ , заданной соотношением (2.7.9), то  $y = (2^{n-1}, 2^{n-2}, \dots, 2, 1)^T$  и  $\|y\|_\infty = \|B_n^{-1}\|_\infty$ .

### 3.5.4. Оценка обусловленности

Предположим, что система  $Ax = b$  решена посредством разложения  $PA = LU$  и мы хотим установить количество правильных цифр в вычисленном решении  $\hat{x}$ . Как следует из вывода 2, чтобы это сделать, нам надо оценить обусловленность  $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$ . Вычисление  $\|A\|_\infty$  не вызывает проблем, когда мы просто используем формулу

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

Сложной задачей является только вычисление множителя  $\|A^{-1}\|_\infty$ . Понятно, что мы можем оценить эту величину при помощи  $\|\hat{x}\|_\infty$ , где  $\hat{X} = [\hat{x}_1, \dots, \hat{x}_n]$  и  $\hat{x}_i$  является вычисленным решением системы  $Ax_i = e_i$ . (См. § 3.4.9.) Сложности применения этого подхода вызваны его стоимостью: вычисление  $\hat{x}_\infty = \|A\|_\infty \|\hat{X}\|_\infty$  стоит почти в три раза дороже, чем вычисление  $\hat{x}$ .

Главная проблема оценки обусловленности состоит в том, чтобы оценить обусловленность за  $O(n^2)$  флопов, допуская наличие разложения  $PA = LU$  или других разложений, представленных в предыдущих главах. Подход, описанный в работе Forsythe и Moler (SLE, стр. 51), основан на итерационном уточнении и формуле  $\kappa_\infty(A) \approx \|z\|_\infty / \|x\|_\infty$ , где вектор  $z$  является первой поправкой к вектору  $x$  в алгоритме (3.5.5). Общее время такого способа оценки обусловленности есть  $O(n^2)$ , что получается из-за ошибок итерационного уточнения, т. е. машинно-зависимой величины.

Cline, Moler, Stewart и Wilkinson (1979) предложили очень удачный подход к задаче оценки обусловленности без этого недостатка. Он основан на использовании соотношения

$$Ay = d \Rightarrow \|A^{-1}\|_\infty \geq \|y\|_\infty / \|d\|_\infty.$$

Идея их способа оценки состоит в выборе вектора  $d$  таким, чтобы решение  $y$  являлось большим по норме, и потом положить

$$\hat{x}_\infty = \|A\|_\infty \|y\|_\infty / \|d\|_\infty.$$

Успех этого метода зависит от того, как близко отношение  $\|y\|_\infty / \|d\|_\infty$  к своему максимальному значению  $\|A^{-1}\|_\infty$ .

Рассмотрим случай, когда матрица  $A = T$  является верхней треугольной. Отношение между векторами  $d$  и  $y$  полностью описывается следующей столбцовой

версией алгоритма обратной подстановки

```

 $p(1:n) = 0$ 
for  $k = n:-1:1$ 
    Выбрать  $d(k)$ 
     $y(k) = (d(k) - p(k))/T(k, k)$ 
     $p(1:k-1) = p(1:k-1) + y(k) T(1:k-1, k)$ 
end

```

(3.5.6)

Обычно этот алгоритм используется для решения *заданной* треугольной системы  $Ty = d$ . Однако сейчас мы свободны выбирать правую часть  $d$ , исходя из того «ограничения», что вектор  $y$  велик относительно  $d$ .

Один способ поддержать рост в векторе  $y$  – это выбирать элементы  $d(k)$  из множества  $\{-1, +1\}$  так, чтобы максимизировать  $y(k)$ . Если  $p(k) \geq 0$ , то положим  $d(k) = -1$ . Если  $p(k) < 0$ , то положим  $d(k) = +1$ . Другими словами, алгоритм (3.5.6) работает с элементами  $d(k) = -\text{sign}(p(k))$ . Так как  $d$  является вектором вида  $d(1:n) = (\pm 1, \dots, \pm 1)^T$ , мы получаем оценку  $\hat{\kappa}_\infty = \|T\|_\infty \|y\|_\infty$ .

Более надежный способ оценки получается, если выбирать  $d(k) \in \{-1, +1\}$  так, чтобы поддержать рост сразу и в  $y(k)$ , и в изменении текущей суммы, заданной соотношением  $p(1:k-1, k) + T(1:k-1, k)y(k)$ . В частности, на  $k$ -м шаге мы вычисляем

$$\begin{aligned} y(k)^+ &= (1 - p(k))/T(k, k), \\ s(k)^+ &= |y(k)^+| + \|p(1:k-1) + T(1:k-1, k)y(k)^+\|_1, \\ y(k)^- &= (-1 - p(k))/T(k, k), \\ s(k)^- &= |y(k)^-| + \|p(1:k-1) + T(1:k-1, k)y(k)^-\|_1. \end{aligned}$$

Потом полагаем  $y(k)$  равным  $y(k)^+$ , если  $s(k)^+ \geq s(k)^-$ , и  $y(k)^-$  в противном случае. Это дает

**Алгоритм 3.5.1 (Метод оценки обусловленности).** Пусть  $T \in \mathbf{R}^{n \times n}$  является невырожденной верхней треугольной матрицей. Данный алгоритм вычисляет  $\infty$ -норму вектора  $y$  и скаляр  $\kappa$ , такие, что  $\|Ty\|_\infty \approx 1/\|T^{-1}\|_\infty$  и  $\kappa \approx \kappa_\infty(T)$

```

 $p(1:n) = 0$ 
for  $k = n:-1:1$ 
     $y(k)^+ = (1 - p(k))/T(k, k)$ 
     $y(k)^- = (-1 - p(k))/T(k, k)$ 
     $p(k)^+ = p(1:k-1) + T(1:k-1, k)y(k)^+$ 
     $p(k)^- = p(1:k-1) + T(1:k-1, k)y(k)^-$ 
    if  $|y(k)^+| + \|p(k)^+\|_1 \geq |y(k)^-| + \|p(k)^-\|_1$ 
         $y(k) = y(k)^+; p(1:k-1) = p(k)^+$ 
    else
         $y(k) = y(k)^-; p(1:k-1) = p(k)^-$ 
    end
end
 $\kappa = \|y\|_\infty \|T\|_\infty$ 

```

Алгоритм неоднократно использует обычную обратную подстановку.

Сейчас мы в состоянии описать процедуру оценки обусловленности квадратной невырожденной матрицы  $A$ , для которой известно разложение  $PA = LU$ :

- Применим нижнюю треугольную версию алгоритма 3.5.1 к матрице  $U^T$  и получим большое по норме решение системы  $U_y^T = d$ .
- Решим треугольные системы  $L^T r = y$ ,  $Lw = Pr$  и  $Uz = w$ .
- $\hat{\kappa}_\infty = \|A\|_\infty \|z\|_\infty / \|r\|_\infty$ .

Отметим, что  $\|z\|_\infty \leq \|A^{-1}\|_\infty \|r\|_\infty$ . Метод основан на нескольких эвристических соображениях. Первое, если матрица  $A$  является плохо обусловленной и  $PA = LU$ , то в этом случае обычно и матрица  $U$  является соответственно плохо обусловленной. Нижний треугольник  $L$  достаточно хорошо обусловлен. Таким образом, более целесообразно оценить обусловленность матрицы  $U$ , чем матрицы  $L$ . Вектор  $r$ , поскольку он разрешает систему  $A^T P^T r = d$ , сильно смещен в направлении левых сингулярных векторов, соответствующих  $\sigma_{\min}(A)$ . Правые части, обладающие таким свойством, приводят к большим решениям задачи  $Az = r$ .

Практика показывает, что метод оценки обусловленности, который мы описали, дает хорошие приближенные оценки реальных чисел обусловленности. См. Linpack.

### Задачи

**3.5.1.** Показать на примере, что существует более одного способа для уравновешивания матрицы.

**3.5.2.** Применяя арифметику с параметрами  $\beta = 10$ ,  $t = 2$ , решить систему

$$\begin{bmatrix} 11 & 15 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \end{bmatrix}$$

при помощи исключения Гаусса с частичным выбором. Выполнить один шаг итерационного уточнения, применяя арифметику при  $t = 4$ , вычислить невязку. (Не забудьте округлить вычисленную невязку до двух знаков.)

**3.5.3.** Пусть  $P(A + E) = \hat{L} \hat{U}$ , где  $P$  является перестановочной матрицей,  $\hat{L}$  – нижней треугольной с элементами  $|\hat{l}_{ij}| \leq 1$  и  $\hat{U}$  – верхней треугольной. Показать, что

$$\hat{\kappa}_\infty(A) \geq \frac{\|A\|_\infty}{\|E\|_\infty + \mu},$$

где  $\mu = \min |\hat{u}_{ii}|$ . Сделайте вывод, что если при применении исключения Гаусса с частичным выбором в матрице  $A$  встречается малый ведущий элемент, то матрица  $A$  является плохо обусловленной. Обратное неверно. (Например  $A = B_n$ .)

**3.5.4.** (Kahan 1966). Система  $Ax = b$ , где

$$A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 10^{-10} & 10^{-10} \\ 1 & 10^{-10} & 10^{-10} \end{bmatrix}, \quad b = \begin{bmatrix} 2(1 + 10^{-10}) \\ -10^{-10} \\ 10^{-10} \end{bmatrix},$$

имеет решение  $x = (10^{-10}, -1, 1)$ . (а) Показать, что если  $(A + E)y = b$  и  $|E| \leq 10^{-8}|A|$ , то  $|x - y| \leq 10^{-7}|x|$ . То есть относительно малые изменения в элементах матрицы  $A$  не приводят к большим изменениям в векторе  $x$ , хотя  $\kappa_\infty(A) = 10^{10}$ . (б) Обозначим  $D = \text{diag}(10^{-5}, 10^{-5}, 10^{-5})$ . Показать, что  $\kappa_\infty(DAD) \leq 5$ . (с) Объяснить, что это означает в терминах теоремы 2.7.3.

**3.5.5.** Рассмотрите матрицу:

$$T = \begin{bmatrix} 1 & 0 & M & -M \\ 0 & 1 & -M & M \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad M \in \mathbb{R}.$$

Какая оценка числа  $\kappa_\infty(T)$  будет получена, если применяется алгоритм (3.5.6) с элементами  $d(k) = -\text{sign}(p(k))$ ? Какую оценку дает алгоритм 3.5.1? Каково действительное число  $\kappa_\infty(T)$ ?

### Замечания и литература к § 3.5

Следующие работы имеют отношение к задачам масштабирования системы  $Ax = b$

- Bauer F. L. (1963). "Optimally Scaled Matrices", *Numer. Math.*, 5, 73–87.  
 Businger P. A. (1968). "Matrices Which Can be Optimally Scaled", *Numer. Math.* 12, 346–48.  
 Fenner T. and Loizou G. (1974). "Some New Bounds on the Condition Numbers of Optimally Scaled Matrices", *J. ACM* 21, 514–24.  
 Golub G. H. and Varah J. M. (1974). "On a Characterization of the Best  $L_2$ -Scaling of a Matrix", *SIAM J. Num.Anal.* 11, 472–79.  
 McCarthy C. and Strang G. (1973). "Optimal Conditioning of Matrices", *SIAM J. Num. Anal.* 10, 370–88.  
 Skeel R. (1979). "Scaling for Numerical Stability in Gaussian Elimination", *J. ACM* 26, 494–526.  
 Skeel R. (1981). "Effect of Equilibration on Residual Size for Partial Pivoting", *SIAM J. Num. Anal.* 18, 449–55.  
 van der Sluis A. (1969). "Condition Numbers and Equilibration Matrices", *Numer. Math.* 14, 14–23.  
 van der Sluis A. (1970). "Condition, Equilibration, and Pivoting in Linear Algebraic Systems", *Numer. Math.* 15, 74–86.

Часть трудностей масштабирования связана с выбором нормы для измерения ошибок. Интересное обсуждение этого, часто не замечаемого вопроса, приводится в работе

- Kahan W. (1966). "Numerical Linear Algebra", *Canadian Math. Bull.* 9, 757–801.

Для изучения строгого анализа итерационного уточнения см.

- Jankowski M. and Wozniakowski M. (1977). "Iterative Refinement Implies Numerical Stability", *BII* 17, 303–311.  
 Moler C. B. (1967). "Iterative Refinement in Floating Point", *J. ACM* 14, 316–71.  
 Skeel R. D. (1980). "Iterative Refinement Implies Numerical Stability for Gaussian Elimination", *Math. Comp.* 35, 817–832.  
 Stewart G. W. (1981). "In the Implicit Deflation of Nearly Singular Systems of Linear Equations", *SIAM J. Sci. and Stat. Comp.* 2, 136–140.

Метод оценки обусловленности, который мы описали, дается в работе

- Cline A. K., Moler C. B., Stewart G. W., and Wilkinson J. H. (1979). "An Estimate for the Condition Number of a Matrix", *SIAM J. Num. Anal.* 16, 368–75.

и включен в Linpack (пп. 1.10–1.13). Другие ссылки связанные с задачей оценки обусловленности содержатся в работах

- Broyden C. G. (1973). "Some Condition Number Bounds for the Gaussian Elimination Process", *J. Inst. Math. Applic.* 12, 273–86.  
 Cline A. K., Conn A. R., and Van Loan C. (1982). "Generalizing the LINPACK Condition Estimator", in *Numerical Analysis*, ed., J. P. Hennart, Lecture Notes in Mathematics no. 909, Springer-Verlag, New York.  
 Cline A. K. and Rew R. K. (1983). "A Set of Counter examples to Three Condition Number Estimators", *SIAM J. Sci. and Stat. Comp.* 4, 602–611.  
 Grimes R. G. and Lewis J. G. (1981). "Condition Number Estimation for Sparse Matrices", *SIAM J. Sci. and Stat. Comp.* 2, 384–88.  
 Hager W. (1984). "Condition Estimates", *SIAM J. Sci. and Stat. Comp.* 5, 311–316.  
 Higham N. J. (1987). "A Survey of Condition Number Estimation for Triangular Matrices", *SIAM Review* 29, 575–596.  
 Higham N. J. (1988). "Fortran Codes for Estimating the One-norm of a Real or Complex Matrix with Applications to Condition Estimation", *ACM Trans. Math. Soft.* 14, 381–396.  
 Lemeire F. (1973). "Bounds for Condition Numbers of Triangular Value of a Matrix", *Lin. Alg. and Its Applic.* 11, 1–2.  
 O'Leary D. P. (1980). "Estimating Matrix Condition Numbers", *SIAM J. Sci. Stat. Comp.* 1, 205–9.  
 Stewart G. W. (1980). "The Efficient Generation of Random Orthogonal Matrices with an Application to Condition Estimators", *SIAM J. Num. Anal.* 17, 403–9.  
 Van Loan C. (1987). "On Estimating the Condition of Eigenvalues and Eigenvectors", *Lin. Alg. and Its Applic.* 88/89, 715–732.  
 Varga R. S. (1976). "On Diagonal Dominance Arguments for Bounding  $\|A^{-1}\|_\infty$ ", *Lin. Alg. and Its Applic.* 14, 211–17.

## Глава 4

# Линейные системы специального вида

Основной принцип численного анализа состоит в том, что при решении задач всякий раз надо использовать их специфику. Можно надеяться, что в численных методах линейной алгебры алгоритмы решения задач с матрицами общего вида приспособлены к использованию таких свойств как симметричность, определенность, разреженность. Это и будет центральной темой данной главы, в которой наша главная цель – предложить специальные алгоритмы для вычисления специальных вариантов  $LU$ -разложения.

Мы начнем с установления зависимости между множителями  $L$  и  $U$ , когда матрица  $A$  является симметричной. Это достигается исследованием  $LDM^T$ -разложения в § 4.1. Потом мы займемся важным случаем, когда матрица  $A$  является одновременно симметричной и положительно определенной, и получим в § 4.2 устойчивое разложение Холецкого. В этом разделе мы также исследуем несимметричные положительно определенные системы. В § 4.3 обсуждается ленточная варианта исключения Гаусса и другие методы разложения. Потом будет исследована интересная ситуация, когда  $A$  симметричная, но неопределенная. Наша трактовка этой задачи в § 4.4 выявляет двойственность стоящей перед исследователем проблемы выбора ведущего элемента. Мы предпочитаем выбирать ведущий элемент так, чтобы добиться устойчивости, и не обращаем внимания, что при этом может быть нарушена специфика. К счастью, в задаче с симметричной неопределенной матрицей имеется удачный способ разрешения этого конфликта.

Любые блочные ленточные матрицы являются также просто ленточными, и поэтому к ним могут применяться методы из § 4.3. Однако бывают ситуации, когда такой подход неприемлем. Для иллюстрации мы рассмотрим в § 4.5 случай блочной трехдиагональной системы.

В последних параграфах мы исследуем несколько очень интересных  $O(n^2)$  алгоритмов, которые могут быть использованы для решения систем Вандермонда и тёплицевых систем.

## 4.1. Разложения типа $LDM^T$ и $LDL^T$

Мы хотим создать метод, использующий специфику при решении задачи  $Ax = b$ . Для этого установим возможность  $LU$ -разложения матрицы  $A$  в виде произведения трех матриц  $LDM^T$ , где  $D$  – диагональная, а  $L$  и  $M$  – нижние унитреугольные. Когда такое разложение получено, решение  $Ax = b$  может быть найдено за  $O(n^2)$  флопов посредством решения систем  $Ly = b$  (прямая подстановка),  $Dz = y$  и  $M^Tx = z$  (обратная подстановка). Исследование  $LDM^T$ -разложения позволяет для симметричного случая установить следующий факт: если  $A = A^T$ , то  $L = M$  и работа, связанная с разложением, составляет половину от того, что требуется для исключения Гаусса. Проблема выбора ведущего элемента поднимается в последующих разделах.

### 4.1.1. $LDM^T$ -разложение

Наш первый результат связывает  $LDM^T$ -разложение и  $LU$ -разложение.

**Теорема 4.1.1.** Если все ведущие главные подматрицы  $A \in \mathbb{R}^{n \times n}$  невырождены, тогда существуют единственныи нижние унитреугольные матрицы  $L$  и  $M$ , и единственная диагональная матрица  $D = \text{diag}(d_1, \dots, d_n)$ , такие, что  $A = LDM^T$ .

**Доказательство.** Из теоремы 3.2.1 мы знаем, что для матрицы  $A$  существует  $LU$ -разложение  $A = LU$ . Пусть  $D = \text{diag}(d_1, \dots, d_n)$ , где  $d_i = u_{ii}$ , при  $i = 1:n$ . Заметим, что матрица  $D$  является невырожденной, а матрица  $M^T = D^{-1}U$  – верхней унитреугольной. Поэтому  $A = LU = LD(D^{-1}U) = LDM^T$ . Единственность следует из единственности  $LU$ -разложения, как было показано в теореме 3.2.1.  $\square$

Доказательство показывает, что  $LDM^T$ -разложение может быть найдено при помощи исключения Гаусса, вычисляющего  $A = LU$ , с последующим определением  $D$  и  $M$  из уравнения  $U = DM^T$ . Тем не менее может быть установлен интересный альтернативный алгоритм для непосредственного вычисления  $L$ ,  $D$  и  $M$ .

Допустим, что мы знаем первые  $j-1$  столбцов матрицы  $L$ , диагональные элементы  $d_1, \dots, d_{j-1}$  матрицы  $D$  и первые  $j-1$  строк матрицы  $M$  для некоторого  $j$ ,  $1 \leq j \leq n$ . Чтобы получить способ вычисления  $L(j+1:n, j)$ ,  $M(j, 1:j-1)$  и  $d_j$ , приравняем  $j$ -е столбцы в уравнении  $A = LDM^T$ . В частности,

$$A(1:n, j) = Lv, \quad (4.1.1)$$

где

$$v = DM^T e_j.$$

«Верхняя» половина в соотношении (4.1.1) определяет  $v(1:j)$  как решение известной нижней треугольной системы:

$$L(1:j, 1:j)v(1:j) = A(1:j, j).$$

Когда мы знаем  $v$ , мы вычисляем

$$\begin{aligned} d(j) &= v(j), \\ m(j, i) &= v(i)/v(j), \quad i = 1:j-1. \end{aligned}$$

«Нижняя» половина формулы (4.1.1) дает уравнение

$$L(j+1:n, 1:j)v(1:j) = A(j+1:n, j),$$

которое может быть преобразовано для получения способа вычисления  $j$ -го столбца матрицы  $L$ :

$$L(j+1:n, j)v(j) = A(j+1:n, j) - L(j+1:n, 1:j-1)v(1:j-1).$$

Таким образом, матрица  $L(j+1:n, j)$  задает скалярную операцию, и в целом мы получаем алгоритм

```

for  $j = 1:n$ 
    Решить  $L(1:j, 1:j)v(1:j) = A(1:j, j)$  для  $v(1:j)$ .
    for  $i = 1:j-1$ 
         $M(j, i) = v(i)/v(j)$ 
    end
     $d(j) = v(j)$ 
     $L(j+1:n, j) =$ 
         $(A(j+1:n, j) - L(j+1:n, 1:j-1)v(1:j-1))/v(j)$ 
end

```

(4.1.2)

Как и в случае  $LU$ -разложения, возможно замещение матрицы  $A$  на множители  $L$ ,  $D$  и  $M$ . Если для нахождения вектора  $v(1:j)$  используется столбцовая версия прямого исключения, то мы получаем следующую процедуру:

**Алгоритм 4.1.1. ( $LDM^T$ ).** Если для матрицы  $A \in \mathbb{R}^{n \times n}$  выполнено  $LU$ -разложение, то данный алгоритм вычисляет нижние унитреугольные матрицы  $L$  и  $M$  и диагональную матрицу  $D = \text{diag}(d_1, \dots, d_n)$ , такие, что  $A = LDM^T$ . Элемент  $a_{ij}$  замещается на  $l_{ij}$  при  $i > j$ , на  $d_i$ , если  $i = j$ , и на  $m_{ij}$ , если  $i < j$ .

```

for  $j = 1:n$ 
    {Решить  $L(1:j, 1:j)v(1:j) = A(1:j, j)$ .}
     $v(1:j) = A(1:j, j)$ 
    for  $k = 1:j - 1$ 
         $v(k + 1:j) = v(k + 1:j) - v(k)A(k + 1:j, k)$ 
    end
    {Вычислить  $M(j, 1:j - 1)$  и запомнить в  $A(1:j - 1, j)$ .}
    for  $i = 1:j - 1$ 
         $A(i, j) = v(i)/A(i, i)$ 
    end
    {Запомнить  $d(j)$  в  $A(j, j)$ .}
     $A(j, j) = v(j)$ 
    {Вычислить  $L(j + 1:n, j)$  и запомнить в  $A(j + 1:n, j)$ .}
    for  $k = 1:j - 1$ 
         $A(j + 1:n) = A(j + 1:n, j) - v(k)A(j + 1:n, k)$ 
    end
     $A(j + 1:n, j) = A(j + 1:n, j)/v(j)$ 
end

```

Алгоритм содержит такой же объем работ, как и  $LU$ -разложение, около  $2n^3/3$  флопов.

Можно показать, что вычисленное решение  $\hat{x}$  системы  $Ax = b$ , полученное посредством алгоритма 4.1.1 и методами решения треугольных систем из § 3.1, удовлетворяет возмущенной системе  $(A + E)\hat{x} = b$ , где

$$|E| \leq n\mathbf{u}(3|A| + 5|\hat{L}||\hat{D}||\hat{M}^T|) + O(\mathbf{u}^2) \quad (4.1.3)$$

и матрицы  $\hat{L}$ ,  $\hat{D}$  и  $\hat{M}$  – это вычисленные версии матриц  $L$ ,  $D$  и  $M$  соответственно.

Как и в случае  $LU$ -разложения, рассмотренного в предыдущей главе, верхняя оценка в формуле (4.1.3) дается без ограничений на выбор ведущего элемента. Следовательно, чтобы алгоритм 4.1.1 был практической процедурой, он должен быть модифицирован под вычисление разложения вида  $PA = LDM^T$ , где  $P$  – матрица перестановок, выбиравшаяся таким образом, что элементы матрицы  $L$  удовлетворяют условию  $|l_{ij}| \leq 1$ . Детали этой модификации мы здесь не приводим, так как они могут быть получены непосредственно, наша же главная цель введения  $LDM^T$ -разложения – это описать специальные методы для симметричных систем.

**Пример 4.1.1.**

$$A = \begin{bmatrix} 10 & 10 & 20 \\ 20 & 25 & 40 \\ 40 & 50 & 61 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

и в завершение алгоритм 4.1.1 замещает матрицу  $A$  следующий матрицей:

$$A = \begin{bmatrix} 10 & 1 & 2 \\ 2 & 5 & 0 \\ 3 & 4 & 1 \end{bmatrix}.$$

#### 4.1.2. Симметрия и $LDL^T$ -разложение

Если матрица  $A$  симметричная, то в  $LDM^T$ -разложении имеется избыточная информация.

**Теорема 4.1.2.** Если представление  $A = LDM^T$  есть  $LDM^T$ -разложение невырожденной симметричной матрицы  $A$ , то  $L = M$ .

**Доказательство.** Матрица  $M^{-1}AM^{-T} = M^{-1}LD$  является одновременно и симметричной, и нижней треугольной и поэтому диагональна. Так как  $D$  невырождена, то отсюда вытекает, что  $M^{-1}L$  тоже диагональна. Но  $M^{-1}L$  является нижней унитреугольной, поэтому  $M^{-1}L = I$ .  $\square$

Учитывая этот результат, можно наполовину уменьшить работу алгоритма 4.1.1, когда он применяется к симметричной матрице. На  $j$ -м шаге нам уже известна строка  $M(j, 1:j-1)$ , так как  $M = L$  и мы предполагаем знание первых  $j-1$  столбцов матрицы  $L$ . Вспомним, что на  $j$ -м шаге алгоритма (4.1.2) вектор  $v$  определяется уравнением  $v = DM^Te_j$ . Так как  $M = L$ , отсюда видно, что

$$v = \begin{bmatrix} d(1)L(j, 1) \\ \vdots \\ d(j-1)L(j, j-1) \\ d(j) \end{bmatrix}.$$

Следовательно, вектор  $v(1:j-1)$  может быть получен простым масштабированием  $j$ -й строки матрицы  $L$ . Формула для  $j$ -й компоненты вектора  $v$  получается из  $j$ -го уравнения системы  $L(1:j, 1:j)v = A(1:j, j)$

$$v(j) = A(j, j) - L(j, 1:j-1)v(1:j-1),$$

и поэтому мы имеем

```

for j = 1:n
    for i = 1:j-1
        v(i) = L(j, i)d(i)
    end
    v(j) = A(j, j) - L(j, 1:j-1)v(1:j-1)
    d(j) = v(j)
    L(j+1:n, j) =
        (A(j+1:n, j) - L(j+1:n, 1:j-1)v(1:j-1))/v(j)
end

```

С учетом замещения получаем

**Алгоритм 4.1.2 ( $LDL^T$ ).** Если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и для нее существует  $LU$ -разложение, то данный алгоритм вычисляет нижнюю унитреугольную матрицу  $L$  и диагональную матрицу  $D = \text{diag}(d_1, \dots, d_n)$ , такие, что  $A = LDL^T$ . Элементы  $a_{ij}$  замещаются на  $l_{ij}$ , если  $i > j$ , и на  $d_i$ , если  $i = j$ .

```

for  $j = 1 : n$ 
    {Вычислить  $v(1:j)$ .}
    for  $i = 1:j - 1$ 
         $v(i) = A(j, i) A(i, i)$ 
    end
     $v(j) = A(j, j) - L(j, 1:j - 1) v(1:j - 1)$ 
    {Запомнить  $d(j)$  и вычислить  $L(j + 1:n, j)$ .}
     $A(j, j) = v(j)$ 
     $A(j + 1:n, j) =$ 
         $(A(j + 1:n, j) - A(j + 1:n, 1:j - 1) v(1:j - 1)) / v(j)$ 
end

```

Этот алгоритм требует  $n^3/3$  флопов, около половины количества флопов, требуемых для исключения Гаусса.

В следующем разделе мы покажем, что если матрица  $A$  одновременно и симметрична, и положительно определенная, то алгоритм 4.1.2 не только работает, но также чрезвычайно устойчив. Если матрица  $A$  симметричная, но не является положительно определенной, то, возможно, есть необходимость применять выбор ведущего элемента и использовать методы из § 4.4.

#### Пример 4.1.2.

$$A = \begin{bmatrix} 10 & 20 & 30 \\ 20 & 45 & 80 \\ 30 & 80 & 171 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix},$$

и, если применять алгоритм 4.1.2, матрица  $A$  замещается матрицей

$$A = \begin{bmatrix} 10 & 20 & 30 \\ 2 & 5 & 80 \\ 3 & 4 & 1 \end{bmatrix}.$$

#### Задачи

**4.1.1.** Показать, что если  $LDM^T$ -разложение невырожденной матрицы существует, то оно единственno.

**4.1.2.** Модифицировать алгоритм 4.1.1 так, чтобы он вычислял разложение вида  $PA = LDM^T$ , где  $L$  и  $M$ -нижние унитреугольные матрицы,  $D$ -диагональная матрица, а  $P$ -матрица перестановок, выбираемая таким образом, что  $|l_{ij}| \leq 1$ .

**4.1.3.** Пусть симметричная  $n \times n$ -матрица  $A = (a_{ij})$  хранится в векторе  $c$  следующим образом:  $c = (a_{11}, a_{21}, \dots, a_{n1}, a_{22}, \dots, a_{n2}, \dots, a_{nn})$ . Переписать алгоритм 4.1.2 для матрицы  $A$ , хранимой таким способом. Упростить индексацию в пределах внутреннего цикла, насколько это возможно.

**4.1.4.** Переписать алгоритм 4.1.2 для матрицы  $A$ , хранимой по диагоналям. Смотри § 1.2.8.

#### Замечания и литература к § 4.1

Алгоритм 4.1.1 близок к методам Краута и Дулитла, в которых избегается модификация с внешним произведением. См. гл. 4 книги

Fox L. (1964). An Introduction to Numerical Linear Algebra, Oxford University Press, Oxford.

и также Stewart (IMC, стр. 131–149). Алгол-процедуру можно найти в работе

Bowdler H. J., Martin R. S., Peters G., and Wilkinson J. H. (1966). “Solution of Real and Complex Systems of Linear Equations”, Numer. Math. 8, 217–234. See also HACLA, pp. 93–110.

См. также

Forsythe G. E. (1960). "Crout with Pivoting", *Comm. ACM* 3, 507–508.

McKeeman W. M. (1962). "Crout with Equilibration and Iteration", *Comm. ACM*. 5, 553–555.

## 4.2. Положительно определенные системы

Матрица  $A \in \mathbf{R}^{n \times n}$  является *положительно определенной*, если  $x^T A x > 0$  для всех ненулевых векторов  $x \in \mathbf{R}^n$ . Положительно определенные системы составляют один из наиболее важных классов задач  $Ax = b$  со спецификой. Рассмотрим  $2 \times 2$ -симметричный случай. Если

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

является положительно определенной, то

$$x = (1, 0)^T \Rightarrow x^T A x = a_{11} > 0,$$

$$x = (0, 1)^T \Rightarrow x^T A x = a_{22} > 0,$$

$$x = (1, 1)^T \Rightarrow x^T A x = a_{11} + 2a_{12} + a_{22} > 0,$$

$$x = (1, -1)^T \Rightarrow x^T A x = a_{11} - 2a_{12} + a_{22} > 0.$$

Последние два уравнения приводят к неравенству  $|a_{12}| \leq (a_{11} + a_{22})/2$ . Из этих результатов мы видим, что наибольший элемент матрицы  $A$  лежит на диагонали и он положителен. Это верно и в общем случае. Положительно определенная матрица имеет «весомую» диагональ. Масса диагонали не столь заметна, как в случае с диагональным доминированием (ср. § 3.4.9), но она дает аналогичный эффект – устраняет необходимость выбора ведущего элемента.

Мы начнем с нескольких замечаний о свойстве положительной определенности и о том, что она влечет за собой в несимметричном случае при выборе ведущего элемента. Потом мы остановимся на эффективной реализации процедуры Холецкого, которая может быть использована для надежного разложения симметричной положительно определенной матрицы  $A$ . Исследуются гахру-версия, версия с внешним произведением и блочная версия. Раздел заканчивается несколькими замечаниями о неотрицательно определенном случае.

### 4.2.1. Положительная определенность

Пусть матрица  $A \in \mathbf{R}^{n \times n}$  является положительно определенной. Очевидно, что положительно определенная матрица невырождена, так как в противном случае нашелся бы ненулевой вектор  $x$ , такой, что  $x^T A x = 0$ . Однако, как показывают следующие результаты, из положительности *квадратичной формы*  $x^T A x$  вытекают гораздо более сильные свойства.

**Теорема 4.2.1.** *Если матрица  $A \in \mathbf{R}^{n \times n}$  является положительно определенной и матрица  $X \in \mathbf{R}^{n \times k}$  имеет ранг  $k$ , то матрица  $B = X^T A X \in \mathbf{R}^{k \times k}$  также положительно определена.*

*Доказательство.* Если вектор  $z \in \mathbf{R}^k$  удовлетворяет условию  $0 \geq z^T B z = (Xz)^T A (Xz)$ , то  $Xz = 0$ . Но так как  $X$  – матрица полного ранга, отсюда следует, что  $z = 0$ .  $\square$

**Следствие 4.2.2.** *Если  $A$  – положительно определенная матрица, то все ее главные подматрицы положительно определенные. В частности, все диагональные элементы положительны.*

*Доказательство.* Если  $v \in \mathbf{R}^k$  – это целочисленный вектор, причем  $1 \leq v_1 < \dots < v_k \leq$

$\leq n$ , то  $X = I_n(:, v)$  является матрицей ранга  $k$ , составленной из единичных столбцов с индексами  $v_1, \dots, v_k$ . Как следует из теоремы 4.2.1, матрица  $A(v, v) = X^TAX$  является положительно определенной.  $\square$

**Следствие 4.2.3.** Если матрица  $A$  положительно определенная, то разложение  $A = LDM^T$  существует и матрица  $D = \text{diag}(d_1, \dots, d_n)$  имеет положительные диагональные элементы.

**Доказательство.** Из следствия 4.2.2 следует, что подматрицы  $A(1:k, 1:k)$  невырожденны для  $k = 1:n$ , и поэтому согласно теореме 4.1.1 разложение  $A = LDM^T$  существует. Если мы применим теорему 4.2.1 при  $X = L^{-T}$ , то матрица  $B = D M^T L^{-T} = L^{-1} A L^{-T}$  является положительно определенной. Так как  $M^T L^{-T}$  – это верхняя унитреугольная матрица, а матрицы  $B$  и  $D$  имеют одинаковую диагональ, то эта диагональ должна быть положительна.  $\square$

Существует несколько типичных ситуаций, которые на практике вызывают положительную определенность матрицы:

- Квадратичная форма является энергетической функцией, положительность которой гарантирована ее физическим смыслом.
- Матрица  $A$  является крест-произведением  $X^T X$ , где  $X$  имеет полный столбцовый ранг. (Положительная определенность следует из теоремы 4.2.1, если положить  $A = I_n$ .)
- Обе матрицы  $A$  и  $A^T$  диагонально доминирующие и все  $a_{ii}$  положительны.

Доказательство того, что третье условие влечет положительную определенность, оставляем читателю.

#### 4.2.2. Несимметричные положительно определенные системы

Простое существование  $LDM^T$ -разложения не означает, что оно вычисляется удовлетворительно, потому что результирующие множители могут иметь неприемлемо большие элементы. Например, если  $\varepsilon > 0$ , то матрица

$$A = \begin{bmatrix} \varepsilon & m \\ -m & \varepsilon \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -m/\varepsilon & 1 \end{bmatrix} \begin{bmatrix} \varepsilon & 0 \\ 0 & \varepsilon + m^2/\varepsilon \end{bmatrix} \begin{bmatrix} 1 & m/\varepsilon \\ 0 & 1 \end{bmatrix}$$

является положительно определенной. Но если  $m/\varepsilon \gg 1$ , то рекомендуется применять выбор ведущего элемента.

Следующая теорема подсказывает, когда можно ожидать роста элементов при  $LDM^T$ -разложении положительно определенной матрицы.

**Теорема 4.2.4.** Пусть  $A \in \mathbb{R}^{n \times n}$  – положительно определенная матрица, и пусть  $T = (A + A^T)/2$  и  $S = (A - A^T)/2$ . Если  $A = LDM^T$ , то

$$\|L\| \|M^T\|_F \leq n (\|T\|_2 + \|ST^{-1}S\|_2). \quad (4.2.1)$$

**Доказательство.** См. Golub и Van Loan (1979).  $\square$

Теорема подсказывает, когда без всякого риска можно не делать выбора ведущего элемента. Допустим, что вычисленные множители  $\hat{L}$ ,  $\hat{D}$  и  $\hat{M}$  удовлетворяют условию

$$\| \hat{L} \| \| \hat{D} \| \| \hat{M}^T \|_F \leq c \|L\| \|D\| \|M^T\|_F, \quad (4.2.2)$$

где  $c$  – константа умеренной величины. Как следует из формулы (4.2.1) и из анализа в § 3.3, если эти множители использовать для вычисления решения системы  $Ax = b$ ,

то вычисленное решение  $\hat{x}$  удовлетворяет системе  $(A + E)\hat{x} = b$ , где

$$\|E\|_F \leq \mathbf{u}(3n\|A\|_F + 5cn^2(\|T\|_2 + \|ST^{-1}S\|_2)) + O(\mathbf{u}^2). \quad (4.2.3)$$

Легко показать, что  $\|T\|_2 \leq \|A\|_2$ , и тогда следует, что если

$$\Omega = \frac{\|ST^{-1}S\|_2}{\|A\|_2} \quad (4.2.4)$$

не слишком велико, то можно не делать выбор ведущего элемента. Другими словами, норма кососимметричной части  $S$  умерена относительно обусловленности симметричной части  $T$ . Иногда в приложениях бывает возможно оценить  $\Omega$ . Тривиальный случай, когда  $A$  симметрична, тогда  $\Omega = 0$ .

### 4.2.3. Симметричные положительно определенные системы

Когда мы применяем полученные выше результаты к симметричной положительно определенной системе, мы знаем, что разложение  $A = LDL^T$  существует и, более того, оно устойчиво к вычислениям. Тем не менее в этом случае используется другое разложение.

**Теорема 4.2.5 (Разложение Холецкого).** *Если матрица  $A \in \mathbb{R}^{n \times n}$  является симметричной положительно определенной, то существует единственная нижняя треугольная матрица  $G \in \mathbb{R}^{n \times n}$  с положительными диагональными элементами, такая, что  $A = GG^T$ .*

*Доказательство.* Согласно теореме 4.1.2 существует нижняя унитреугольная матрица  $L$  и диагональная матрица  $D = \text{diag}(d_1, \dots, d_n)$ , такие, что  $A = LDL^T$ . Так как элементы  $d_k$  положительны, то матрица  $G = L\text{diag}(\sqrt{d_1}, \dots, \sqrt{d_n})$  является вещественной нижней треугольной с положительными диагональными элементами. Также выполняется соотношение  $A = GG^T$ . Единственность следует из единственности  $LDL^T$ -разложения.  $\square$

Разложение  $A = GG^T$  известно как *разложение Холецкого*, а матрицы  $G$  называются *треугольниками Холецкого*. Заметим, что если мы вычислили разложение Холецкого и решаем треугольные системы  $Gy = b$  и  $G^Tx = y$ , то  $b = Gy = G(G^Tx) = (GG^T)x = Ax$ .

Наше доказательство существования разложения Холецкого в теореме 4.2.5 конструктивно. Тем не менее более эффективные методы для вычисления треугольника Холецкого могут быть получены преобразованием уравнения  $A = GG^T$ . Как мы покажем в следующих нескольких разделах, это может быть достигнуто различными способами.

### 4.2.4. Gaxpy-версия метода Холецкого

Сначала мы получим реализацию метода Холецкого, богатую операциями типа gaxpy. Если мы сравним  $j$ -е столбцы в уравнении  $A = GG^T$ , то мы получим

$$A(:, j) = \sum_{k=1}^j G(j, k)G(:, k).$$

Отсюда следует, что

$$G(j, j)G(:, j) = A(:, j) - \sum_{k=1}^{j-1} G(j, k)G(:, k) \equiv v. \quad (4.2.5)$$

Если мы знаем  $(j - 1)$  первых столбцов матрицы  $G$ , то может быть вычислен вектор  $v$ . Отсюда через сравнение компонент в формуле (4.2.5) следует, что  $G(j:n, j) = v(j:n)/\sqrt{v(j)}$ . Это и есть скалярная операция *гахру*, и поэтому мы получаем следующий метод, основанный на гахру-операции для вычисления разложения Холецкого:

```

for  $j = 1:n$ 
     $v(j:n) = A(j:n, j)$ 
    for  $k = 1:j - 1$ 
         $v(j:n) = v(j:n) - G(j, k)G(j:n, k)$ 
    end
     $G(j:n, j) = v(j:n)/\sqrt{v(j)}$ 
end
```

Вычисления можно организовать так, что  $G$  замещает нижние треугольные матрицы  $A$ .

**Алгоритм 4.2.1 (Метод Холецкого: Гахру-версия).** При данной симметричной положительно определенной матрице  $A \in \mathbb{R}^{n \times n}$  следующий алгоритм вычисляет нижнюю треугольную матрицу  $G \in \mathbb{R}^{n \times n}$ , такую, что  $A = GG^T$ . Для всех  $i \geq j$   $G(i, j)$  замещает  $A(i, j)$ .

```

for  $j = 1:n$ 
    if  $j > 1$ 
         $A(j:n, j) = A(j:n, j) - A(j:n, 1:j - 1)A(j, 1:j - 1)^T$ 
    end
     $A(j:n, j) = A(j:n, j)/\sqrt{A(j, j)}$ 
end
```

Алгоритм требует  $n^3/3$  флопов.

**Пример 4.2.1.** Матрица

$$\begin{bmatrix} 2 & -2 \\ -2 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{3} \end{bmatrix} \begin{bmatrix} \sqrt{2} & -\sqrt{2} \\ 0 & \sqrt{3} \end{bmatrix}$$

является положительно определенной.

#### 4.2.5. Метод Холецкого с внешним произведением

Альтернативная процедура Холецкого, основанная на модификации с внешним произведением, может быть получена из следующего разбиения:

$$A = \begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \beta & 0 \\ v/\beta & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - vv^T/\alpha \end{bmatrix} \begin{bmatrix} \beta & v^T/\beta \\ 0 & I_{n-1} \end{bmatrix}. \quad (4.2.6)$$

Здесь  $\beta = \sqrt{\alpha}$  и мы знаем, что  $\alpha > 0$ , потому что  $A$  положительно определена. Заметим, что  $B - vv^T/\alpha$  тоже положительно определена, потому что она является главной подматрицей матрицы  $X^TAX$ , где

$$X = \begin{bmatrix} 1 & -v^T/\alpha \\ 0 & I_{n-1} \end{bmatrix}.$$

Таким образом, если мы имеем разложение Холецкого  $G_1 G_1^T = B - vv^T/\alpha$ , то из формулы (4.2.6) следует, что  $A = GG^T$ , где

$$G = \begin{bmatrix} \beta & 0 \\ v/\beta & G_1 \end{bmatrix}.$$

Далее наша цель – многократно выполнить разбиение (4.2.6) до предельно маленьких подматриц, почти так же, как в  $kji$ -алгоритме исключения Гаусса. Если мы замещаем нижнюю треугольную часть матрицы  $A$  на матрицу  $G$ , то получается

**Алгоритм 4.2.2 (Метод Холецкого: версия с внешним произведением).** Данна симметричная положительно определенная матрица  $A \in \mathbb{R}^{n \times n}$ ; следующий алгоритм вычисляет нижнюю треугольную матрицу  $G \in \mathbb{R}^{n \times n}$ , такую, что  $A = GG^T$ . Для всех  $i \geq j$   $G(i, j)$  замещает  $A(i, j)$ .

```

for  $k = 1 : n$ 
     $A(k, k) = \sqrt{A(k, k)}$ 
     $A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k)$ 
    for  $j = k + 1 : n$ 
         $A(j : n, j) = A(j : n, j) - A(j : n, k) A(j, k)$ 
    end
end

```

Алгоритм требует  $n^3/3$  флопов. Заметим, что  $j$ -цикл вычисляет нижнюю треугольную часть модификации с внешним произведением

$$A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - A(k + 1 : n, k) A(k + 1 : n, k)^T.$$

Возвращаясь к нашим рассуждениям в § 1.4.8 о сравнении гахру-версии и модификации с внешним произведением, можно легко показать, что алгоритм 4.2.1 осуществляет в два раза меньший векторный обмен, чем алгоритм 4.2.2.

#### 4.2.6. Блочный метод Холецкого со скалярным произведением

Пусть  $A \in \mathbb{R}^{n \times n}$  – симметричная положительно определенная матрица. Будем рассматривать матрицу  $A = (A_{ij})$  и ее множители Холецкого  $G = G_{ij}$  как блочные  $N \times N$ -матрицы с квадратными диагональными блоками. Приравнивая  $(i, j)$ -е блоки из уравнения  $A = GG^T$  при  $i \geq j$ , мы получаем, что  $A_{ij} = \sum_{k=1}^j G_{ik} G_{jk}^T$ . Обозначив

$$S = A_{ij} - \sum_{k=1}^{j-1} G_{ik} G_{jk}^T,$$

мы видим, что  $G_{jj} G_{jj}^T = S$ , если  $i = j$ , и  $G_{ij} G_{jj}^T = S$ , если  $i > j$ . При правильном упорядочивании эти уравнения могут быть использованы для вычисления всех матриц  $G_{ij}$ .

**Алгоритм 4.2.3 (Метод Холецкого: блочная версия со скалярным произведением).** Данна симметричная положительно определенная матрица  $A \in \mathbb{R}^{n \times n}$ ; следующий алгоритм вычисляет нижнюю треугольную матрицу  $G \in \mathbb{R}^{n \times n}$ , такую, что  $A = GG^T$ . Нижний треугольник матрицы  $A$  замещается на нижнюю треугольную часть матрицы  $G$ . Матрица  $A$  рассматривается как блочная  $N \times N$ -матрица с квадратными диагональными блоками.

```

for  $j = 1 : N$ 
    for  $i = j : N$ 
         $S = A_{ij} - \sum_{k=1}^{j-1} G_{ik} G_{jk}^T$ 
        if  $i = j$ 

```

Вычислить разложение Холецкого  $S = G_{jj} G_{jj}^T$ .

```

else
    Решить  $G_{ij}G_{jj}^T = S$  относительно  $G_{ij}$ .
end
Выполнить замещение  $A_{ij}$  на  $G_{ij}$ .
end
end

```

Весь процесс включает  $n^3/3$  флопов, как и другие процедуры Холецкого, которые мы изучали. Процедура богата матричными умножениями, получаемыми при соответствующем блочном разбиении матрицы  $A$ . Например, если  $n = rN$  и каждая  $A_{ij}$  – это  $r \times r$ -матрица, тогда доля операций 3-го уровня аппроксимируется выражением  $1 - (1/N^2)$ .

Алгоритм 4.2.3 не полон в том смысле, что мы не уточнили, как выполняется произведение матриц  $G_{ik}G_{jk}$  или как вычисляется  $r \times r$ -разложение Холецкого  $S = G_{jj}G_{jj}^T$ . Эти важные детали должны быть тщательно проработаны для того, чтобы добиться хороших характеристик.

Другая блочная процедура может быть получена из гахру-алгоритма Холецкого. Через  $r$  шагов алгоритма 4.2.1 нам известны матрицы  $G_{11} \in \mathbf{R}^{r \times r}$  и  $G_{21} \in \mathbf{R}^{(n-r) \times r}$  в разложении

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} G_{11} & 0 \\ G_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} I_r & 0 \\ 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} G_{11} & 0 \\ G_{21} & I_{n-r} \end{bmatrix}^T.$$

После  $r$  шагов гахру-версии Холецкого мы преобразуем уже не матрицу  $A$ , а редуцированную матрицу  $\tilde{A} = A_{22} - G_{21}G_{21}^T$ , которую получили явно с учетом симметрии. Продолжая этот подход, мы получим блочный алгоритм Холецкого,  $k$ -й шаг которого содержит  $r$  шагов гахру-версии Холецкого для матрицы порядка  $n - (k-1)r$ , следовательно, вычисления 3-го уровня имеют порядок  $n - kr$ . Вклад операций 3-го уровня аппроксимируется соотношением  $1 - 3/(2N)$ , если  $n \approx rN$ .

#### 4.2.7. Устойчивость процесса Холецкого

Мы знаем, что в точной арифметике для симметричной положительно определенной матрицы существует разложение Холецкого. И, наоборот, если процесс Холецкого может быть реализован со строго положительными квадратными корнями, значит, матрица  $A$  положительно определена. Таким образом, чтобы выяснить, является ли матрица  $A$  положительно определенной, мы просто пытаемся вычислить ее разложение Холецкого, используя некоторые из методов, приведенных выше.

В условиях ошибок округления ситуация становится более интересной. Численная устойчивость алгоритма Холецкого следует из неравенства

$$g_{ij}^2 \leq \sum_{k=1}^i g_{ik}^2 = a_{ii}.$$

Оно показывает, что элементы треугольника Холецкого хорошо ограничены. Аналогичный вывод может быть получен из уравнения  $\|G\|_2^2 = \|A\|_2$ .

Ошибки округления в разложении Холецкого были широко исследованы в классической работе Уилкинсона (1968). Используя результаты этой работы, можно показать, что вычисленное решение  $\hat{x}$  системы  $Ax = b$ , полученное некоторой из наших процедур Холецкого, является решением возмущенной системы  $(A + E)\hat{x} = b$ ,

где  $\|E\|_2 \leq c_n \|A\|_2$  и  $c_n$  – маленькая константа, зависящая от  $n$ . Более того, Уилкинсон показал, что если  $q_n \|x\|_2(A) \leq 1$ , где  $q_n$  – другая маленькая константа, тогда процесс Холецкого может быть выполнен, т. е. не возникает квадратных корней из отрицательных чисел.

**Пример 4.2.2.** Если алгоритм 4.2.2 применяется к положительно определенной матрице

$$A = \begin{bmatrix} 100 & 15 & 0.01 \\ 15 & 2.3 & 0.01 \\ 0.01 & 0.01 & 1.00 \end{bmatrix}$$

и  $\beta = 10$ ,  $t = 2$ , причем используется приближенная арифметика, тогда  $\hat{g}_{11} = 10$ ,  $\hat{g}_{21} = 1.5$ ,  $\hat{g}_{31} = 0.001$  и  $\hat{g}_{22} = 0.00$ . Алгоритм обрывается при попытке вычислить  $\hat{g}_{32}$ .

#### 4.2.8. Неотрицательно определенные матрицы

Мы говорим, что матрица  $A$  будет *неотрицательно определенной*, если  $x^T A x \geq 0$  для любого вектора  $x$ . Симметричные неотрицательно определенные матрицы (*sps*) представляют большой интерес, и мы кратко обсудим некоторые преобразования типа Холецкого, которые могут быть использованы для решения различных *sps*-задач. Вначале необходимо получить результаты о свойствах диагональных элементов *sps*-матрицы.

**Теорема 4.2.6.** Если матрица  $A \in \mathbb{R}^{n \times n}$  является симметричной неотрицательно определенной, тогда

$$|a_{ij}| \leq (a_{ii} + a_{jj})/2, \quad (4.2.7)$$

$$|a_{ij}| \leq \sqrt{a_{ii} a_{jj}} \quad (i \neq j), \quad (4.2.8)$$

$$\max_{i,j} |a_{ij}| = \max_i |a_{ii}|, \quad (4.2.9)$$

$$a_{ii} = 0 \Rightarrow A(i, :) = 0, A(:, i) = 0. \quad (4.2.10)$$

*Доказательство.* Если вектор  $x = e_i + e_j$ , тогда  $0 \leq x^T A x = a_{ii} + a_{jj} + 2a_{ij}$ , если же  $x = e_i - e_j$ , то получаем  $0 \leq x^T A x = a_{ii} + a_{jj} - 2a_{ij}$ . Неравенство (4.2.7) следует из этих двух результатов. Равенство (4.2.9) очевидно следует из неравенства (4.2.7).

Для доказательства неравенства (4.2.8) допустим без ограничения общности, что  $i = 1$  и  $j = 2$ , и рассмотрим неравенство

$$0 \leq \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = a_{11}x^2 + 2a_{12}x + a_{22},$$

которое справедливо, поскольку матрица  $A(1:2, 1:2)$  является неотрицательно определенной. Чтобы это квадратное уравнение относительно  $x$  удовлетворяло неравенству, необходимо, чтобы дискриминант  $4a_{12}^2 - 4a_{11}a_{22}$  был отрицательным. Справедливость (4.2.10) следует из (4.2.8).  $\square$

Рассмотрим, что произойдет, если метод Холецкого с внешним произведением применить к *sps*-матрице. Если нам встретится нулевой элемент  $A(k, k)$ , тогда из (4.2.10) следует, что элементы  $A(k:n, k)$  нулевые, и «ничего не надо делать», поэтому мы получаем алгоритм

```

for  $k = 1:n$ 
  if  $A(k, k) > 0$ 
     $A(k, k) = \sqrt{A(k, k)}$ ;
     $A(k + 1:n, k) = A(k + 1:n, k)/A(k, k)$ 
    for  $j = k + 1:n$ 
       $A(j:n, j) = A(j:n, j) - A(j:n, k)A(j, k)$ 
    end
  end
end

```

Таким образом, простое изменение делает алгоритм 4.2.2 применимым в неотрицательно определенном случае. Однако на практике ошибки округления делают невозможным появление точных нулей и может быть предпочтительнее ввести в алгоритм выбор ведущего элемента.

#### 4.2.9. Симметричный выбор ведущего элемента

Для сохранения симметрии симметричной матрицы  $A$  мы рассмотрим только преобразования типа  $PAP^T$ , где  $P$  – матрица перестановок. Использование только строчных перестановок ( $A \leftarrow PA$ ) или столбцовых перестановок ( $A \leftarrow AP$ ) нарушает симметрию. Перестановка вида  $A \leftarrow PAP^T$  называется *симметричной перестановкой* матрицы  $A$ . Заметим, что такая операция *не переставляет* внедиагональные элементы на диагональ. Диагональ матрицы  $PAP^T$  – это переупорядоченная диагональ матрицы  $A$ .

Пусть в начале  $k$ -го шага алгоритма (4.2.11) мы симметрично переставили наибольший диагональный элемент матрицы  $A(k:n, k:n)$  в ведущую позицию. Если этот наибольший диагональный элемент нулевой, то  $A(k:n, k:n) = 0$  в силу (4.2.10). В этом случае мы можем вычислить разложение  $PAP^T = GG^T$ , где  $G \in \mathbb{R}^{(n \times (k-1))}$  – нижняя треугольная матрица.

**Алгоритм 4.2.4.** Пусть  $A \in \mathbb{R}^{n \times n}$  – симметричная неотрицательно определенная матрица и  $\text{rank}(A) = r$ . Следующий алгоритм вычисляет матрицу перестановок  $P$ , индекс  $r$  и нижнюю  $n \times r$  треугольную матрицу  $G$ , такие, что  $PAP^T = GG^T$ . Нижний треугольник  $A(:, 1:r)$  замещается на нижнюю треугольную часть матрицы  $G$ . Матрица  $P = P_r \dots P_1$ , где  $P_k$  – тождественная матрица с переставленными  $k$ -й и  $piv(k)$ -й строками.

```

 $r = 0$ 
for  $k = 1:n$ 
  Найти  $q$  ( $q \leqslant q \leqslant n$ ), такое, что  $A(q, q) = \max\{A(k, k) \dots A(n, n)\}$ .
  if  $A(q, q) > 0$ 
     $r = r + 1$ 
     $piv(k) = q$ 
     $A(k, :) \leftrightarrow A(q, :)$ 
     $A(:, k) \leftrightarrow A(:, q)$ 
     $A(k, k) = \sqrt{A(k, k)}$ 
     $A(k + 1:n, k) = A(k + 1:n, k)/A(k, k)$ 
    for  $j = k + 1:n$ 
       $A(j:n, j) = A(j:n, j) - A(j:n, k)A(j, k)$ 
    end
  end
end

```

На практике для определения малости  $A(k, k)$  используется некоторый барьер. Тем не менее ситуация бывает крайне непростой, и читатель может проконсультироваться у Higham (1989). Добавим, что в § 5.5 обсуждается выбор барьера в случае задачи неполного ранга. В заключение отметим, что правильное эффективное использование алгоритма 4.2.4 возможно, лишь когда осуществляется доступ только к нижней треугольной части матрицы  $A$ .

### Задачи

**4.2.1.** Предположим, что матрица  $A + iB$  эрмитова и положительно определенная, причем  $A, B \in \mathbb{R}^{n \times n}$ . Это означает, что  $x^T Ax > 0$  для любого вектора  $x \neq 0$ . (a) Показать, что матрица

$$C = \begin{bmatrix} A & -B \\ B & A \end{bmatrix}$$

симметричная и положительно определенная. (b) Сформулировать алгоритм сложности  $8n^3/3$  для решения системы  $(A + iB)(x + iy) = b + ic$ , где  $b, c, x$  и  $y$  принадлежат  $\mathbb{R}^n$ . Сколько памяти он требует?

**4.2.2.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметрична и положительно определена. Построить алгоритм для вычисления верхней треугольной матрицы  $R \in \mathbb{R}^{n \times n}$ , такой, что  $A = RR^T$ .

**4.2.3.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  положительно определена, и пусть  $T = (A + A^T)/2$ , а  $S = (A - A^T)/2$ . (a) Показать, что  $\|A^{-1}\|_2 \leq \|T^{-1}\|_2$  и  $x^T A^{-1} x \leq x^T T^{-1} x$  для всех  $x \in \mathbb{R}^n$ . (b) Показать, что если  $A = LDM^T$ , тогда  $d_k \geq 1/\|T^{-1}\|_2$  при  $k = 1:n$ .

**4.2.4.** Найти вещественную  $2 \times 2$ -матрицу  $A$ , обладающую свойством, что  $x^T Ax > 0$  для всех вещественных ненулевых векторов длины 2, но которая не является положительно определенной для векторов из множества  $\mathbb{C}^{2 \times 2}$ .

**4.2.5.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$ . Показать, что если обе матрицы  $A$  и  $A^T$  строго диагонально доминирующие, тогда  $A$  положительно определена. Диагональные элементы матрицы  $A$  положительны.

**4.2.6.** Показать, что функция  $f(x) = (x^T Ax)/2$  является векторной нормой на  $\mathbb{R}^n$  в том и только том случае, когда  $A$  положительно определена.

**4.2.7.** Модифицировать алгоритм 4.2.1 так, что если встречается квадратный корень из отрицательного элемента, то алгоритм находит единичный вектор  $x$ , удовлетворяющий условию  $x^T Ax < 0$ , и заканчивается.

**4.2.8.** Числовая сфера  $W(A)$  комплексной матрицы  $A$  определяется как множество  $W(A) = \{x^H Ax : x^H x = 1\}$ . Показать, что если  $W(A)$  не содержит нуля, то для матрицы  $A$  существует LU-разложение.

**4.2.9.** Используя SVD, показать, что если матрица  $A \in \mathbb{R}^{m \times n}$ , где  $m \geq n$ , тогда существуют матрицы  $Q \in \mathbb{R}^{m \times n}$  и  $P \in \mathbb{R}^{n \times n}$ , такие, что  $A = QP$ , где  $Q^T Q = I_n$  и  $P$ -симметричная и неотрицательно определенная. Это разложение иногда называется *полярным разложением* по аналогии с разложением комплексного числа  $z = e^{i \arg(z)} |z|$ .

**4.2.10.** Пусть матрица  $A = I + uu^T$ , причем  $A \in \mathbb{R}^{n \times n}$  и  $\|u\|_2 = 1$ . Дать точное описание диагонального и поддиагонального множителей Холецкого матрицы  $A$ .

### Замечания и литература к § 4.2

Определенность квадратичной формы  $x^T Ax$  часто может быть установлена при рассмотрении математических свойств исходной задачи. Например, дискретизация дифференциальных операторов частных производных дает возможность доказать положительную определенность матриц. Вопрос о необходимости выбора ведущего элемента при решении положительно определенных систем рассматривается в работе

Golub G. H. and Van Loan C. (1979). "Unsymmetric Positive Definite Linear Systems", Lin. Alg. and Its Appl. 28, 85–98.

Результаты этой статьи были вызваны более ранней работой по линейным системам вида  $(I + S)x = b$ , где  $S$  – кососимметричная матрица:

Buckley A. (1974). "A Note on Matrices  $A = I + H$ ,  $H$  Skew-Symmetric", Z. Angew. Math. Mech. 54, 125–126.

Buckley A. (1977). On the Solution of Certain Skew-Symmetric Linear Systems", SIAM J. Num. Anal. 14, 566–570.

Симметричные положительно определенные системы составляют наиболее важный класс задач  $Ax = b$  со спецификой. Алгоритмы для  $GG^T$  и  $LDL^T$  разложений даются в работе

Martin R. S., Peters G., and Wilkinson J. H. (1965). "Symmetric Decomposition of a Positive Definite Matrix", Numer. Math. 7, 362–383. See also HACLA, pp. 9–30.

Техника итерационного уточнения, которую мы обсуждали в § 3.5.3 в связи с LU-разложением, также может применяться при  $GG^T$  и  $LDL^T$ -разложениях. См.

Martin R. S., Peters G., and Wilkinson J. H. (1966). "Iterative Refinement of the Solution of a Positive Definite System of Equations", Numer. Math. 8, 203–2. See also HACLA, pp. 31–44.

Анализ ошибок округления для разложения Холецкого обсуждается в работах

Higham N. J. (1989). "Analysis of the Cholesky Decomposition of a Semi-definite Matrix", in Reliable Numerical Computation, eds. M. G. Cox and S. J. Hammarling, Oxford University Press.

Kielbasinski A. (1987). "A Note on Rounding Error Analysis of Cholesky Factorization", Lin. Alg. and Its. Applic. 88/89, 487–494.

Meinguet J. (1983). "Refined Error Analyses of Cholesky Factorization", SIAM J. Numer. Anal. 20, 1243–1250.

Wilkinson J. H. (1968). "À Priori Error Analysis of Algebraic Processes", Pro. International Congress Math. (Moscow: Izdat. Mir, 1968), p. 629–639.

Вопрос об изменении треугольника Холецкого  $G$  при возмущении матрицы  $A = GG^T$  исследуется в статье

Stewart G. W. (1977b). "Perturbation Bounds for the QR Factorization of a Matrix", SIAM J. Num. Anal. 14, 509–518.

Алгоритм для обращения симметричной положительно определенной матрицы без использования дополнительной памяти дается в работе

Bauer F. L. and Reinsch C. (1970). "Inversion of Positive Definite Matrices by the Gauss-Jordam Method", in HACLA, pp. 45–49.

Фортран-программы для решения симметричных положительно определенных систем имеются в LINPACK, гл. 3 и 8.

## 4.3. Ленточные системы

Во многих приложениях, в которых присутствуют линейные системы, матрица коэффициентов является *ленточной*. В этом случае уравнения всегда могут быть упорядочены так, что каждое неизвестное  $x_i$  появляется только в нескольких уравнениях, «соседних» с  $i$ -м уравнением. Формально мы говорим, что матрица  $A = (a_{ij})$  имеет *верхнюю ширину ленты*  $q$ , если  $a_{ij} = 0$  для всех  $j > i + q$ , и *нижнюю ширину ленты*  $p$ , если  $a_{ij} = 0$  для всех  $i > j + p$ . При решении ленточных систем можно добиться реальной экономии, так как треугольные множители  $LU$ ,  $GG^T$ ,  $LDM^T$  и т. д., также являются ленточными.

Прежде чем продолжить, мы советуем читателю просмотреть § 1.2, где обсуждаются некоторые аспекты преобразований с ленточными матрицами.

### 4.3.1. Ленточное LU-разложение

Наш первый результат показывает, что если матрица  $A$  является ленточной и  $A = LU$ , тогда  $L(U)$  сохраняет нижнюю (верхнюю) ширину ленты матрицы  $A$ .

**Теорема 4.3.1.** Пусть для матрицы  $A \in \mathbb{R}^{n \times n}$  существует LU-разложение  $A = LU$ . Если  $A$  имеет верхнюю ширину ленты  $q$  и нижнюю ширину ленты  $p$ , тогда  $U$  имеет верхнюю ширину ленты  $q$  и  $L$  имеет нижнюю ширину ленты  $p$ .

**Доказательство.** Доказательство проведем по индукции относительно  $n$ . Выпишем разложение

$$A = \begin{bmatrix} a & w^T \\ v & B \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & B - vw^T/a \end{bmatrix} \begin{bmatrix} a & w^T \\ 0 & I_{n-1} \end{bmatrix}.$$

Очевидно, что  $B - vw^T/a$  имеет верхнюю ширину ленты  $q$  и нижнюю ширину ленты  $p$ , потому что только первые  $q$  компонент  $w$  и первые  $p$  компонент  $v$  являются ненулевыми. Пусть  $L_1 U_1$  будет LU-разложением этой матрицы. Используя предположение индукции и разреженность  $w$  и  $v$ , получаем, что

$$L = \begin{bmatrix} 1 & 0 \\ v/a & L_1 \end{bmatrix} \quad \text{и} \quad U = \begin{bmatrix} a & w^T \\ 0 & U_1 \end{bmatrix}$$

имеют необходимые размеры ширины ленты и удовлетворяют соотношению  $A = LU$ .

Модификация исключения Гаусса для ленточных матриц, имеющих LU-разложение, получается непосредственно.

**Алгоритм 4.3.1 (Ленточное исключение Гаусса: версия с внешним произведением).** Данна матрица  $A \in \mathbb{R}^{n \times n}$  с верхней шириной ленты  $q$  и нижней шириной ленты  $p$ , следующий алгоритм вычисляет разложение  $A = LU$  при условии, что оно существует. Элементы  $A(i, j)$  замещаются на  $L(i, j)$ , если  $i > j$ , и на  $U(i, j)$  в противном случае.

```

for k = 1:n - 1
    for i = k + 1:min(k + p, n)
        A(i, k) = A(i, k)/A(k, k)
    end
    for j = k + 1:min(k + q, n)
        for i = k + 1:min(k + p, n)
            A(i, j) = A(i, j) - A(i, k) A(k, j)
        end
    end
end
end

```

Если  $n \gg p$  и  $n \gg q$ , то этот алгоритм содержит около  $2npq$  флоков. Также существуют ленточные версии алгоритма 4.1.1 ( $LDM^T$ ) и всех процедур Холецкого, мы оставляем их формулировку как упражнения.

### 4.3.2. Решение треугольных ленточных систем

Аналогичная экономия может быть получена при решении треугольных ленточных систем.

**Алгоритм 4.3.2 (Ленточная прямая подстановка: столбцовая версия).** Пусть матрица  $L \in \mathbb{R}^{n \times n}$  является нижней унитреугольной с нижней шириной ленты  $p$ . Для данного вектора  $b \in \mathbb{R}^n$  следующий алгоритм замещает  $b$  на решение системы  $Lx = b$ .

```

for  $j = 1 : n$ 
    for  $i = j + 1 : \min(j + p, n)$ 
         $b(i) = b(i) - L(i, j) b(j)$ 
    end
end

```

Если  $n \gg p$ , тогда этот алгоритм требует около  $2np$  флопов.

**Алгоритм 4.3.3 (Ленточная обратная подстановка: столбцовая версия).** Пусть  $U \in \mathbb{R}^{n \times n}$  является невырожденной треугольной матрицей с верхней шириной ленты  $q$ . Для данного вектора  $b \in \mathbb{R}^n$  следующий алгоритм замещает  $b$  на решение системы  $Ux = b$ .

```

for  $j = n : -1 : 1$ 
     $b(j) = b(j)/U(j, j)$ 
    for  $i = \max(1, j - q) : j - 1$ 
         $b(i) = b(i) - U(i, j) b(j)$ 
    end
end

```

Если  $n \gg q$ , то этот алгоритм требует около  $2nq$  флопов.

### 4.3.3. Структура данных ленточной матрицы

Приведенные выше алгоритмы записаны так, словно матрица  $A$  хранится обычным образом в массиве  $n \times n$ . На практике алгоритм решения ленточной линейной системы создается для структуры данных, которая имеет преимущества от наличия многих нулей в матрице  $A$ . Вспомним из § 1.2.6, что если матрица  $A$  имеет нижнюю ширину ленты  $p$  и верхнюю ширину ленты  $q$ , то она может быть представлена в массиве  $A.band$  размера  $(p + q + 1) \times n$ , в котором ленточный элемент  $a_{ij}$  хранится в позиции  $A.band(i - j + q + 1, j)$ . При таком размещении ненулевая часть  $i$ -го столбца матрицы  $A$  содержится в  $j$ -м столбце массива  $A.band$ . Другая возможная структура данных для ленточной матрицы, которую обсуждали в § 1.2.8, осуществляет хранение матрицы  $A$  по диагонали в 1-мерном массиве  $A.diag$ .

Независимо от принятой структуры данных, организация вычислений с ленточным расположением памяти требует тщательности, чтобы уменьшить дополнительные затраты на вычисление индексных выражений. Для пояснения рассмотрим выполнение алгоритма 4.3.1 с заменой ссылок на массив  $A$  соответствующими ссылками на массив  $A.band$ :

```

for  $k = 1 : n - 1$ 
    for  $i = k + 1 : \min(k + p, n)$ 
         $A.band(i - k + q + 1, k) =$ 
             $A.band(i - k + q + 1, k) / A.band(q + 1, k)$ 
    end
    for  $j = k + 1 : \min(k + q, n)$ 
        for  $i = k + 1 : \min(k + p, n)$ 
             $A.band(i - j + q + 1, j) = A.band(i - j + q + 1, j)$ 
             $- A.band(i - k + q + 1, k) A.band(k - j + q + 1, j)$ 
        end
    end
end

```

Хотя все сделано правильно, такой буквальный перевод алгоритма 4.3.1 не является оптимальным, потому что имеется излишек в индексной арифметике. Например, первый  $i$ -цикл лучше записать следующим образом:

```

 $\epsilon = q + 1$ 
for  $i = q + 2 : q + p + 1$ 
     $A.band(i, k) = A.band(i, k)/A(\epsilon, k)$ 
end
```

(Это сработает из-за «угловых нулей», наличие которых в массиве  $A.band$  мы допускаем, ср. § 1.2.6.) Создание эффективной ленточной процедуры требует преобразований такого рода. Умные компиляторы все сделают сами, но стремление к наибыстейшей реализации может потребовать, чтобы программист полностью пересмотрел целочисленные вычисления. Это одна из причин, почему программы для ленточных матриц неудобны для чтения.

#### 4.3.4. Ленточное исключение Гаусса с выбором ведущего элемента

Исключение Гаусса с частичным выбором тоже может быть приспособлено для использования ленточной структуры матрицы  $A$ . Тем не менее если  $PA = LU$ , то ленточные свойства матриц  $L$  и  $U$  не такие простые. Например, если  $A$  трехдиагональная матрица и ее первые две строки переставлялись на самом первом шаге алгоритма, то элемент  $a_{13}$  ненулевой. Следовательно, перестановки строк увеличивают ширину ленты. Как именно увеличивается лента – это содержание следующей теоремы.

**Теорема 4.3.2.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  невырождена и имеет верхнюю и нижнюю ширину ленты  $q$  и  $p$  соответственно. Если используется исключение Гаусса с частичным выбором для вычисления матриц преобразования Гаусса

$$M_j = I - a^{(j)} e_j^T, \quad j = 1 : n - 1$$

и матриц перестановок  $P_1 \dots P_{n-1}$ , таких, что  $M_{n-1} P_{n-1} \dots M_1 P_1 A = U$  будет верхняя треугольная матрица, тогда матрица  $U$  имеет верхнюю ширину ленты  $p + q$  и  $a_i^{(j)} = 0$  для всех  $i \leq j$  и  $i > j + p$ .

**Доказательство.** Пусть разложение  $PA = LU$  выполнено исключением Гаусса с частичным выбором, напомним, что  $P = P_{n-1} \dots P_1$ . Представим матрицу  $P^T = [e_{s_1}, \dots, e_{s_n}]$ , где  $\{s_1, \dots, s_n\}$  – перестановки набора  $\{1, 2, \dots, n\}$ . Если  $s_i > i + p$ , из этого следует, что ведущая главная подматрица матрицы  $PA$  порядка  $i \times i$  вырождена, так как

$$(PA)_{ij} = a_{s_i, j}, \quad j = 1 : s_i - p - 1$$

и  $s_i - p - 1 \geq i$ . Из этого вытекает, что матрицы  $U$  и  $A$  вырождены, что противоречит условию. Таким образом,  $s_i \leq i + p$  для  $i = 1 : n$ , и поэтому  $PA$  имеет ширину ленты  $p + q$ . Это следует из теоремы 4.3.1, согласно которой  $U$  имеет ширину ленты  $p + q$ .

Утверждение относительно  $a^{(j)}$  может быть установлено, если заметить, что матрица  $M_j$  должна обнулить только элементы  $(j+1, j), \dots, (j+p, j)$  частично уменьшенной матрицы  $P_j M_{j-1} P_{j-1} \dots P_1 A$ .  $\square$

Таким образом, выбор ведущего элемента нарушает структуру ленты в том смысле, что матрица  $U$  становится «толще», чем верхний треугольник матрицы  $A$ , и пока мы ничего не можем сказать о ширине ленты матрицы  $L$ . Тем не менее, так

как  $j$ -й столбец матрицы  $L$  переставлялся с  $j$ -м вектором Гаусса  $a_j$ , из этого следует, что матрица  $L$  имеет самое большое  $p + 1$  ненулевых элементов в столбце.

#### 4.3.5. LU-разложение матрицы Хессенберга

В качестве примера вычислений с несимметричной ленточной матрицей мы покажем, как исключение Гаусса с частичным выбором может быть применено для разложения верхней матрицы Хессенберга  $H$ . (Напомним, что  $H$  является верхней матрицей Хессенберга, если  $h_{ij} = 0$ ,  $i > j + 1$ .) После  $(k - 1)$ -го шага исключения Гаусса с частичным выбором у нас останется верхняя матрица Хессенберга вида

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad k = 3, \quad n = 6.$$

Благодаря специальной структуре этой матрицы мы видим, что следующая матрица перестановок  $P_3$  или тождественная, или совпадает с матрицей перестановок 3-й и 4-й строк. Более того, следующая матрица преобразования Гаусса  $M_k$  имеет единственный ненулевой множитель в позиции  $(k + 1, k)$ . Это поясняет  $k$ -й шаг следующего алгоритма.

**Алгоритм 4.3.4 (LU-разложение матрицы Хессенберга).** Для данной верхней треугольной матрицы Хессенберга  $H \in \mathbf{R}^{n \times n}$  следующий алгоритм вычисляет верхнюю треугольную матрицу  $M_{n-1} P_{n-1} \dots M_1 P_1 A = U$ , здесь каждая матрица  $P_k$  является перестановочной, а каждая матрица  $M_k$  — матрица преобразования Гаусса, элементы которой ограничены единицей. Элементы  $H(i, k)$  замещаются элементами  $U(i, k)$ , если  $i \leq k$ , и элементами  $(M_k)_{k+1, k}$ , если  $i = k + 1$ . Целочисленный вектор  $piv(1 : n - 1)$  задает перестановки. Если  $P_k = I$ , то  $piv(k) = 0$ . Если  $P_k$  переставляет  $k$ -ю и  $(k + 1)$ -ю строки, то  $piv(k) = 1$ .

```

for  $k = 1 : n - 1$ 
  if  $|H(k, k)| < |H(k + 1, k)|$ 
     $piv(k) = 1$ ;  $H(k, k:n) \leftrightarrow H(k + 1, k:n)$ 
  else
     $piv(k) = 0$ 
  end
  if  $H(k, k) \neq 0$ 
     $t = -H(k + 1, k)/H(k, k)$ 
    for  $j = k + 1 : n$ 
       $H(k + 1, j) = H(k + 1, i) + t H(k, j)$ 
    end
     $H(k + 1, k) = t$ 
  end
end

```

Этот алгоритм требует  $n^2$  флопов.

### 4.3.6. Ленточный метод Холецкого

Остаток этого раздела посвящен ленточной задаче  $Ax = b$ , где матрица  $A$  является симметричной положительно определенной. Тот факт, что выбор ведущего элемента не нужен для таких матриц, приводит к некоторым очень компактным, элегантным алгоритмам. В частности, как следует из теоремы 4.3.1, если  $A = GG^T$  является разложением Холецкого матрицы  $A$ , то матрица  $G$  имеет такую же нижнюю ширину ленты, как и матрица  $A$ . Это приводит к следующей ленточной версии алгоритма 4.2.1 метода Холецкого, основанного на гахру-операции.

**Алгоритм 4.3.5 (Ленточный метод Холецкого: Гахру-версия).** Для данной симметричной положительно определенной матрицы  $A \in \mathbb{R}^{n \times n}$  с шириной ленты  $p$  следующий алгоритм вычисляет нижнюю треугольную матрицу  $G$  с нижней шириной ленты  $p$ , такую, что  $A = GG^T$ . Для всех  $i \geq j$  матрица  $G(i, j)$  замещает элементы  $A(i, j)$ .

```

for  $j = 1:n$ 
  for  $k = \max(1, j - p):j - 1$ 
     $\lambda = \min(k + p, n)$ 
     $A(j:\lambda, j) = A(j:\lambda, j) - A(j, k)A(j:\lambda, k)$ 
  end
   $\lambda = \min(j + p, n)$ 
   $A(j:\lambda, j) = A(j:\lambda, j)/\sqrt{A(j, j)}$ 
end

```

Если  $n \gg p$ , тогда алгоритм требует около  $n(p^2 + 3p)$  флопов и  $n$  квадратных корней. Конечно, в серьезных приложениях должна быть использована соответствующая структура данных при задании матрицы  $A$ . Например, если мы храним только нижнюю треугольную часть, тогда будет достаточным массив размером  $(p + 1) \times n$ . (См. § 1.2.6.)

Если наша ленточная процедура Холецкого сочетается с программами решения ленточных треугольных систем, тогда решение задачи  $Ax = b$  потребует приблизительно  $np^2 + 7np + 2n$  флопов и  $n$  квадратных корней. Из этого следует, что для маленьких  $p$  квадратные корни представляют значительную часть вычислений и предпочтительнее использовать LDL-подход. Действительно, тщательный подсчет флопов на этапах  $A = LDL^T$ ,  $Ly = b$ ,  $Dz = y$  и  $L^Tx = z$  обнаруживает, что требуется  $np^2 + 8np + p$  флопов без квадратных корней.

### 4.3.7. Решение трехдиагональных систем

Как образец применения  $LDL^T$ -процедуры для решения системы с узкой лентой рассмотрим случай симметричных положительно определенных трехдиагональных систем. Пусть

$$L = \begin{bmatrix} 1 & & & \dots & & 0 \\ e_1 & 1 & & & & \vdots \\ \vdots & \ddots & \ddots & & & \vdots \\ 0 & \dots & & e_{n-1} & & 1 \end{bmatrix}$$

и  $D = \text{diag}(d_1, \dots, d_n)$ , тогда из уравнения  $A = LDL^T$  мы получаем, что

$$\begin{aligned} a_{11} &= d_1, \\ a_{k,k-1} &= e_{k-1} d_{k-1}, & k = 2:n, \\ a_{kk} &= d_k + e_{k-1}^2 d_{k-1} = d_k + e_{k-1} a_{k,k-1}, & k = 2:n. \end{aligned}$$

Таким образом,  $d_i$  и  $e_i$  могут быть найдены из следующей процедуры:

```
 $d_1 = a_{11}$ 
for  $k = 2:n$ 
     $e_{k-1} = a_{k,k-1}/d_{k-1};$      $d_k = a_{kk} - e_{k-1} a_{k,k-1}$ 
end
```

Чтобы получить решение системы  $Ax = b$ , мы решаем системы  $Ly = b$ ,  $Dz = y$  и  $L^T x = z$ . С учетом замещения получаем

**Алгоритм 4.3.6 (Решение симметричной трехдиагональной положительно определенной системы).** Для данной  $n \times n$  симметричной трехдиагональной положительно определенной матрицы  $A$  и вектора  $b \in \mathbb{R}^n$  следующий алгоритм замещает  $b$  на решение системы  $Ax = b$ . Предполагается, что диагональ матрицы  $A$  хранится в векторе  $d(1:n)$ , а поддиагональ – в векторе  $e(1:n-1)$ .

```
for  $k = 2:n$ 
     $t = e(k-1);$      $e(k-1) = t/d(k-1);$      $d(k) = d(k) - te(k-1)$ 
end
for  $k = 2:n$ 
     $b(k) = b(k) - e(k-1)b(k-1)$ 
end
 $b(n) = b(n)/d(n)$ 
for  $k = n-1:-1:1$ 
     $b(k) = b(k)/d(k) - e(k)b(k+1)$ 
end
```

Алгоритм требует  $8n$  флопов.

#### 4.3.8. Вопросы векторизации

Пример с трехдиагональной системой затрагивает болевую точку: задачи с узкой лентой и векторно-конвейерные архитектуры плохо совмещаются. Узкая лента приводит к коротким векторам. Тем не менее иногда бывает, что большой независимый поток таких задач может решаться в одно время. Давайте посмотрим, как могут быть организованы такие вычисления в свете вопросов, поднятых в § 1.4.

Для простоты предположим, что мы должны решать  $n \times n$  нижние унитреугольные двухдиагональные системы

$$A^{(k)} x^{(k)} = b^{(k)}, \quad k = 1:m,$$

и что  $m \gg n$ . Допустим, у нас есть массивы  $E(1:n-1, 1:m)$  и  $B(1:n, 1:m)$ , такие, что  $E(1:n-1, k)$  содержит поддиагональ матрицы  $A^{(k)}$ , а  $B(1:n, k)$  содержит  $k$ -ю правую часть  $b^{(k)}$ . Мы можем выполнить замещение  $b^{(k)}$  на решение  $x^{(k)}$  следующим образом:

```
for  $k = 1:m$ 
    for  $i = 2:n$ 
         $B(i, k) = B(i, k) - E(i-1, k)B(i-1, k)$ 
    end
end
```

Трудность, связанная с этим алгоритмом, который по очереди последовательно решает каждую двухдиагональную систему, состоит в том, что внутренний цикл не векторизуем. Это следствие того, что  $B(i, k)$  зависит от  $B(i - 1, k)$ . Если мы переставим  $k$ -й и  $i$ -й циклы, получим

```
for  $i = 2:n$ 
  for  $k = 1:m$ 
     $B(i, k) = B(i, k) - E(i - 1, k) B(i - 1, k)$ 
  end
end
```

(4.3.1)

Сейчас внутренний цикл хорошо векторизуется, так как он содержит векторное сложение и векторное умножение. К несчастью, алгоритм (4.3.1) является процедурой с шагом, не равным единице. Тем не менее эта трудность легко исправима, если хранить поддиагонали и правые части по строкам. То есть мы используем массивы  $E(1:m, 1:n-1)$  и  $B(1:m, 1:n-1)$  и храним поддиагональ матрицы  $A^{(k)}$  в  $E(k, 1:n-1)$  и  $b^{(k)T}$  в  $B(k, 1:n)$ . Вычисления (4.3.1) могут быть преобразованы к виду

```
for  $i = 2:n$ 
  for  $k = 1:m$ 
     $B(k, i) = B(k, i) - E(k, i - 1) B(k, i - 1)$ 
  end
end
```

который снова иллюстрирует влияние структур данных на производительность.

### Задачи

4.3.1. Построить ленточную процедуру  $LDM^T$  подобно алгоритму 4.3.1.

4.3.2. Показать, как результат алгоритма 4.3.4 может быть использован для решения верхней хессенберговой системы  $Hx = b$ .

4.3.3. Предложить алгоритм для решения несимметричной трехдиагональной системы  $Ax = b$ , который использует исключение Гаусса с частичным выбором и требует хранения только четырех векторов длины  $n$ , заданных в плавающей арифметике.

4.3.4. Для матрицы  $G \in \mathbb{R}^{n \times n}$  определен индекс профиля  $m(C, i) = \min\{j: c_{ij} \neq 0\}$ , где  $i = 1:n$ . Показать, что если  $A = GG^T$  – это разложение Холецкого матрицы  $A$ , то  $m(A, i) = m(G, i)$  для  $i = 1:n$ . (Говорят, что  $G$  и  $A$  имеют одинаковый профиль.)

4.3.5. Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметричная положительно определенная с индексами профиля  $m_i = m(A, i)$ , где  $i = 1:n$ . Допустим, что матрица  $A$  хранится в одномерном массиве  $v$  следующим образом:  $v = (a_{11}, a_{2, m_2}, \dots, a_{22}, a_{3, m_3}, a_{33}, \dots, a_{nn})$ . Написать алгоритм, который замещает  $v$  на соответствующие элементы множителя Холецкого  $G$  и потом использует это разложение для решения системы  $Ax = b$ . Сколько требуется флопов?

4.3.6. Для матрицы  $C \in \mathbb{R}^{n \times n}$  задан индекс  $p(C, i) = \max\{j: c_{ij} \neq 0\}$ . Предположим, что для матрицы  $A \in \mathbb{R}^{n \times n}$  известно LU-разложение  $A = LU$ , а также

$$\begin{aligned} m(A, 1) &\leq m(A, 2) \leq \dots \leq m(A, n), \\ p(A, 1) &\leq p(A, 2) \leq \dots \leq p(A, n). \end{aligned}$$

Показать, что  $m(A, i) = m(L, i)$  и  $p(A, i) = p(U, i)$  для  $i = 1:n$ . Вспомнить определение  $m(A, i)$  из задачи 4.3.4.

4.3.7. Построить гахру-версию алгоритма 4.3.1.

4.3.8. Предложить векторизуемый алгоритм с единичным шагом для решения симметричных положительно определенных трехдиагональных систем  $A^{(k)}x^{(k)} = b^{(k)}$ . Допускается, что диаго-

нали, поддиагонали и правые части хранятся по строкам в массивах  $D$ ,  $E$  и  $B$ , и что  $b^{(k)}$  замещается на  $x^{(k)}$ .

**4.3.9.** Предложить версию алгоритма 4.3.1, в которой матрица  $A$  хранится по диагоналям.

**4.3.10.** Построить пример симметричной положительно определенной  $3 \times 3$ -матрицы, трехдиагональная часть которой не является положительно определенной.

### Замечания и литература к § 4.3

Мы хотим подчеркнуть, что наш подсчет флопов означает только оценку объема работы. Читатель не должен придавать большого значения их точному количеству, особенно в области ленточных матриц, где так много зависит от разумной модификации.

Фортран-программы для линейных ленточных систем можно найти в Linpack, гл. 2, 4 и 7. Литература, имеющая отношение к ленточным системам, огромна. Мы включили некоторые наиболее представительные статьи:

- Allgower E. L. (1973). "Exact Inverses of Certain Band Matrices", Numer. Math. 21, 279–284.  
 Bohte Z. (1975). "Bounds for Rounding Errors in the Gaussian Elimination for Band Systems", J. Inst. Math. Applic. 16, 133–142.  
 Duff I. S. (1977). "A Survey of Sparse Matrix Research", Proc. IEEE 65, 500–535.  
 Martin R. S. and Wilkinson J. H. (1965). "Symmetric Decomposition of Positive Definite Band Matrices", Numer. Math. 7, 355–361. See also HACLA, pp. 50–56.  
 Martin R. S. and Wilkinson J. H. (1967). "Solution of Symmetric and Unsymmetric Band Equations and the Calculation of Eigenvalues of Band Matrices", Numer. Math. 9, 279–301. See also HACLA, pp. 70–92.
- Значительный интерес в области решения задач с ленточными матрицами представляет тема, связанная с методами для уменьшения ширины ленты. См.
- Cuthill E. (1972). "Several Strategies for Reducing the Bandwidth of Matrices", in Sparse Matrices and Their Applications, ed. D. J. Rose and R. A. Willoughby, Plenum Press, New York.  
 Gibbs N. E., Poole W. G., Jr., and Stockmeyer P. K. (1976). "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix", SIAM J. Num. Anal. 13, 236–250.  
 Gibbs N. E., Poole W. G., Jr., and Stockmeyer P. K. (1976). "A Comparison of Several Bandwidth and Profile Reduction Algorithms", ACM Trans. Math. Soft. 2, 322–330.

Как мы упоминали, трехдиагональные системы возникают чрезвычайно часто. Поэтому нет ничего удивительного, что уделяется огромное внимание специальным методам для решения этого класса ленточных задач:

- Fischer C. and Usmani R. A. (1969). "Properties of Some Tridiagonal Matrices and Their Application to Boundary Value Problems", SIAM J. Num. Anal. 6, 127–142.  
 Higham N. J. (1986c). "Efficient Algorithms for computing the condition number of a tridiagonal matrix", SIAM J. Sci. and Stat. Comp. 7, 150–165.  
 Kershaw D. (1982). "Solution of Single Tridiagonal Linear Systems and Vectorization of the ICCG Algorithm on the Cray-1", in Parallel Computation, ed. G. Rodrigue, Academic Press, NY, 1982.  
 Lambiotte J. and Voiht R. G. (1975). "The Solution of Tridiagonal Linear Systems of the CDC-STAR 100 Computer", ACM Trans. Math. Soft. 1, 308–329.  
 Malcolm M. A. and Palmer J. (1974). "A Fast Method for Solving A Class of Tridiagonal Systems of Linear Equations", Comm. ACM 17, 14–17.  
 Rose D. J. (1969). "An Algorithm for Solving a Special Class of Tridiagonal Systems of Linear Equations", Comm. ACM 12, 234–236.  
 Stone H. S. (1973). "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations", J. ACM 20, 27–38.  
 Stone H. S. (1975). "Parallel Tridiagonal Equation Solvers", ACM Trans. Math. Soft. 1, 289–307.

В гл. 4 работы

George J. A. and Liu J. W. (1981). Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, New Jersey

содержится прекрасный обзор по ленточным методам для положительно определенных систем.

### Добавления при переводе

В следующей работе установлен явный вид матриц, обратных к блочно-ленточной и блочно-хессенберговой:

Нечепуренко Ю. М. Об одной факторизации элементов обратной матрицы. – ЖВМ и МФ. 1984, Т. 24, № 4, с. 601–605.

Параллельный алгоритм для трехдиагональной системы порядка  $n$ , требующий  $3\log n$  тактов  $4n$ -процессорной ЭВМ, описан в работе

Нечепуренко Ю. М. Полиномиально устойчивый быстрый параллельный алгоритм для трехдиагональных систем. – ЖВМ и МФ, 1986, Т. 26, № 7, с. 963–969.

## 4.4. Симметричные неопределенные системы

Симметричная матрица, квадратичная форма которой  $x^T Ax$  принимает и положительные, и отрицательные значения, называется *неопределенной*. Хотя неопределенная матрица может иметь  $LDL^T$ -разложение, элементы множителей могут быть произвольной величины:

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1/\varepsilon & 1 \end{bmatrix} \begin{bmatrix} \varepsilon & 0 \\ 0 & -1/\varepsilon \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1/\varepsilon & 1 \end{bmatrix}^T.$$

Конечно, можно призвать на помощь некоторые стратегии с выбором из § 3.4. Однако они теряют симметрию, и вместе с этим пропадает шанс для построения метода решения неопределенных систем «со скоростью Холецкого». Надо применить, согласно нашим рекомендациям из § 4.2.9, симметричный выбор ведущего элемента, т. е. преобразование данных вида  $A \leftarrow PAP^T$ . К несчастью, симметричный выбор не всегда устойчив при вычислении  $LDL^T$ -разложения. Если  $\varepsilon_1$  и  $\varepsilon_2$  малы, то независимо от матрицы  $P$  матрица

$$\tilde{A} = P \begin{bmatrix} \varepsilon_1 & 1 \\ 1 & \varepsilon_2 \end{bmatrix} P^T$$

имеет маленькие элементы на диагонали и в процессе разложения появляются большие числа. При симметричном выборе ведущие элементы всегда выбираются из диагонали и получаются ужасные результаты, если эти числа малы относительно тех внедиагональных элементов, которые обнуляются. Таким образом,  $LDL^T$ -разложение не может быть рекомендовано как надежный подход к решению симметричной неопределенной системы. По-видимому, это действительно очень сложная задача – включить внедиагональные элементы в процесс выбора ведущего элемента и при этом сохранить симметрию.

В данном разделе мы обсудим два способа, как это можно сделать. Первый метод предложен Aasen (1971), он вычисляет разложение

$$PAP^T = LTL^T, \quad (4.4.1)$$

где  $L = (l_{ij})$  – нижняя унитреугольная матрица, а  $T$  – трехдиагональная. Матрица  $P$  является перестановкой и выбирается так, что  $|l_{ij}| \leq 1$ . Метод с *диагональным выбором* вычисляет матрицу  $P$  так, что

$$PAP^T = LDL^T, \quad (4.4.2)$$

где матрица  $D$  состоит из блоков размеров  $1 \times 1$  и  $2 \times 2$ . И снова матрица  $P$  выбирается так, что элементы нижней унитреугольной матрицы  $L$  удовлетворяют условию  $|l_{ij}| \leq 1$ . Оба разложения требуют  $n^3/3$  флопов и после вычисления могут

использоваться для решения системы  $Ax = b$  с объемом работы  $O(n^2)$ :

$$PAP^T = LTL^T, \quad Lz = Pb, \quad Tw = z, \quad L^Ty = w, \quad x = Py \Rightarrow Ax = b,$$

$$PAP^T = LDL^T, \quad Lz = Pb, \quad Dw = z, \quad L^Ty = w, \quad x = Py \Rightarrow Ax = b.$$

В этих процедурах имеется только одна «новая» тема, которую имеет смысл обсудить – это решение систем  $Tw = z$  и  $Dw = z$ .

В методе Аазена симметричная неопределенная трехдиагональная система решается за  $O(n)$  операций при помощи исключения Гаусса с выбором. Отметим, что на этом уровне вычислений не будет серьезной платы за невнимание к симметрии, поскольку общий объем вычислений  $O(n^3)$ .

В подходе с диагональным выбором система  $Dw = z$  составлена из множества симметричных неопределенных систем порядка  $1 \times 1$  и  $2 \times 2$ . Задачи порядка  $2 \times 2$  могут быть решены посредством исключения Гаусса с выбором. Опять же не будет вреда от пренебрежения симметрией на этой фазе вычислений сложности  $O(n)$ .

Таким образом, центральный вопрос этого раздела – это эффективное вычисление разложений (4.4.1) и (4.4.2).

#### 4.4.1. Алгоритм Парлетта–Рейда

Парлетт и Рейд (1970) показали, как, используя преобразования Гаусса, можно вычислить разложение (4.4.1). Их алгоритм достаточно хорошо просматривается на примере  $k = 2$  шагов случая  $n = 5$ . На начальном шаге матрица  $A$  преобразована к виду

$$A^{(1)} = M_1 P_1 A P_1^T M_1^T = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & 0 \\ \beta_1 & \alpha_2 & v_3 & v_4 & v_5 \\ 0 & v_3 & \times & \times & \times \\ 0 & v_4 & \times & \times & \times \\ 0 & v_5 & \times & \times & \times \end{bmatrix},$$

где перестановочная матрица  $P_1$  выбирается так, что элементы матрицы преобразования Гаусса  $M_1$  ограничены по модулю единицей. Изучая вектор  $(v_3, v_4, v_5)^T$  на предмет выбора максимального элемента, мы определим  $3 \times 3$  перестановочную матрицу  $\tilde{P}_2$ , такую, что

$$\tilde{P}_2 \begin{bmatrix} v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} \tilde{v}_3 \\ \tilde{v}_4 \\ \tilde{v}_5 \end{bmatrix} \Rightarrow |\tilde{v}_3| = \max\{|\tilde{v}_3|, |\tilde{v}_4|, |\tilde{v}_5|\}.$$

Если максимальный элемент нулевой, мы положим  $M_2 = P_2 - I$  и перейдем к следующему шагу. В противном случае мы положим  $\tilde{P}_2 = \text{diag}(I_2, \tilde{v}_3)$  и  $M_2 = I - \alpha^{(2)} e_3^T$ , где

$$\alpha^{(2)} = (0 \ 0 \ 0 \ \tilde{v}_4/\tilde{v}_3 \ \tilde{v}_5/\tilde{v}_3)^T,$$

и заметим, что

$$A^{(2)} = M_2 P_2 A^{(1)} P_2^T M_2^T = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & 0 \\ \beta_1 & \alpha_2 & \tilde{v}_3 & 0 & 0 \\ 0 & \tilde{v}_3 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}.$$

В общем случае процесс продолжается в течение  $n - 2$  шагов, предоставляя нам трехдиагональную матрицу

$$T = A^{(n-2)} = (M_{n-2} P_{n-2} \dots M_1 P_1) A (M_{n-2} P_{n-2} \dots M_1 P_1)^T.$$

Можно показать, что формула (4.4.1) содержит матрицы  $P = P_{n-2} \dots P_1$  и

$$L = (M_{n-2} P_{n-2} \dots M_1 P_1)^{-1}.$$

Анализируя матрицу  $L$ , обнаруживаем, что ее первый столбец – это вектор  $e_1$ , а все ее поддиагональные элементы в  $k$ -м столбце при  $k > 1$  «задаются» множителями матрицы  $M_{k-1}$ .

Эффективная реализация метода Парлетта–Рейда требует аккуратности, когда вычисляется преобразование

$$A^{(k)} = M_k (P_k A^{(k-1)} P_k^T) M_k^T. \quad (4.4.3)$$

Чтобы разобраться, в чем трудности при минимуме обозначений, предположим, что матрица  $B = B^T$  имеет порядок  $n - k$ , и пусть нам надо выполнить преобразование  $B_+ = (I - we_1^T)B(I - we_1^T)^T$ , где  $w \in \mathbb{R}^{n-k}$  и  $e_1$  – это первый столбец матрицы  $I_{n-k}$ . Такие вычисления совпадают с формулой (4.4.3). Если положить

$$u = Be_1 - \frac{b_{11}}{2}w,$$

то нижняя половина симметричной матрицы  $B_+ = B - uw^T - uw^T$  может быть вычислена за  $2(n - k)^2$  флопов. Суммируя эту величину по переменной  $k$  от 1 до  $n - 2$ , получим, что процедура Парлетта–Рейда требует  $2n^3/3$  флопов, что в два раза больше, чем хотелось бы.

**Пример 4.4.1.** Если алгоритм Парлетта–Рейда применяется к матрице

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 2 & 2 \\ 2 & 2 & 3 & 3 \\ 3 & 2 & 3 & 4 \end{bmatrix},$$

то

$$\begin{aligned} P_1 &= [e_1 \ e_4 \ e_3 \ e_2], \\ M_1 &= I_4 - (0, 0, 2/3, 1/3)^T e_2^T, \\ P_2 &= [e_1 \ e_2 \ e_4 \ e_3], \\ M_2 &= I_4 - (0, 0, 0, 1/2)^T e_3^T \end{aligned}$$

и  $PAP^T = LTL^T$ , где  $P = [e_1, e_3, e_4, e_2]$ ,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1/3 & 1 & 0 \\ 0 & 2/3 & 1/2 & 1 \end{bmatrix} \text{ и } T = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 4 & 2/3 & 0 \\ 0 & 2/3 & 10/9 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix}.$$

#### 4.4.2. Метод Аазена

Метод сложности  $n^3/3$ , предложенный Аазеном (1971), может быть получен пересмотром некоторых вычислительных деталей метода Парлетта–Рейда. Введем

необходимые обозначения для трехдиагональной матрицы  $T$ :

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ & & & \ddots & \beta_{n-1} \\ 0 & \cdots & \vdots & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Для ясности мы временно игнорируем выбор ведущего элемента и допускаем, что разложение  $A = LTL^T$  существует, здесь  $L$  – нижняя унитреугольная матрица со столбцом  $L(1, 1) = e_1$ . Метод Аазена организован следующим образом:

```

for  $j = 1:n$ 
    Вычислить  $h(1:j)$ , где  $h = TL^Te_j = He_j$ .
    Вычислить  $\alpha(j)$ .
    if  $j \leq n - 1$ 
        Вычислить  $\beta(j)$ .
    end
    if  $j \leq n - 2$ 
        Вычислить  $L(i + 2:n, j + 1)$ .
    end

```

Таким образом, работа  $j$ -го шага метода Аазена состоит в вычислении  $j$ -го столбца матрицы  $T$  и  $(j+1)$ -го столбца матрицы  $L$ . Алгоритм использует тот факт, что матрица  $H = TL^T$  является верхней хессенберговой. Вектор  $h(1:j) = H(1:j, j)$  содержит необходимую информацию для вычисления коэффициентов  $\alpha(j)$ ,  $\beta(j)$  и  $L(j+2:n, j+1)$  по формуле (4.4.4). Давайте посмотрим почему.

Рассмотрим  $j$ -й столбец уравнения  $A = LH$ :

$$A(:, j) = L(:, 1:j+1)h(1:j+1). \quad (4.4.5)$$

Соотношение (4.4.5) показывает, что вектор  $A(:, j)$  является линейной комбинацией первых  $(j+1)$  столбцов матрицы  $L$ . В частности,

$$A(j+1:n, j) = L(j+1:n, 1:j)h(1:j) + L(j+1:n, j+1)h(j+1).$$

Отсюда следует, что если вычислить

$$v(j+1:n) = A(j+1:n, j) - L(j+1:n, 1:j)h(1:j),$$

то

$$L(j+1:n, j+1)h(j+1) = v(j+1:n). \quad (4.4.6)$$

Таким образом, вектор  $L(j+2:n, j+1)$  коллинеарен вектору  $v(j+2:n)$ . Так как матрица  $L$  нижняя унитреугольная, мы получаем из (4.4.6), что

$$v(j+1) = h(j+1),$$

и, следовательно, учитывая, что это одно и то же уравнение, мы получаем следующую формулу для  $(j+1)$ -го столбца матрицы  $L$ :

$$L(j+2:n, j+1) = v(j+2:n)/v(j+1).$$

Заметим, что  $L(j+2:n, j+1)$  задается скалярной операцией  $\text{gaxp}$ .

Теперь получим формулы для  $\alpha(j)$  и  $\beta(j)$ . Сравним  $(j, j)$ -й и  $(j + 1, j)$ -й элементы в уравнении  $H = TL^T$ . С учетом условия  $\beta(0) = 0$  мы находим, что  $h(j) = \beta(j - 1)L(j, j - 1) + \alpha(j)$  и  $h(j + 1) = v(j + 1)$ , следовательно,

$$\begin{aligned}\alpha(j) &= h(j) - \beta(j - 1)L(j, j - 1), \\ \beta(j) &= v(j + 1).\end{aligned}$$

С учетом этих формул мы можем полностью описать процедуру Азена:

```

for  $j = 1 : n$ 
    Вычислить  $h(1:j)$ , где  $h = TL^T e_j$ 
    if  $j = 1 \vee j = 2$ 
         $\alpha(j) = h(j)$ 
    else
         $\alpha(j) = h(j) - \beta(j - 1)L(j, j - 1)$ 
    end
    if  $j \leq n - 1$ 
         $v(j + 1:n) = A(j + 1:n, j) - L(j + 1:n, 1:j)h(1:j)$ 
         $\beta(j) = v(j + 1)$ 
    end
    if  $j \leq n - 2$ 
         $L(j + 2:n, j + 1) = v(j + 2:n)/v(j + 1)$ 
    end
end
```

(4.4.7)

Для полноты описания нам необходимо показать в деталях вычисление вектора  $h(1:j)$ . Из (4.4.5) следует, что

$$A(1:j, j) = L(1:j, 1:j)h(1:j). \quad (4.4.8)$$

Эта нижняя треугольная система может быть решена относительно  $h(1:j)$ , так как мы знаем первые  $j$  столбцов матрицы  $L$ . Тем не менее значительно более эффективный способ для вычисления  $H(1:j, j)$  можно получить, используя  $j$ -й столбец уравнения  $H = TL^T$ . В частности, учитывая условие  $\beta(0)L(j, 0) = 0$ , мы имеем

$$h(k) = \beta(k - 1)L(j, k - 1) + \alpha(k)L(j, k) + \beta(k)L(j, k + 1)$$

для  $k = 1:j$ . Эта формула срабатывает, за исключением случая  $k = j$ , поскольку у нас еще нет вычисленных коэффициентов  $\alpha(j)$  и  $\beta(j)$ . Тем не менее, когда вектор  $h(1:j - 1)$  известен, мы можем получить  $h(j)$  из последней строки треугольной системы (4.4.8), а именно

$$h(j) = A(j, j) - \sum_{k=1}^{j-1} L(j, k)h(k).$$

Суммируя полученные результаты и используя рабочий массив  $l(1:n)$  для  $L(j, 1:j)$ , мы видим, что вычисление вектора  $h(1:j)$  из (4.4.7) может быть организовано следующим образом:

```

if  $j = 1$ 
     $h(1) = A(1, 1)$ 
elseif  $j = 2$ 
     $h(1) = \beta(1); h(2) = A(2, 2)$ 
else
     $l(0) = 0; l(1) = 0; l(2:j - 1) = L(j, 2:j - 1); l(j) = 1$ 
     $h(j) = A(j, j)$ 
    for  $k = 1:j - 1$ 
```

(4.4.9)

```

 $h(k) = \beta(k-1)l(k-1) + \alpha(k)l(k) + \beta(k)l(k+1)$ 
 $h(j) = h(j) - l(k)h(k)$ 
end
end

```

Заметим, что с учетом  $O(j)$ -метода для вычисления  $h(1:j)$ , гахру-вычисления вектора  $v(j+1:n)$  являются доминирующей операцией в (4.4.7). В течение  $j$ -го шага данная гахру содержит около  $2j(n-j)$  флопов. Суммируя по  $j = 1:n$ , покажем, что метод Азена требует  $n^3/3$  флопов. Таким образом, алгоритмы Азена и Холецкого требуют одинаковое количество арифметических операций.

#### 4.4.3. Выбор ведущего элемента в методе Азена

Как было установлено, столбцы матрицы  $L$  являются масштабированными столбцами  $v$ -векторов из (4.4.7). Если некоторые из масштабированных коэффициентов велики, т. е. если некоторые из компонент  $v(j+1)$  малы, то ситуация крайне плохая. Чтобы выбраться из нее, нам необходимо переставить наибольшую компоненту вектора  $v(j+1:n)$  в верхнюю позицию. Конечно, подобные перестановки должны быть применимы как для неприведенной части матрицы  $A$ , так и для предварительно вычисленной части матрицы  $L$ .

**Алгоритм 4.4.1 (Метод Азена).** Если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, то следующий алгоритм вычисляет матрицу перестановок  $P$ , нижнюю унитреугольную матрицу  $L$  и трехдиагональную матрицу  $T$ , такие, что  $PAP^T = LTL^T$ , где  $|l_{ij}| \leq 1$ . Матрица перестановок  $P$  задается целочисленным вектором  $piv$ . В частности,  $P = P_1 \dots P_{n-2}$ , где  $P_j$  – это единичная матрица с переставленными  $(j+1)$ -й и  $piv(j)$ -й строками. Диагональ и поддиагональ матрицы  $T$  хранятся в векторах  $\alpha(1:n)$  и  $\beta(1:n-1)$  соответственно. В матрице  $L(2:n, 2:n)$  вычисляется только поддиагональная часть.

```
for  $j = 1:n$ 
```

Вычислить  $h(1:j)$  посредством (4.4.9).

```
if  $j = 1 \vee j = 2$ 
```

$\alpha(j) = h(j)$

```
else
```

$\alpha(j) = h(j) - \beta(j-1)L(j, j-1)$

```
end
```

```
if  $j \leq n-1$ 
```

$v(j+1:n) = A(j+1:n, j) - L(j+1:n, 1:j)h(1:j)$

Найти  $q$ , такое, что  $|v(q)| = ||v(j+1:n)||_\infty$  при  $j+1 \leq q \leq n$ .

$piv(j) = q; v(j+1) \leftrightarrow v(q); L(j+1, 2:j) \leftrightarrow L(q, 2:j)$

$A(j+1, j:1:n) \leftrightarrow A(q, j+1:n)$

$A(j+1:n, j+1) \leftrightarrow A(j+1:n, q)$

$\beta(j) = v(j+1)$

```
end
```

```
if  $j \leq n-2$ 
```

$L(j+2:n, j+1) = v(j+2:n)$

**if**  $v(j+1) \neq 0$

$L(j+2:n, j+1) = L(j+2:n, j+1)/v(j+1)$

**end**

```
end
```

```
end
```

Метод Азена устойчив в том смысле, в каком устойчив метод исключения Гаусса, т. е. получается точное разложение матрицы, близкой к  $A$ , если выполнено условие  $\|\tilde{T}\|_2/\|A\|_2 \approx 1$ , где  $\tilde{T}$  – вычисленный аналог трехдиагональной матрицы  $T$ . В общем случае это почти всегда имеет место.

При практической реализации алгоритма Азена нижняя треугольная часть матрицы  $A$  замещается матрицами  $L$  и  $T$ . Возьмем случай  $n = 5$

$$A \leftarrow \begin{bmatrix} a_1 & & & & \\ \beta_1 & a_2 & & & \\ l_{32} & \beta_2 & a_3 & & \\ l_{42} & l_{43} & \beta_3 & a_4 & \\ l_{52} & l_{53} & l_{54} & \beta_4 & a_5 \end{bmatrix}.$$

Заметим, что при таком расположении столбцы матрицы  $L$  сдвинуты влево.

#### 4.4.4. Методы с диагональным выбором

Мы сейчас опишем вычисление блочного  $LDL^T$ -разложения по формуле (4.4.2). Мы следуем результатам работы Банча и Парлетта (1971). Пусть

$$P_1 A P_1^T = \begin{bmatrix} E & C^T \\ C & B \\ s & n-s \end{bmatrix}_{n-s},$$

где  $P_1$  – перестановочная матрица и  $s = 1$  или  $2$ . Если матрица  $A$  ненулевая, то всегда можно сделать выбор этих параметров таким, что  $E$  будет невырожденной, поэтому мы можем записать

$$P_1 A P_1^T = \begin{bmatrix} I_s & 0 \\ CE^{-1} & I_{n-s} \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I_s & E^{-1}C^T \\ 0 & I_{n-s} \end{bmatrix}.$$

Для достижения устойчивости  $s \times s$ -ведущий «элемент»  $E$  должен быть выбран так, что элементы матрицы

$$\tilde{A} = (\tilde{a}_{ij}) \equiv B - CE^{-1}C^T \quad (4.4.10)$$

ограничены соответствующим образом. С этой целью зададим  $\alpha \in (0, 1)$  и введем следующие обозначения:

$$\mu_0 = \max_{i,j} |a_{ij}|, \quad \mu_1 = \max_i |a_{ii}|.$$

Стратегия Банча–Парлетта для выбора ведущего элемента имеет следующий вид

```

if  $\mu_1 \geq \alpha \mu_0$ 
     $s = 1$ 
    Выбираем матрицу  $P_1$ , такую, что  $|e_{11}| = \mu_1$ .
else
     $s = 2$ 
    Выбираем матрицу  $P_1$ , такую, что  $|e_{21}| = \mu_0$ .
end

```

Легко проверить из формулы (4.4.10), что если  $s = 1$ , то

$$|\tilde{a}_{ij}| \leq (1 + \alpha^{-1})\mu_0, \quad (4.4.11)$$

если же  $s = 2$ , то

$$|\tilde{a}_{ij}| \leq \frac{3 - \alpha}{1 - \alpha} \mu_0. \quad (4.4.12)$$

Сравнивая величину  $(1 + \alpha^{-1})^2$ , фактор роста, связанный с двумя шагами метода при  $s = 1$ , и величину  $(3 - \alpha)/(1 - \alpha)$ , соответствующий фактор при  $s = 2$ , Банч и Парлетт пришли к выводу, что  $\alpha = (1 + \sqrt{17})/8$  является оптимальным с точки зрения минимизации оценки роста элементов.

Указанное выше приведение потом повторяется для симметричной матрицы  $\tilde{A}$  порядка  $n - s$ . Используя индукцию, устанавливаем, что разложение (4.4.2) существует и требует  $n^3/3$  флопов, если не учитывать работу, связанную с выбором ведущего блока.

#### 4.4.5. Устойчивость и эффективность

Как показал Банч (1971а), рассмотренный выше метод со стратегией диагонального выбора так же устойчив, как и исключение Гаусса с полным выбором. К несчастью, в целом процесс требует между  $n^3/12$  и  $n^3/6$  сравнений, так как  $\mu_0$  включает двумерный поиск на каждой стадии приведения. Действительное количество сравнений зависит от общего числа  $2 \times 2$  ведущих блоков, но, вообще, метод Банча–Парлетта для вычисления разложения (4.4.2) значительно более медленный, чем метод Аазена. Смотри Barwell и George (1976).

Метод Банча и Кауфмана (1977) – это не то же, что и диагональный выбор. В этой схеме на каждой стадии процесса приведения необходимо исследовать два столбца. Эта стратегия хорошо иллюстрируется рассмотрением самого первого шага приведения:

```

 $\alpha = (1 + \sqrt{17})/8; \quad \lambda = |a_{r1}| = \max \{|a_{21}|, \dots, |a_{n1}|\}$ 
if  $\lambda > 0$ 
  if  $|a_{11}| \geq \alpha\lambda$ 
     $s = 1; \quad P_1 = I$ 
  else
     $\sigma = |a_{pr}| = \max \{|a_{1r}|, \dots, |a_{r-1,r}|, |a_{r+1,r}|, \dots, |a_{nr}|\}$ 
    if  $\sigma|a_{11}| \geq \alpha\lambda^2$ 
       $s = 1, \quad P_1 = I$ 
    elseif  $|a_{rr}| \geq \alpha\sigma$ 
       $s = 1$  и выбрать  $P_1$  так, что  $(P_1^T A P_1)_{11} = a_{rr}$ .
    else
       $s = 2$  и выбрать  $P_1$  так, что  $(P_1^T A P_1)_{21} = a_{rp}$ .
    end
  end
end

```

В целом, алгоритм Банча–Кауфмана требует  $n^3/3$  флопов,  $O(n^2)$  сравнений и, как и все методы этого раздела,  $n^2/2$  ячеек памяти.

**Пример 4.4.2.** Если алгоритм Банча–Кауфмана применить к матрице

$$A = \begin{bmatrix} 1 & 10 & 20 \\ 10 & 1 & 30 \\ 20 & 30 & 1 \end{bmatrix},$$

то на первом шаге  $\lambda = 20$ ,  $r = 3$ ,  $\sigma = 1$  и  $p = 3$ . Матрица перестановок  $P = [e_1 \ e_3 \ e_2]$  при применении дает

$$PAP^T = \begin{bmatrix} 1 & 20 & 10 \\ 20 & 1 & 30 \\ 10 & 30 & 1 \end{bmatrix}.$$

Потом используется  $2 \times 2$ -ведущий блок, с которым выполняется приведение

$$PAP^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{590}{399} & \frac{170}{399} & 1 \end{bmatrix} \begin{bmatrix} 1 & 20 & 0 \\ 20 & 1 & 0 \\ 0 & 0 & \frac{-10601}{399} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{540}{399} & \frac{170}{399} & 1 \end{bmatrix}^T.$$

#### 4.4.6. Сравнение метода Аазена с диагональным выбором

Поскольку в процессе выбора ведущего элемента должны изучаться «будущие» столбцы, то это делает затруднительным (хотя и возможным) получение алгоритма диагонального выбора с большим содержанием гахру-операций. С другой стороны, метод Аазена по своей природе богат гахру-операциями и, возможно, предпочтительнее выбрать его для векторно-конвейерных реализаций. В конце отметим, что для обеих процедур возможны блочные варианты.

#### Задачи

**4.4.1.** Показать, что если все главные подматрицы порядков  $1 \times 1$  и  $2 \times 2$  симметричной матрицы  $A$  вырождены, то  $A$  – нулевая матрица.

**4.4.2.** Показать, что если матрица  $A$  положительно определенная, то в алгоритме Банча – Кауфмана могут выбираться ведущие блоки порядка  $2 \times 2$ .

**4.4.3.** Реализовать алгоритм 4.4.1 так, что используется только нижняя треугольная часть матрицы  $A$ , и при этом  $\alpha(j)$  замещает  $A(j, j)$  для  $j = 1:n$ ,  $\beta(j)$  замещает  $A(j+1, j)$  для  $j = 1:n-1$  и  $L(i, j)$  замещает  $A(i, j-1)$  для  $j = 2:n-1$  и  $i = j+1:n$ .

**4.4.4.** Пусть матрица  $A \in \mathbb{R}$  невырожденная и симметричная. Построить алгоритм, который вычисляет разложение

$$PAP^T = \begin{bmatrix} R & 0 \\ S & -M \end{bmatrix} \begin{bmatrix} R^T & S^T \\ 0 & M^T \end{bmatrix},$$

где  $R \in \mathbb{R}^{k \times k}$  и  $M \in \mathbb{R}^{(n-k) \times (n-k)}$  – нижние треугольные и невырожденные матрицы и  $P$  – матрица перестановок. Матрица  $A$  – диагонально доминирующая.

#### Замечания и литература к § 4.4

Основные ссылки по вычислению разложения (4.4.1) следующие

Aasen J. O. (1971). "On the Reduction of a Symmetric Matrix to Tridiagonal Form", BIT 11, 233–242.  
Parlett B. N. and Reid J. K. (1970). "On the Solution of a System of Linear Equations Whose Matrix is Symmetric but not Definite", BIT 10, 386–397.

Литература по диагональному выбору содержит работы

Bunch J. R. (1971a). "Analysis of the Diagonal Pivoting Method", SIAM J. Num. Anal. 8, 656–680.

- Bunch J. R. (1974). "Partial Pivoting Strategies for Symmetric Matrices", SIAM J. Num. Anal. 11, 521–528.
- Bunch J. R. and Kaufman L. (1977). "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems", Math. Comp. 31, 162–179.
- Bunch J. R., Kaufman L., and Partlett B. N. (1976). "Decomposition of a Symmetric Matrix", Numer. Math. 27, 95–109.
- Bunch J. R. and Partlett B. N. (1971). "Direct Methods for Solving Symmetric Indefinite Systems of Linear Equations", SIAM J. Num. Anal. 8, 639–655.

Предпоследняя ссылка содержит алгоритмический вариант метода с диагональным выбором, имеющийся в Linpack гл. 5.

Вопрос о том, имеет ли метод Азена преимущества перед алгоритмом Банча – Кауфмана, изучается в работе

- Barwell V. and George J. A. (1976). "A Comparison of Algorithms for Solving Symmetric Indefinite Systems of Linear Equations", ACM Trans. Math. Soft. 2, 242–951.

Они полагают, что оба алгоритма ведут себя одинаково на условных процессорах, возможно со слабым преимуществом метода Азена, когда  $n$  больше чем 200. Характеристики данных в этой статье являются основой для интересного статистического анализа

- Hoaglin D. (1977). "Mathematical Software and Exploratory Data Analysis", in Mathematical Software III, ed. John Rice, Academic Press, New York, pp. 139–159.

Маленькое преимущество метода Азена, возможно, вызвано более простой стратегией выбора.

Другая идея для дешевой стратегии выбора, сохраняющая границы ошибок, основана на менее жестком критерии перестановок; идея взята из области, связанной с методами разреженных исключений. См.

- Fletcher R. (1976). "Factorizing Symmetric Indefinite Matrices", Lin. Alg. and Its Applic. 14, 257–272.

Мы также упомянем статью

- Dax A. and Kaniel S. (1977). "Pivoting Techniques for Symmetric Gaussian Elimination", Numer. Math. 28, 221–242,

в которой диагональные ведущие элементы «строются», если это необходимо, при помощи треугольных матричных множителей. К несчастью, метод может потребовать  $O(n^3)$  сравнений.

Перед решением симметричной системы  $Ax = b$  возможна ситуация, когда матрицу  $A$  желательно уравновесить. Алгоритм сложности  $O(n^2)$ , выполняющий эту задачу, дан в работе

- Bunch J. R. (1971b). "Equilibration of Symmetric Matrices in the Max-Norm", J. ACM 18, 566–572.

Наконец, напомним, что методы, аналогичные методам этого раздела, существуют также и для кососимметричных систем, см.

- Bunch J. R. (1982). "A Note on the Stable Decomposition of Skew Symmetric Matrices", Math. Comp. 158, 475–480.

## 4.5. Блочные трехдиагональные системы

Во многих областях приложения возникают матрицы, имеющие блочную структуру, которую можно эффективно использовать. Например, в задачах оптимизации с ограничениями часто требуется решать линейные системы вида

$$\begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}, \quad (4.5.1)$$

где  $A$  – симметричная положительно определенная матрица, а  $B$  – матрица полного столбцового ранга. В данной ситуации имеет смысл использовать структуру системы, а не трактовать (4.5.1) как «еще одну» симметричную неопределенную систему. Для ознакомления с деталями см. Heath (1978).

Чтобы изучить проблему, как именно можно использовать блочную структуру, выберем для рассмотрения решение блочных трехдиагональных систем вида

$$\begin{bmatrix} D_1 & F_1 & & \cdots & 0 \\ E_1 & D_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ & & & F_{n-1} & \\ 0 & \cdots & \ddots & E_{n-1} & D_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}. \quad (4.5.2)$$

Здесь мы считаем, что все блоки имеют размер  $q \times q$  и что  $x_i$  и  $b_i$  принадлежат  $\mathbb{R}^q$ . В этом разделе мы рассмотрим два подхода к этой задаче – блочное  $LU$ -разложение и уступающую ему схему, известную как *циклическая редукция*. Другие аспекты, связанные с блочными трехдиагональными системами, обсуждаются в § 10.3.3.

#### 4.5.1. Блочное $LU$ -разложение

Мы начнем с рассмотрения блочного  $LU$ -разложения для матрицы (4.5.2). Определим блочные трехдиагональные матрицы  $A_k$  следующим образом:

$$A_k = \begin{bmatrix} D_1 & F_1 & & \cdots & 0 \\ E_1 & D_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ & & & F_{k-1} & \\ 0 & \cdots & \ddots & E_{k-1} & D_k \end{bmatrix}, \quad k = 1:n. \quad (4.5.3)$$

Сравнивая блоки в представлении

$$A = \begin{bmatrix} I & & \cdots & 0 \\ L_1 & I & & \vdots \\ \vdots & \ddots & & \vdots \\ 0 & \cdots & L_{n-1} & I \end{bmatrix} \begin{bmatrix} U_1 & F_1 & & \cdots & 0 \\ E_1 & U_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & \vdots \\ 0 & \cdots & \ddots & E_{k-1} & U_n \end{bmatrix}, \quad (4.5.4)$$

мы формально получаем следующий алгоритм для вычислений  $L_i$  и  $U_i$ :

$U_1 = D_1$   
for  $i = 2:n$

Решить  $L_{i-1} U_{i-1} = E_{i-1}$  относительно  $L_{i-1}$ .

$U_i = D_i - L_{i-1} F_{i-1}$

end

(4.5.5)

Процедура определена до тех пор, пока  $U_i$  невырождены. Это гарантировано, если, например, матрицы  $A_1, \dots, A_n$  невырождены.

Имея вычисленное разложение (4.5.4), мы можем получить вектор  $x$  из (4.5.2)

посредством блочной прямой и обратной подстановки:

```

 $y_1 = b_1$ 
for  $i = 2:n$ 
 $y_i = b_i - L_{i-1} y_{i-1}$ 
end
Решить  $U_n x_n = y_n$  относительно  $x_n$ .
for  $i = n-1:-1:1$ 
Решить  $U_i x_i = y_i - F_i x_{i+1}$  относительно  $x_i$ 
end
```

(4.5.6)

При выполнении алгоритмов (4.5.5) и (4.5.6), для каждого блока  $U_i$  требуется разложение, так как решаются линейные системы, содержащие эти подматрицы. Это может быть сделано при помощи исключения Гаусса с выбором. Однако это не гарантирует устойчивость процесса в целом. Чтобы понять это, рассмотрите случай, когда размер блока равен единице.

## 4.5.2. Блочное диагональное доминирование

Чтобы получить удовлетворительные границы для  $L_i$  и  $U_i$ , необходимо сделать дополнительные предположения о блочных матрицах, которые встречаются ниже. Например, если для  $i = 1:n$  у нас выполнено отношение диагонального блочного доминирования

$$\|D_i^{-1}\|_1 (\|F_{i-1}\|_1 + \|E_i\|_1) < 1, \quad E_n \equiv F_0 \equiv 0, \quad (4.5.7)$$

то разложение (4.5.3) существует и можно показать, что матрицы  $L_i$  и  $U_i$  удовлетворяют неравенствам

$$\|L_i\|_1 \leq 1, \quad (4.5.8)$$

$$\|U_i\|_1 \leq \|A\|_1. \quad (4.5.9)$$

## 4.5.3. Сравнение блочного и ленточного решений

Сейчас разумно спросить, почему мы не можем просто рассматривать матрицу  $A$  из (4.5.2) как матрицу  $qn \times qn$ , имеющую скалярные элементы с шириной ленты  $2q - 1$ . Тогда можно было бы применить ленточное исключение Гаусса, описанное в § 4.3. Эффективность такого подхода зависит от таких вещей, как размеры блоков и структура разреженности каждого блока.

Чтобы проиллюстрировать это на очень простом примере, предположим, что мы хотим решить систему

$$\begin{bmatrix} D_1 & F_1 \\ E_1 & D_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad (4.5.10)$$

где блоки  $D_1$  и  $D_2$  – диагональные, а  $F_1$  и  $E_1$  – трехдиагональные. Допустим, что каждый из этих блоков размера  $n \times n$  и что решать задачу (4.5.10) при помощи алгоритмов (4.5.4) и (4.5.6) «безопасно». Заметим, что

$$\begin{aligned}
 U_1 &= D_1, && \text{(диагональная матрица)} \\
 L_1 &= E_1 U_1^{-1}, && \text{(трехдиагональная)} \\
 U_2 &= D_2 - L_1 F_1, && \text{(пятидиагональная)} \\
 y_1 &= b_1, \\
 y_2 &= b_2 - E_1 (D_1^{-1} y_1), \\
 U_2 x_2 &= y_2, \\
 D_1 x_1 &= y_1 - F_1 x_2.
 \end{aligned}$$

Следовательно, некоторые очень простые  $n \times n$  вычисления с исходными ленточными блоками дают решение.

С другой стороны, простое применение ленточного исключения Гаусса к системе (4.5.10) вызвало бы много необязательной работы и хранения, как для системы с шириной ленты  $n + 1$ . Тем не менее заметим, что переставляя строки и столбцы в системе, при помощи перестановок

$$P = [e_1, e_{n+1}, e_2, \dots, e_n, e_{2n}] \quad (4.5.11)$$

(для случая  $n = 5$ ) мы найдем, что

$$PAP^T = \begin{bmatrix} x & x & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & x & 0 & 0 & 0 & 0 \\ x & 0 & x & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & x & 0 & x & 0 & 0 \\ 0 & 0 & x & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & x & 0 & x \\ 0 & 0 & 0 & 0 & x & 0 & x & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & x & x \end{bmatrix}.$$

Эта матрица имеет ширину ленты три, и поэтому вполне приемлемая процедура решения для этой переставленной матрицы – это ленточное исключение Гаусса.

Тема перестановок для изменения ширины ленты является важной. См. George и Liu (1981, гл. 4). Мы отсылаем также читателя к работам Varah (1972) и George (1974) для дальнейшего детального изучения решения блочных трехдиагональных систем.

#### 4.5.4. Блочная циклическая редукция

Теперь мы опишем метод *блочной циклической редукции*, который может быть использован для решения некоторых важных специальных видов блочной трехдиагональной системы (4.5.2). Для простоты допускаем, что матрица  $A$  имеет вид

$$A = \begin{bmatrix} D & F & & \cdots & 0 \\ F & D & & & \vdots \\ & \ddots & \ddots & \ddots & \vdots \\ & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & F \\ 0 & \cdots & & F & D \end{bmatrix} \in \mathbb{R}^{nq \times nq}, \quad (4.5.12)$$

где  $F$  и  $D$  – матрицы  $q \times q$ , удовлетворяющие условию  $DF = FD$ . Предположим также, что  $n = 2^k - 1$ . Эти условия имеют место в некоторых важных приложениях, таких, как дискретизация уравнения Пуассона в прямоугольнике. В этом случае

$$D = \begin{bmatrix} 4 & -1 & & \dots & 0 \\ -1 & 4 & & & \vdots \\ & & \ddots & & \vdots \\ & & \vdots & \ddots & \vdots \\ & & \vdots & & -1 \\ & & 0 & \dots & -1 & 4 \end{bmatrix} \quad (4.5.13)$$

и  $F = -I_q$ . Целое число  $n$  определяется размером сетки и часто может быть выбрано таким, что  $n = 2^k - 1$ . (Sweet (1977) показал, что делать, если размер не удовлетворяет такому условию.)

Основная идея циклической редукции – уменьшать вдвое размерность задачи, повторяя это неоднократно, пока у нас не останется простая  $q \times q$  система относительно неизвестного подвектора  $x_{2^{k-1}}$ . Эта система потом решается стандартными способами. Ранее исключенные  $x_i$  находятся в процессе обратной подстановки.

Для понимания общей процедуры достаточно рассмотреть случай  $n = 7$ :

$$\begin{aligned} b_1 &= Dx_1 + Fx_2, \\ b_2 &= Fx_1 + Dx_2 + Fx_3, \\ b_3 &= Fx_2 + Dx_3 + Fx_4, \\ b_4 &= Fx_3 + Dx_4 + Fx_5, \\ b_5 &= Fx_4 + Dx_5 + Fx_6, \\ b_6 &= Fx_5 + Dx_6 + Fx_7, \\ b_7 &= Fx_6 + Dx_7. \end{aligned} \quad (4.5.14)$$

Для  $i = 2, 4$  и  $6$  мы умножим уравнения  $i-1$ ,  $i$  и  $i+1$  на матрицы  $F$ ,  $-D$  и  $F$  соответственно, и, складывая результат, получим

$$\begin{aligned} (2F^2 - D^2)x_2 + F^2x_4 &= F(b_1 + b_3) - Db_2, \\ F^2x_2 + (2F^2 - D^2)x_4 + F^2x_6 &= F(b_3 + b_5) - Db_4, \\ F^2x_4 + (2F^2 - D^2)x_6 &= F(b_5 + b_7) - Db_6. \end{aligned}$$

Таким образом, следуя этой тактике, мы удалим все  $x$  с нечетными индексами и у нас останется уменьшенная блочная трехдиагональная система вида

$$\begin{aligned} D^{(1)}x_2 + F^{(1)}x_4 &= b_2^{(1)}, \\ F^{(1)}x_2 + D^{(1)}x_4 + F^{(1)}x_6 &= b_4^{(1)}, \\ F^{(1)}x_4 + D^{(1)}x_6 &= b_6^{(1)}. \end{aligned}$$

где матрицы  $D^{(1)} = 2F^2 - D^2$  и  $F^{(1)} = F^2$  являются перестановочными. Применяя такую же стратегию исключения, как и выше, мы умножим эти три уравнения соответственно на  $F^{(1)}$ ,  $-D^{(1)}$  и  $F^{(1)}$ . Если эти преобразованные уравнения сложить вместе, мы получим простое уравнение

$$(2[F^{(1)}]^2 - D^{(1)2})x_4 = F^{(1)}(b_2^{(1)} + b_6^{(1)}) - D^{(1)}b_4^{(1)},$$

которое запишем в виде

$$D^{(2)}x_4 = b^{(2)}.$$

Это полная циклическая редукция. Мы решим эту (маленькую) систему  $q \times q$  относительно  $x_4$ . Векторы  $x_2$  и  $x_6$  находятся потом решением систем

$$D^{(1)}x_2 = b_2^{(1)} - F^{(1)}x_4, \quad D^{(1)}x_6 = b_6^{(1)} - F^{(1)}x_4.$$

В заключение мы используем первое, третье, пятое и седьмое уравнения из (4.5.14) для вычисления  $x_1, x_3, x_5$  и  $x_7$  соответственно.

Для произвольного  $n$  вида  $n = 2^k - 1$  положим  $D^{(0)} = D, F^{(0)} = F, b^{(0)} = b$  и вычислим:

```

for  $p = 1:k-1$ 
   $D^{(p)} = 2[F^{(p-1)}]^2 - [D^{(p-1)}]^2$ 
   $F^{(p)} = [F^{(p-1)}]^2$ 
   $r = 2^p$ 
  for  $j = 1:2^{k-p}-1$ 
     $b_{jr}^{(p)} = F^{(p-1)}(b_{jr-r/2}^{(p-1)} + b_{jr+r/2}^{(p-1)}) - D^{(p-1)}b_{jr}^{(p-1)}$ 
  end
end

```

Элементы  $x_i$  затем вычисляются следующим образом:

Решить  $D^{(k-1)}x_{2^{k-1}} = b_1^{(k-1)}$  относительно  $x_{2^{k-1}}$ .

```

for  $p = k-2:-1:0$ 
   $r = 2^p$ 
  for  $j = 1:2^{k-p-1}$ 
    if  $j = 1$ 
       $c = b_{(2j-1)r}^{(p)} - F^{(p)}x_{2jr}$ 
    elseif  $j = 2^{k-p+1}$ 
       $c = b_{(2j-1)r}^{(p)} - F^{(p)}x_{(2j-2)r}$ 
    else
       $c = b_{(2j-1)r}^{(p)} - F^{(p)}(x_{2jr} + x_{(2j-2)r})$ 
    end
    Решить  $D^{(p)}x_{(2j-1)r} = c$  относительно  $x_{(2j-1)r}$ .
  end
end

```

Объем работы, требуемой для реализации этих рекурсий, сильно зависит от разреженности блоков  $D^{(p)}$  и  $F^{(p)}$ . В худшем случае, когда эти матрицы полные, общий объем флопов имеет порядок  $\log(n)q^3$ . При выполнении необходимо быть внимательным, чтобы обеспечить устойчивость процесса. Для дальнейшего изучения см. Buneman (1969).

**Пример 4.5.1.** Пусть  $q = 1, D = (4)$  и  $F = (-1)$  в (4.5.14), и мы хотим решить систему:

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \\ 12 \\ 22 \end{bmatrix}.$$

Выполняя (4.5.15), мы получим следующую редуцированную систему:

$$\begin{bmatrix} -14 & 1 & 0 \\ 1 & -14 & 1 \\ 0 & 1 & -14 \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ x_6 \end{bmatrix} = \begin{bmatrix} -24 \\ -48 \\ -80 \end{bmatrix}, p = 1,$$

$$[-194] = [x_4] [-776], p = 2.$$

Элементы  $x_i$  потом получаются через (4.5.16):

$$\begin{aligned} p = 2: \quad & x_4 = 4, \\ p = 1: \quad & x_2 = 2, \quad x_6 = 6, \\ p = 0: \quad & x_1 = 1, \quad x_3 = 3, \quad x_5 = 5, \quad x_7 = 7. \end{aligned}$$

Циклическая редукция является примером алгоритма типа разделяй и властвуй. Другие процедуры типа разделяй и властвуй обсуждаются в § 1.3.8 и § 8.6.

### Задачи

- 4.5.1. Показать, что матрицы с блочным диагональным доминированием невырожденны.
- 4.5.2. Проверить, что из условия (4.5.7) вытекают соотношения (4.5.8) и (4.5.9).
- 4.5.3. Пусть применяется блочная циклическая редукция, причем  $D$  задается формулой (4.5.13), а  $F = -I_q$ . Что можно сказать о ленточной структуре вычисляемых матриц  $F^{(p)}$  и  $D^{(p)}$ ?

### Замечания и литература к § 4.5

Обсуждение симметричных неопределенных систем, упомянутых в начале раздела, может быть найдено в работе

Heath M. T. (1978). "Numerical Algorithms for Nonlinearly Constrained Optimization", Report STAN-CS 78-656, Department of Computer Science, Stanford University, (Ph. D. thesis), Stanford, California.

Следующие статьи позволяют разобраться в нюансах различных блочных матричных вычислений:

- Fourer R. (1984). "Staircase Matrices and Systems", SIAM Review 26, 1–71.  
 George J. A. (1974). "On Block Elimination for Sparse Linear Systems", SIAM J. Num. Anal. 11, 585–603.  
 George J. A. and Liu J. W. (1981). Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, New Jersey.  
 Merriam M. L. (1985). "On the Factorization of Block Tridiagonals With Storage Constraints", SIAM J. Sci. and Stat. Comp. 6, 182–192.  
 Varah J. M. (1972). "On the Solution of Block-Tridiagonal Systems Arising from Certain Finite-Difference Equations", Math. Comp. 26, 859–868.

Свойство блочного диагонального доминирования и его различные аспекты являются центральной темой работы

Feingold D. G. and Varga R. S. (1962). "Block Diagonally Dominant Matrices and Generalizations of the Gershgorin Circle Theorem", Pacific. J. Math. 12, 1241–1250.

Ранние методы, которые содержат идею циклической редукции, описаны в статьях

- Buzbee B. L., Golub G. H., and Nielson C. W. (1970). "On Direct Methods for Solving Poisson's Equations", SIAM J. Numer. Anal. 7, 627–656.  
 Hockney R. W. (1965). "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis", J. ACM 12, 95–113.

Накопление правой части должно быть выполнено с большой тщательностью, в противном случае мы получим существенно худшую точность. Устойчивый способ выполнения этой процедуры описан в работе

Buneman O. (1969). "A Compact Non-Iterative Poisson Solver", Report 294, Stanford University Institute for Plasma Research, Stanford, California.

Другая литература, посвященная циклической редукции, содержится ниже:

Buzbee B. L., Dorr F. W., George J. A., and Golub G. H. (1971). "The Direct Solution of the Discrete Poisson Equation on Irregular Regions", SIAM J. Num. Anal. 8, 722–736.

- Buzbee B. L., and Dorr F. W. (1974). "The Direct Solution of the Biharmonic Equation on Rectangular Regions and the Poisson Equation on Irregular Regions", SIAM J. Num. Anal. 11, 753–763.
- Concus P. and Golub G. H. (1973). "Use of Fast Direct Methods for the Efficient Numerical Solution of Nonseparable Elliptic Equations", SIAM J. Num. Anal. 10, 1103–1120.
- Dorr F. W. (1970). "The Direct Solution of the Discrete Poisson Equation on a Rectangle", SIAM Review 12, 248–263.
- Dorr F. W. (1973). "The Direct Solution of the Discrete Poisson Equation in  $O(n^2)$  Operations", SIAM Review 15, 412–415.
- Heller D. (1976). "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems", SIAM J. Num. Anal. 13, 484–496.

Различные обобщения и расширения циклической редукции могут быть предложены для решения задачи с нерегулярными границами:

- Diamond M. A. and Ferreira D. L. V. (1976). "On a Cyclic Reduction Method for the Solution of Poisson's Equation", SIAM J. Num. Anal. 13, 54–70,

задачи произвольной размерности:

- Sweet R. A. (1974). "A Generalized Cyclic Reduction Algorithm", SIAM J. Num. Anal. 11, 506–520.

- Sweet R. A. (1977). "A Cyclic Reduction Algorithm for Solving Block Tridiagonal Systems of Arbitrary Dimension", SIAM J. Num. Anal. 14, 706–720,

и задачи с периодическими условиями на концах:

- Swarztrauber P. N. and Sweet R. A. (1973). "The Direct Solution of the Discrete Poisson Equation on a Disk", SIAM J. Num. Anal. 10, 900–907.

Для определенных матриц, возникающих в связи с эллиптическими дифференциальными уравнениями в частных производных, блочное исключение до некоторой степени соответствует естественным операциям на основной сетке. Классическим примером является метод гнездового сечения, описанный в работе

- George A. (1973). "Nested Dissection of a Regular Finite Element Mesh", SIAM J. Num. Anal. 10, 345–363.

В заключение упомянем общий обзор

- Bunch J. R. (1976). "Block Methods for Solving Sparse Linear Systems", in Sparse Matrix Computations, ed. J. R. Bunch and D. J. Rose, Academic Press, New York.

## 4.6. Системы Вандермонда

Пусть вектор  $x(0:n) \in \mathbf{R}^{n+1}$ . Матрица  $V \in \mathbf{R}^{(n+1) \times (n+1)}$  вида

$$V = V(x_0, \dots, x_n) = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_0 & x_1 & \dots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \dots & x_n^n \end{bmatrix}$$

называется *матрицей Вандермонда*. В этом разделе мы покажем, как можно решить системы  $V^T a = f = f(0:n)$  и  $Vz = b = b(0:n)$  за  $O(n^2)$  флопов. Для удобства нижний индекс элементов векторов и матриц начинается с 0.

### 4.6.1. Интерполяционный многочлен $V^T a = f$

Системы Вандермонда возникают во многих задачах аппроксимации и интерполяции. Действительно, ключом к получению быстрого метода решения системы Вандермонда является осознание того факта, что решение  $V^T a = f$  эквивалентно построению интерполяционного многочлена. Это следует из того, что если  $V^T a = f$  и

$$p(x) = \sum_{j=0}^n a_j x^j, \quad (4.6.1)$$

то  $p(x_i) = f_i$  для  $i = 0 : n$ .

Напомним, что если компоненты  $x_i$  различны, тогда существует единственный многочлен степени  $n$ , который интерполирует узлы  $(x_0, f_0), \dots, (x_n, f_n)$ . Соответственно матрица  $V$  невырождена тогда и только тогда, когда  $x_i$  различны. Считаем, что это условие выполнено на протяжении всего раздела.

Первый шаг для вычисления  $a_j$  из (4.6.1) состоит в определении представления Ньютона интерполяционного многочлена  $p$ :

$$p(x) = \sum_{k=0}^n c_k \prod_{i=0}^{k-1} (x - x_i). \quad (4.6.2)$$

Константы  $c_k$  – это разделенные разности, они могут быть определены следующим образом:

```

 $c(0:n) = f(0:n)$ 
for  $k = 0:n - 1$ 
  for  $i = n:-1:k + 1$ 
     $c_i = (c_i - c_{i-1})/(x_i - x_{i-k-1})$ 
  end
end

```

См. Conte и de Booz (1980, гл. 2).

Сейчас рассмотрим задачу получения  $a(0:n)$  из  $c(0:n)$ . Обозначим через  $p_n(x), \dots, p_0(x)$  многочлены, полученные при помощи итерационной формулы

```

 $p_n(x) = c_n$ 
for  $k = n-1:-1:0$ 
   $p_k(x) = c_k + (x - x_k)p_{k+1}(x)$ 
end

```

и отметим, что  $p_0(x) = p(x)$ . Выпишем

$$p_k(x) = a_k^{(k)} + a_{k+1}^{(k)} x + \dots + a_n^{(k)} x^{n-k}$$

и приравняем одинаковые степени  $x$  в уравнении  $p_k = c_k + (x - x_k)p_{k+1}$ , это дает следующую рекурсию для коэффициентов  $a_l^{(k)}$ :

```

 $a_n^{(n)} = c_n$ 
for  $k = n-1:-1:0$ 
   $a_k^{(k)} = c_k - x_k a_{k+1}^{(k+1)}$ 
  for  $i = k+1:n-1$ 
     $a_i^{(k)} = a_i^{(k+1)} - x_k a_{i+1}^{(k+1)}$ 
  end
   $a_n^{(k)} = a_n^{(k+1)}$ 
end

```

Соответственно коэффициенты  $a_i = a_i^{(0)}$  могут быть вычислены следующим образом:

```

 $a(0:n) = c(0:n)$ 
for  $k = n - 1 : -1 : 0$  (4.6.4)
    for  $i = k : n - 1$ 
         $a_i = a_i - x_k a_{i+1}$ 
    end
end

```

Комбинирование этого итерационного процесса с (4.6.3) приводит к следующему алгоритму:

**Алгоритм 4.6.1.** Для данных вектора  $x(0:n) \in \mathbf{R}^{n+1}$  с различными элементами и вектора  $f(0:n) \in \mathbf{R}^{n+1}$  следующий алгоритм замещает  $f$  на решение  $a = a(0:n)$  системы Вандермонда  $V(x_0, \dots, x_n)^T a = f$ .

```

for  $k = 0 : n - 1$ 
    for  $i = n : -1 : k + 1$ 
         $f(i) = (f(i) - f(i - 1)) / (x(i) - x(i - k - 1))$ 
    end
end
for  $k = n - 1 : -1 : 0$ 
    for  $i = k : n - 1$ 
         $f(i) = f(i) - f(i + 1) x(k)$ 
    end
end

```

Алгоритм требует  $5n^2/2$  флопов.

**Пример 4.6.1.** Пусть алгоритм 4.6.1 используется для решения системы

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix}^T \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 26 \\ 58 \\ 112 \end{bmatrix}.$$

Первый  $k$ -цикл вычисляет представление Ньютона для  $p(x)$ :

$$p(x) = 10 + 16(x - 1) + 8(x - 1)(x - 2) + (x - 1)(x - 2)(x - 3).$$

Второй  $k$ -цикл вычисляет  $a = (4 \ 3 \ 2 \ 1)^T$ , исходя из вектора  $(10 \ 16 \ 8 \ 1)^T$ .

## 4.6.2. Система $Vz = b$

Сейчас рассмотрим систему  $Vz = b$ . Чтобы получить эффективный алгоритм для этой задачи, мы опишем что делает алгоритм 4.6.1 на матрично-векторном языке. Обозначим нижнюю двухдиагональную матрицу  $L_k(a) \in \mathbf{R}^{(n+1) \times (n+1)}$  через

$$L_k(a) = \left[ \begin{array}{cccc|c} I_k & & & & 0 \\ & 1 & & \cdots & & 0 \\ & -\alpha & 1 & & & \\ & & \vdots & \ddots & & \vdots \\ 0 & . & & \ddots & & . \\ & . & & . & & . \\ & 0 & & \cdots & & 1 \\ & & & & -\alpha & 1 \end{array} \right]$$

и диагональную матрицу  $D_k$  через

$$D_k = \text{diag}(\underbrace{1, \dots, 1}_{k+1}, x_{k+1} - x_0, \dots, x_n - x_{n-k-1}).$$

С учетом этих обозначений легко убедиться исходя из (4.6.3), что если  $f = f(0:n)$  и  $c = c(0:n)$  – это вектор разделенных разностей, тогда  $c = U^T f$ , где  $U$  – это верхняя треугольная матрица вида

$$U^T = D_{n-1}^{-1} L_{n-1}(1) \dots D_0^{-1} L_0(1).$$

Аналогично из (4.6.4) мы имеем

$$a = L^T c,$$

где  $L$  – это нижняя унитреугольная матрица вида:

$$L^T = L_0(x_0)^T \dots L_{n-1}(x_{n-1})^T.$$

Таким образом,  $a = L^T U^T f$ , где  $V^{-T} = L^T U^T$ . Другими словами, алгоритм 4.6.1 решает систему  $V^T a = f$  при помощи неявного вычисления "UL"-разложения матрицы  $V^{-1}$ .

Соответственно решение системы  $Vz = b$  задается соотношением

$$z = V^{-1}b = U(Lb) = (L_0(1)^T D_0^{-1} \dots L_{n-1}(1)^T D_{n-1}^{-1})(L_{n-1}(x_{n-1}) \dots L_0(x_0)b).$$

Это наблюдение вызывает следующий алгоритм:

**Алгоритм 4.6.2.** Для данных вектора  $x(0:n) \in \mathbb{R}^{n+1}$  с различными элементами и вектора  $b = b(0:n) \in \mathbb{R}^{n+1}$  следующий алгоритм замещает  $b$  на решение  $z = z(0:n)$  системы Вандермонда  $V(x_0, \dots, x_n)z = b$ .

```

for  $k = 0:n - 1$ 
  for  $i = n:-1:k + 1$ 
     $b(i) = b(i) - x(k)b(i - 1)$ 
  end
end
for  $k = n - 1:-1:0$ 
  for  $i = k + 1:n$ 
     $b(i) = b(i)/(x(i) - x(i - k - 1))$ 
  end
  for  $i = k:n - 1$ 
     $b(i) = b(i) - b(i + 1)$ 
  end
end

```

Алгоритм требует  $5n^2/2$  флопов.

**Пример 4.6.2.** Пусть алгоритм 4.6.2 используется для решения системы

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix} \begin{bmatrix} z0 \\ z1 \\ z2 \\ z3 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 3 \\ 35 \end{bmatrix}.$$

Первый  $k$ -цикл вычисляет вектор

$$L_3(3)L_2(2)L_1(1) \begin{bmatrix} 0 \\ -1 \\ 3 \\ 35 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 6 \\ 6 \end{bmatrix}.$$

Второй  $k$ -цикл потом вычисляет

$$L_0(1)^T D_0^{-1} L_1(1)^T D_1^{-1} L_2(1)^T D_2^{-1} \begin{bmatrix} 0 \\ -1 \\ 3 \\ 35 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ 0 \\ 1 \end{bmatrix}.$$

### 4.6.3. Устойчивость

Алгоритмы 4.6.1 и 4.6.2 рассмотрены и проанализированы в работе Björck и Pereyra (1970). Их вывод состоит в том, что эти алгоритмы часто получают удивительно точные решения, даже когда матрица  $V$  плохо обусловлена. Они также показали, как модифицировать решение, когда добавляется новая координатная пара  $(x_{n+1}, f_{n+1})$  к множеству узлов интерполяции, и как решать *конфлюэнтные системы Вандермонда*, т. е. системы с матрицами вида

$$V = V(x_0, x_1, x_2, x_3) = \begin{bmatrix} 1 & 1 & 0 & 1 \\ x_0 & x_1 & 1 & x_3 \\ x_0^2 & x_1^2 & 2x_1^2 & x_3^2 \\ x_0^3 & x_1^3 & 3x_1^2 & x_3^3 \end{bmatrix}.$$

#### Задачи

**4.6.1.** Показать, что если  $V = V(x_0, \dots, x_n)$ , то

$$\det(V) = \prod_{n \geq i > j \geq 0} (x_i - x_j).$$

**4.6.2.** (Gautschi (1975a)). Проверить следующее неравенство для случая  $n = 1$ :

$$|V^{-1}|_\infty \leq \max_{0 \leq k \leq n} \prod_{\substack{i=0 \\ i \neq k}}^n \frac{1 + |x_i|}{|x_k - x_i|}.$$

Равенство достигается, если все  $x_i$  лежат на одном луче комплексной плоскости.

#### Замечания и литература к § 4.6

Наши рассуждения по линейным системам Вандермонда выведены из статей

Björck A. and Pereyra V. (1970). "Solution of Vandermonde Systems of Equations", Math. Comp. 24, 893–903.

Björck A. and Elfving T. (1973). "Algorithms for Confluent Vandermonde Systems", Numer. Math. 21, 130–137.

Последняя ссылка содержит алгол-процедуру. Более глубокий анализ решения систем Вандермонда может быть найден в работах

Higham N.J. (1987b). "Error Analysis of the Björck–Pereyra Algorithms for Solving Vandermonde Systems", Numer. Math. 50, 613–632.

Higham N.J. (1988a). "Fast Solution of Vandermonde-like Systems Involving Orthogonal Polynomials", IMA J. Numer. Anal. 8, 473–486.

См. также

Galimberti G. and Pereyra V. (1970). "Numerical Differentiation and the Solution of Multidimensional Vandermonde Systems", Math. Comp. 24, 357–364.

Galimberti G. and Pereyra V. (1971). "Solving Confluent Vandermonde Systems of Hermitian Type", Numer. Math. 18, 44–60.

Van de Vel H. (1977). "Numerical Treatment of a Generalized Vandermonde Systems of Equations", Lin. Alg. and Its Applic. 17, 149–174.

Интересные результаты, имеющие отношение к обусловленности систем Вандермонда могут быть найдены в работах

Gautschi W. (1975a). "Norm Estimates for Inverses of Vandermonde Matrices", Numer. Math. 23, 337–347.

Gautschi W. (1975b). "Optimally Conditioned Vandermonde Matrices", Numer. Math. 24, 1–12.

В работе

Golub G. H. and Tang W. P. (1981). "The Block Decomposition of a Vandermonde Matrix and Its Applications", BIT 21, 505–517.

приводится блочный алгоритм Вандермонда, который дает возможность обойти комплексную арифметику в некоторых задачах интерполяции.

Вычисление разделенных разностей, которое мы обсуждали, дается подробно в гл. 2 работы

Conte S. D. and de Boor C. (1980). Elementary Numerical Analysis: An Algorithmic Approach, 3rd ed., McGraw-Hill, New York.

#### Добавления при переводе к § 4.6

Важным частным случаем матрицы Вандермонда является *матрица Фурье*

$$F_n = \left[ \exp\left(i \frac{2\pi}{n} kl\right) \right]_{kl=0}^{n-1}$$

( $i$  – мнимая единица). К числу основополагающих работ по быстрым алгоритмам умножения  $F_n$  на вектор относится

Cooley J. W., Tukey J. W (1965). "An Algorithm for the Machine Computation of Complex Fourier Series", Math. Comp. 19, 297–301.

Литература, посвященная быстрому преобразованию Фурье и его применением, весьма обширна. См., например,

Тыртышников Е. Е. (1981). Об алгоритмах дискретного преобразования Фурье. Численные методы алгебры.–М.: Изд-во Моск. ун-та.

Нечепуренко Ю. М. (1985). Быстрые алгоритмы умножения на вектор матрицы Вандермонда. Архитектура ЭВМ и численные методы.–М.: ОВМ АН СССР.

Воеводин В. В., Тыртышников Е. Е. (1987). Вычислительные процессы с тёплицевыми матрицами.–М.: Наука (особенно § 4).

Блейхут Р. (1989). Быстрые алгоритмы цифровой обработки сигналов.–М.: Мир.

Nechepurenko Yu. M. and Tyrtysnikov E. E. (1990). "Multi-Dimensional Discrete Fourier Transform and Its Optimization", Sov. J. Numer. Math. Modelling, 5, N 3.

Асимптотически оптимальные приближенные алгоритмы умножения для вещественных матриц Вандермонда описаны в

Нечепуренко Ю. М. (1987). Асимптотически оптимальные алгоритмы умножения для матрицы экспонент. Архитектура ЭВМ и численные методы.–М.: ОВМ АН СССР, с. 26–34.

## 4.7. Тёплицевы системы

Матрицы, элементы которых равны вдоль каждой диагонали, возникают во многих приложениях и называются *тёплицевыми матрицами*. Формально матрица  $T \in \mathbb{R}^{n \times n}$  является тёплицевой, если существуют скаляры  $r_{-n+1}, \dots, r_0, \dots, r_{n-1}$ , такие, что  $a_{ij} = r_{j-i}$  для всех  $i$  и  $j$ . Таким образом, матрица

$$T = \begin{bmatrix} r_0 & r_1 & r_2 & r_3 \\ r_{-1} & r_0 & r_1 & r_2 \\ r_{-2} & r_{-1} & r_0 & r_1 \\ r_{-3} & r_{-2} & r_{-1} & r_0 \end{bmatrix}$$

является тёплицевой.

Тёплицевые матрицы относятся к обширному классу *персимметричных матриц*. Говорят, что матрица  $B \in \mathbb{R}^{n \times n}$  является персимметричной, если она симметрична относительно северо-восточной – юго-западной диагонали, т. е.  $b_{ij} = b_{n-j+1, n-i+1}$  для всех  $i$  и  $j$ . Это эквивалентно требованию  $B = EB^TE$ , где

$$E = [e_n \dots, e_1] = I_n(:, n:-1:1)$$

является  $n \times n$ -матрицей *перестановок*. Легко проверить, что (a) тёплицевые матрицы – персимметричные и (b) обратная к невырожденной тёплицевой матрице является персимметричной. В этом разделе мы покажем, как правильное использование свойства (b) поможет нам решать тёплицевые системы за время  $O(n^2)$ . Обсуждение ограничивается важным случаем, когда матрица  $T$  симметричная и положительно определенная.

#### 4.7.1. Три задачи

Допустим, что у нас имеются скаляры  $r_1, \dots, r_n$ , такие, что для  $k = 1:n$  матрицы

$$T_k = \begin{bmatrix} 1 & r_1 & \dots & r_{k-2} & r_{k-1} \\ r_1 & 1 & & & r_{k-2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ r_{k-2} & & \ddots & \ddots & r_1 \\ r_{k-1} & r_{k-2} & \dots & r_1 & 1 \end{bmatrix}$$

положительно определенные. (Мы не ограничиваем общности нормализацией диагонали.) В этом разделе описаны три алгоритма:

- Алгоритм Дурбина для задачи Юла–Уолкера  $T_n y = -(r_1, \dots, r_n)^T$ .
- Алгоритм Левинсона для задачи с произвольной правой частью  $T_n x = b$ .
- Алгоритм Тренча для вычисления  $B = T_n^{-1}$ .

При выводе этих методов обозначим через  $E_k$  перестановочную  $k \times k$ -матрицу,  $E_k = I_k(:, k:-1:1)$ .

#### 4.7.2. Решение уравнений Юла–Уолкера

Мы начнем с представления алгоритма Дурбина для уравнений Юла–Уолкера, которые возникают в связи с линейными задачами предсказания. Пусть для некоторого  $k$ , которое удовлетворяет условию  $1 \leq k \leq n - 1$ , мы решили систему Юла–Уолкера  $T_k y = -r = -(r_1, \dots, r_k)^T$  порядка  $k$ . Покажем, как теперь можно

решить систему порядка  $k + 1$

$$\begin{bmatrix} T_k & E_k r \\ r^T E_k & 1 \end{bmatrix} \begin{bmatrix} z \\ \alpha \end{bmatrix} = -\begin{bmatrix} r \\ r_{k+1} \end{bmatrix}$$

за  $O(k)$  флопов. Сначала заметим, что

$$z = T_k^{-1}(-r - \alpha E_k r) = y - \alpha T_k^{-1} E_k r$$

и

$$\alpha = -r_{k+1} - r^T E_k z.$$

Так как  $T_k^{-1}$  персимметрична,  $T_k^{-1} E_k = E_k T_k^{-1}$ , и поэтому

$$z = y - \alpha E_k T_k^{-1} r = y + \alpha E_k y.$$

Подставляя это в верхнее выражение для  $\alpha$ , находим

$$\alpha = -r_{k+1} - r^T E_k (y + \alpha E_k y) = -(r_{k+1} + r^T E_k y)/(1 + r^T y).$$

Знаменатель является положительным, поскольку  $T_{k+1}$  положительно определена и, кроме того,

$$\begin{bmatrix} I & E_k y \\ 0 & 1 \end{bmatrix}^T \begin{bmatrix} T_k & E_k r \\ r^T E_k & 1 \end{bmatrix} \begin{bmatrix} I & E_k y \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} T_k & 0 \\ 0 & 1 + r^T y \end{bmatrix}.$$

Мы продемонстрировали  $k$ -й шаг алгоритма, предложенного Дурбином (1960). Он состоит в решении систем Юла–Уолкера

$$T_k y^{(k)} = -r^k = -(r_1, \dots, r_k)^T$$

для  $k = 1:n$  и может быть записан следующим образом:

$$\begin{aligned} y^{(1)} &= -r_1 \\ \text{for } k &= 1:n-1 \\ \beta_k &= 1 + [r^{(k)}]^T y^{(k)} \\ \alpha_k &= -(r_{k+1} + r^{(k)T} E_k y^{(k)})/\beta_k \\ z_k &= y^{(k)} + \alpha_k E_k y^{(k)} \\ y^{(k+1)} &= \begin{bmatrix} z^{(k)} \\ \alpha_k \end{bmatrix} \end{aligned} \tag{4.7.1}$$

end

Как теперь нами установлено, этот алгоритм требует  $3n^2$  флопов для произвольного вектора  $y = y^{(n)}$ . Возможно, однако, уменьшить общий объем работы, используя некоторые приведенные выше выражения:

$$\begin{aligned} \beta_k &= 1 + [r^{(k)}]^T y^{(k)} = \\ &= 1 + [r^{(k-1)T} r_k] [y^{(k-1)} + \alpha_{k-1} E_{k-1} y^{(k-1)}] = \\ &= (1 + [r^{(k-1)T} y^{(k-1)}]) + \alpha_{k-1} ([r^{(k-1)T} E_{k-1} y^{(k-1)}] + r_k) = \\ &= \beta_{k-1} + \alpha_{k-1} (-\beta_{k-1} \alpha_{k-1}) = \\ &= (1 - \alpha_{k-1}^2) \beta_{k-1}. \end{aligned}$$

Применяя эту рекурсию, мы получаем следующий алгоритм:

**Алгоритм 4.7.1 (Дурбин).** Даны вещественные числа  $1 = r_0, r_1, \dots, r_n$ , такие, что матрица  $T = (r_{|i-j|}) \in \mathbb{R}^{n \times n}$  является положительно определенной, следующий алгоритм вычисляет вектор  $y \in \mathbb{R}^n$ , такой, что  $Ty = -(r_1, \dots, r_n)^T$ .

```

 $y(1) = -r(1); \quad \beta = 1; \quad \alpha = -r(1)$ 
for  $k = 1:n-1$ 
   $\beta = (1 - \alpha^2)\beta$ 
   $\alpha = -(r(k+1) + r(k:-1:1)^T y(1:k))/\beta$ 
  for  $i = 1:k$ 
     $z(i) = y(i) + \alpha y(k+1-i)$ 
  end
   $y(1:k) = z(1:k); \quad y(k+1) = \alpha$ 
end

```

Алгоритм требует  $2n^2$  флоков. Мы включили вспомогательный вектор  $z$  для прозрачности алгоритма, но он может быть убран.

**Пример 4.7.1.** Пусть мы хотим решить систему Юла–Уолкера

$$\begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.5 & 1 & 0.5 \\ 0.2 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = -\begin{bmatrix} 0.5 \\ 0.2 \\ -0.1 \end{bmatrix},$$

используя алгоритм 4.7.1. После первого прохождения через цикл мы получим

$$\alpha = 1/15, \quad \beta = 3/4, \quad y = \begin{bmatrix} -8/15 \\ 1/15 \end{bmatrix}.$$

Потом вычислим величины

$$\begin{aligned} \beta &= (1 - \alpha^2)\beta = 56/75, \\ \alpha &= -(r_3 + r_2 y_1 + r_1 y_2)/\beta = -1/28, \\ z_1 &= y_1 + \alpha y_2 = -225/420, \\ z_2 &= y_2 + \alpha y_1 = -36/420, \end{aligned}$$

дающие окончательное решение  $y = (-75, 12, -5)^T/140$ .

### 4.7.3. Задача с произвольной правой частью

Если еще немножко потрудиться, можно решить симметричную положительно определенную тёплацеву систему, которая имеет произвольную правую часть. Предположим, что мы решили систему

$$T_k x = b = (b_1, \dots, b_k)^T \tag{4.7.2}$$

для некоторого  $k$ , удовлетворяющего условию  $1 \leq k < n$ , и что сейчас мы хотим решить систему

$$\begin{bmatrix} T_k & E_k z \\ r^T E_k & 1 \end{bmatrix} \begin{bmatrix} v \\ \mu \end{bmatrix} = \begin{bmatrix} b \\ b_{k+1} \end{bmatrix}. \tag{4.7.3}$$

Здесь, как и выше,  $r = (r_1, \dots, r_k)^T$ . Допустим также, что решение системы Юла–Уолкера  $k$ -го порядка  $T_k y = -r$  уже получено. Так как

$$v = T_k^{-1}(b - \mu E_k r) = x + \mu E_k y,$$

то отсюда следует, что

$$\begin{aligned} \mu &= b_{k+1} - r^T E_k v = b_{k+1} - r^T E_k x - \mu r^T y = \\ &= (b_{k+1} - r^T E_k x)/(1 + r^T y). \end{aligned}$$

Следовательно, мы можем сделать переход от (4.7.2) к (4.7.3) за  $O(k)$  флоков.

В целом мы можем эффективно решить систему  $T_n x = b$ , решая системы

$T_k x^{(k)} = b^{(k)} = (b_1, \dots, b_k)^T$  и  $T_k y^{(k)} = -r^{(k)} = (r_1, \dots, r_k)^T$  «параллельно» для  $k = 1:n$ . Это составляет содержание следующего алгоритма.

**Алгоритм 4.7.2 (Левинсон).** Даны вектор  $b \in \mathbb{R}^n$  и вещественные числа  $1 = r_0, r_1, \dots, r_n$ , такие, что матрица  $T(r_{|i-j|}) \in \mathbb{R}^{n \times n}$  является положительно определенной; следующий алгоритм вычисляет вектор  $x \in \mathbb{R}^n$ , такой, что  $Tx = b$ .

```

 $y(1) = -r(1); \quad x(1) = b(1); \quad \beta = 1; \quad \alpha = -r(1)$ 
for  $k = 1:n - 1$ 
   $\beta = (1 - \alpha^2)\beta; \quad \mu = (b(k + 1) - r(1:k)^T x(k: - 1:1))/\beta$ 
   $v(1:k) = x(1:k) + \mu y(k: - 1:1)$ 
   $x(1:k) = v(1:k); \quad x(k + 1) = \mu$ 
  if  $k < n - 1$ 
     $\alpha = -(r(k + 1) + r(1:k)^T y(k: - 1:1))/\beta$ 
     $z(1:k) = y(1:k) + \alpha y(k: - 1:1)$ 
     $y(1:k) = z(1:k); \quad y(k + 1) = \alpha$ 
  end
end
```

Алгоритм требует  $4n^2$  флопов. Векторы  $z$  и  $v$  введены для ясности, и без них можно обойтись.

**Пример 4.7.2.** Пусть мы хотим решить симметричную положительно определенную тёплицеву систему

$$\begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.5 & 1 & 0.5 \\ 0.2 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = -\begin{bmatrix} 4 \\ -1 \\ 3 \end{bmatrix}.$$

используя приведенный выше алгоритм. После первого прохождения через цикл мы получим

$$\alpha = 1/15, \quad \beta = 3/4, \quad y = \begin{bmatrix} -8/15 \\ 1/15 \end{bmatrix}, \quad x = \begin{bmatrix} 6 \\ -4 \end{bmatrix}.$$

Потом вычислим величины

$$\beta = (1 - \alpha^2)\beta = 56/75, \quad \mu = (b_3 - r_1 x_2 - r_2 x_1)/\beta = 285/56,$$

$$v_1 = x_1 + \mu y_2 = 355/56, \quad v_2 = x_2 + \mu y_1 = -376/56,$$

дающие окончательное решение  $x = (355, -376, 285)^T/56$ .

#### 4.7.4. Вычисление обратной матрицы

Одно из наиболее удивительных свойств симметричной положительно определенной тёплицевой матрицы  $T_n$  состоит в том, что обратная к ней может быть вычислена полностью за  $O(n^2)$  флопов. Чтобы получить алгоритм, реализующий эту процедуру, разобьем  $T_n^{-1}$  следующим образом:

$$T_n^{-1} = \begin{bmatrix} A & Er \\ r^T E & 1 \end{bmatrix} = \begin{bmatrix} B & v \\ v^T & \gamma \end{bmatrix}, \tag{4.7.4}$$

где  $A = T_{n-1}$ ,  $E = E_{n-1}$  и  $r = (r_1, \dots, r_{n-1})^T$ . Из уравнения

$$\begin{bmatrix} A & Er \\ r^T E & 1 \end{bmatrix} \begin{bmatrix} v \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

следует, что  $Av = -\gamma Er = -\gamma E(r_1, \dots, r_{n-1})^T$  и  $\gamma = 1 - r^T Ev$ . Если вектор  $y$  разрешает систему Юла–Уолкера ( $n - 1$ ) порядка  $Ay = -r$ , тогда из этих выражений

вытекает, что

$$\begin{aligned}\gamma &= 1/(1 + r^T y), \\ v &= \gamma E y.\end{aligned}$$

Таким образом, последние строку и столбец матрицы  $T_n^{-1}$  получить легко.

Нам осталось разработать формулы для получения элементов подматрицы  $B$  из (4.7.4). Так как  $AB + Erv^T = I_{n-1}$ , отсюда следует, что

$$B = A^{-1} - (A^{-1} Er) v^T = A^{-1} + \frac{vv^T}{\gamma}.$$

И так как  $A = T_{n-1}$  является невырожденной и тёплицевой, обратная к ней персимметрична. Таким образом,

$$\begin{aligned}b_{ij} &= (A^{-1})_{ij} + \frac{v_i v_j}{\gamma} = \\ &= (A^{-1})_{n-j, n-i} + \frac{v_i v_j}{\gamma} = \\ &= b_{n-j, n-i} - \frac{v_{n-j} v_{n-i}}{\gamma} + \frac{v_i v_j}{\gamma} = \\ &= b_{n-j, n-i} + \frac{1}{\gamma}(v_i v_j - v_{n-j} v_{n-i}).\end{aligned}\quad (4.75)$$

Это показывает, что, хотя  $B$  не является персимметричной, мы можем легко вычислить элемент  $b_{ij}$ , используя его отражение относительно северо-восточной – юго-западной диагонали. Это в сочетании с фактом, что  $A^{-1}$  персимметричная, приводит нас к определению  $B$  от «краев» к «середине».

Поскольку порядок операций правильно описать затруднительно, мы сначала покажем наглядно формальные детали алгоритма. С этой целью допустим, что известны последние строка и столбец матрицы  $A^{-1}$ :

$$A^{-1} = \begin{bmatrix} u & u & u & u & u & k \\ u & u & u & u & u & k \\ u & u & u & u & u & k \\ u & u & u & u & u & k \\ u & u & u & u & u & k \\ k & k & k & k & k & k \end{bmatrix}$$

Здесь  $u$  и  $k$  обозначают неизвестные и известные элементы соответственно и  $n = 6$ . Поочередно используя персимметрию матрицы  $A^{-1}$  и рекурсию (4.7.5), мы можем вычислить  $B$  следующим образом:

$$\xrightarrow{\text{персимм.}} \begin{bmatrix} k & k & k & k & k & k \\ k & u & u & u & u & k \\ k & u & u & u & u & k \\ k & u & u & u & u & k \\ k & u & u & u & u & k \\ k & k & k & k & k & k \end{bmatrix} \xrightarrow{(4.7.5)} \begin{bmatrix} k & k & k & k & k & k \\ k & u & u & u & u & k \\ k & u & u & u & u & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \end{bmatrix}$$

$$\begin{array}{c}
 \xrightarrow{\text{персимм.}} \left[ \begin{array}{cccccc} k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & u & u & k & k \\ k & k & u & u & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \end{array} \right] \xrightarrow{(4.7.5)} \left[ \begin{array}{cccccc} k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & u & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \end{array} \right] \\
 \xrightarrow{\text{персимм.}} \left[ \begin{array}{cccccc} k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \\ k & k & k & k & k & k \end{array} \right].
 \end{array}$$

Конечно, когда вычисляемая матрица одновременно и симметричная и персимметрическая, такая, как  $A^{-1}$ , то необходимо вычислять только «верхний клин» матрицы, например

$$\begin{matrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & & \\ \times & \times & & & & \end{matrix} \quad (n = 6).$$

С учетом последнего замечания, мы готовы представить алгоритм полностью.

**Алгоритм 4.7.3 (Тренч).** Даны вещественные числа  $1 = r_0, r_1, \dots, r_n$ , такие, что матрица  $T = (r_{|i-j|}) \in \mathbb{R}^{n \times n}$  является положительно определенной, следующий алгоритм вычисляет матрицу  $B = T_n^{-1}$ . Вычисляются только те  $b_{ij}$ , для которых  $i \leq j$  и  $i + j \leq n + 1$ .

Используя алгоритм 4.7.1, решить систему  $T_{n-1}y = -(r_1, \dots, r_{n-1})^T$ .

```

 $\gamma = 1/(r(1:n - 1)^T y(n:-1:1))$ 
 $v(1:n - 1) = \gamma y(n - 1:-1:1)$ 
 $B(1, 1) = \gamma$ 
 $B(1, 2:n) = v(n - 1:-1:1)^T$ 
for  $i = 2:\text{floor}((n - 1)/2) + 1$ 
  for  $j = i:n - i + 1$ 
     $\cdot B(i, j) = B(i - 1, j + 1) +$ 
     $(v(n + 1 - j)v(n + 1 - i) - v(i - 1)v(j - 1))/\gamma$ 
  end
end

```

Алгоритм требует  $13n^2/4$  флопов.

**Пример 4.7.3.** Если приведенный выше алгоритм применяется для вычисления обратной матрицы  $B$  положительно определенной тёплицевой матрицы

$$\begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.5 & 1 & 0.5 \\ 0.2 & 0.5 & 1 \end{bmatrix},$$

то мы получим  $\gamma = 75/56$ ,  $b_{11} = 75/56$ ,  $b_{12} = -5/7$ ,  $b_{13} = 5/56$  и  $b_{22} = 12/7$ .

#### 4.7.5. Вопросы устойчивости

Анализ ошибок упомянутых выше алгоритмов был проведен Субенко (1978), и мы вкратце опишем некоторые из его результатов.

Оказывается, ключевыми величинами являются  $\alpha_k$  из (4.7.1). В точной арифметике эти скаляры удовлетворяют соотношению

$$|\alpha_k| < 1$$

и могут быть использованы для оценки  $\|T_n^{-1}\|_1$ :

$$\max \left\{ \frac{1}{\prod_{j=1}^{n-1} (1 - \alpha_j^2)}, \frac{1}{\prod_{j=1}^{n-1} (1 - \alpha_j)} \right\} \leq \|T_n^{-1}\|_1 \leq \prod_{j=1}^{n-1} \frac{1 + |\alpha_j|}{1 - |\alpha_j|}. \quad (4.7.7)$$

Более того, решение системы Юла–Уолкера  $T_n y = -r(1:n)$  удовлетворяет соотношению

$$\|y\|_1 = \left( \prod_{k=1}^{n-1} (1 + \alpha_k) \right) - 1 \quad (4.7.8)$$

в том случае, если все  $\alpha_k$  неотрицательные.

Теперь если  $\hat{x}$  – это вычисленное алгоритмом Дурбина решение уравнений Юла–Уолкера, то  $r_D = T_n \hat{x} + r$  может быть оценено следующим образом:

$$\|r_D\| \approx u \prod_{k=1}^n (1 + |\hat{\alpha}_k|),$$

где  $\hat{\alpha}_k$  – вычисленный аналог  $\alpha_k$ . С помощью сравнения, так как каждый  $|r_i|$  ограничен единицей, получаем, что  $\|r_C\| \approx u \|y\|_1$ , где  $r_C$  – невязка, соответствующая вычисленному решению, полученному алгоритмом Холецкого. Заметим, что невязки могут быть сравняны по величине в том случае, если справедлива формула (4.7.7). Эксперимент дает основания предположить, что это имеет место, даже если некоторые из  $\alpha_k$  отрицательные. Аналогичные объяснения применимы к численному поведению алгоритма Левинсона.

Для метода Тренча можно показать, что матрица  $\hat{B}$  – вычисленный аналог обратной к  $T_n^{-1}$ , удовлетворяет условию

$$\frac{\|T_n^{-1} - \hat{B}\|_1}{\|T_n^{-1}\|_1} \approx u \prod_{k=1}^n \frac{1 + |\hat{\alpha}_k|}{1 - |\hat{\alpha}_k|}.$$

В свете формулы (4.7.7) видно, что правая часть является приближенной верхней оценкой для  $u \|T_n^{-1}\|_1$ , которая аппроксимирует величину относительной ошибки, когда  $T_n^{-1}$  вычислена при помощи разложения Холецкого.

#### Задачи

**4.7.1.** Для произвольного вектора  $v \in \mathbb{R}^n$  определим векторы  $v_+ = (v + E_n v)/2$  и  $v_- = (v - E_n v)/2$ . Пусть матрица  $A \in \mathbb{R}^{n \times n}$  является симметричной и персимметричной. Показать, что если  $Ax = b$ , то  $Ax_+ = b_+$  и  $Ax_- = b_-$ .

**4.7.2.** Пусть матрица  $U \in \mathbb{R}^{n \times n}$  является верхней унитреугольной и обладает свойством:  $U(1:k-1, k) = E_{k-1} y^{(k-1)}$ , где  $y^{(k)}$  определен как в (4.7.1). Показать, что

$$U^T U = \text{diag}(1, \beta_1, \dots, \beta_{n-1}).$$

**4.7.3.** Пусть вектор  $z \in \mathbb{R}^n$ , и пусть матрица  $S \in \mathbb{R}^{n \times n}$  ортогональна. Показать, что если

$$X = [z, Sz, \dots, S^{n-1}z],$$

тогда  $X^T X$  является тёплицевой.

**4.7.4.** Рассмотреть  $LDL^T$ -разложение  $n \times n$  симметричной, трехдиагональной, положительно определенной тёплицевой матрицы. Показать, что  $d_n$  и  $l_{n,n-1}$  сходится при  $n \rightarrow \infty$ .

**4.7.5.** Показать, что произведение двух нижних треугольных тёплицевых матриц является тёплицевым.

**4.7.6.** Дать алгоритм вычисления  $\mu \in \mathbb{R}$ , такого, что

$$T_n + \mu(e_n e_1^T + e_1 e_n^T)$$

является вырожденной. Предположить, что  $T_n = (r_{|i-j|})$  является положительно определенной и  $r_0 = 1$ .

**4.7.7.** Переписать алгоритм 4.7.2 таким образом, чтобы не требовались векторы  $z$  и  $v$ .

**4.7.8.** Предложить алгоритм для вычисления  $x_\infty$  ( $T_k$ ) при  $\kappa = 1:n$ .

### Замечания и литература к § 4.7

Всякий кто отважится проникнуть в литературу по быстрым тёплицевым методам, должен прежде всего прочитать

Bunch J. R. (1985). "Stability of Methods for Solving Toeplitz systems of Equations", SIAM J. Sci. Stat. Comp. 6, 349–364.

для выяснения вопросов устойчивости. Как и вообще, среди «быстрых алгоритмов» есть много неустойчивых тёплицевых методов и необходимо проявлять осторожность.

Первоначальными работами по трем методам, описанным в этом разделе, являются:

Durbin J. (1960). "The Fitting of Time Series Models", Rev. Inst. Int. Stat. 28, 233–243.

Levinson N. (1947). "The Weiner RMS Error Criterion in Filter Design and Prediction", J. Math. Phys. 25, 261–278.

Trench W. F. (1964). "An Algorithm for the Inversion of Finite Toeplitz Matrices", J. SIAM 12, 515–522.

Более детальное описание несимметричного алгоритма Тренча дается в работе

Zohar S. (1969). "Toeplitz Matrix Inversion: The Algorithm of W. F. Trench", J. ACM 16, 592–601.

Другие ссылки относительно обращений тёплицевой матрицы содержат работы

Trench W. F. (1974). "Inversion of Toeplitz Band Matrices", Math. Comp. 28, 1089–1095.

Watson G. A. (1973). "An Algorithm for the Inversion of Block Matrices of Toeplitz Form", J. ACM 20, 409–415.

Оценки ошибок, которые мы приводили, взяты из анализа ошибок округления, имеющегося в работах

Cybenko G. (1978). "Error Analysis of Some Signal Processing Algorithms", Ph. D. thesis, Princeton University.

Cybenko G. (1980). "The Numerical Stability of the Levinson–Durbin Algorithm for Toeplitz Systems of Equations", SIAM J. Sci. and Stat. Comp. 1, 303–319.

Также известны методы треугольного разложения тёплицевых систем сложности  $O(n^3)$ .

Phillips J. L. (1971). "The Triangular Decomposition of Hankel Matrices", Math. Comp. 25, 599–602.

Rissanen J. (1973). "Algorithms for Triangular Decomposition of Block Hankel and Toeplitz Matrices with Application for Factoring Positive Matrix Polynomials", Math. Comp. 27, 147–154.

Упомянем следующие работы, описывающие некоторые важные приложения тёплицевых матриц:

Makhoul J. (1975). "Linear Prediction: A Tutorial Review", Proc. IEEE 63(4), 561–580.

Markel J. and Gray A. (1976). Linear Prediction of Speech, Springer-Verlag, Berlin and New York.  
 Oppenheim A. V. (1978). Applications of Digital Signal Processing, Prentice-Hall, Englewood Cliffs.

### **Добавления при переводе к § 4.7**

Отметим монографии, посвященные проблемам вычислений с матрицами типа тёплitzевых:

Воеводин В. В., Тыртышников Е. Е. (1978). Вычислительные процессы с тёплitzевыми матрицами.– М.: Наука.

Тыртышников Е. Е. (1989). Тёплitzевые матрицы, некоторые их аналоги и приложения.– М.: ОВМ АН СССР.

Различные алгебраические вопросы, связанные с тёплitzевыми матрицами и их обобщениями, рассматриваются в книгах

Heinig G., Rost K. (1984). "Algebraic Methods for Toeplitz-like Matrices and Operators", Berlin, Akademie-Verlag.

Иохвидов И. С. (1974). Ганкелевы и тёплitzевые матрицы и формы.– М.: Наука.

Обширная библиография и сведения о приложениях имеются в следующих обзорных работах:

Воеводин В. В., Тыртышников Е. Е. (1983). Вычисления с тёплitzевыми матрицами.– Вычислительные процессы и системы. Вып. 1.– М.: Наука, 124–266.

Лифанов И. К., Тыртышников Е. Е. (1990). Тёплitzевые матрицы и сингулярные интегральные уравнения.– Вычислительные процессы и системы. Вып. 7.– М.: Наука, 94–278.

Bultheel A. (1985). "Triangular Decomposition of Toeplitz and Related Matrices: A Guided Tour in Algorithmic Aspects", Bull. Soc. Math. Belg., A 37, N 2, 101–141.

В последнее время появилось довольно много работ, посвященных «сверхбыстрым» алгоритмам, решающим тёплitzевые системы порядка  $n$  за  $O(n \log_2^2 n)$  операций. Отметим здесь лишь некоторые наиболее свежие из них:

de Hoog F. (1987). "A New Algorithm for Solving Toeplitz Systems of Equations", *Lin. Alg. Appl.* 88/89, 123–138.

Ammar G. S., Gragg W. G. (1988). "Superfast Solution of Real Positive Definite Toeplitz Systems", *SIAM J. Math. Anal. Appl.* 9, N 1, 61–76.

Тыртышников Е. Е. (1989). Новые быстрые алгоритмы для систем с ганкелевой и тёплitzевой матрицами.– ЖВМ и МФ, 29, № 5, 645–652.

Рекордное число операций для произвольных (блочных) систем с невырожденными ведущими (блочными) подматрицами составляет  $8/3n \log_2^2 n$ . См. упоминавшуюся выше монографию (Е. Е. Тыртышников (1989)), а также

Tyrtysnikov E. E. (1989). "Fast Algorithms for Toeplitz and Quasi-Toeplitz Systems", *Sov. J. Numer. Anal. Math. Modelling*, 5, 138–160.

## Глава 5

# Ортогонализация и метод наименьших квадратов

В этой главе речь пойдет главным образом о решении переопределенных систем уравнений методом наименьших квадратов, т.е. о минимизации функционала  $\|Ax - b\|_2$ , где  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ ,  $b \in \mathbb{R}^m$ . Наиболее надежные методы решения этой задачи связаны с приведением матрицы  $A$  к разнообразным каноническим формам путем ортогональных преобразований. Центральную роль при этом играют отражения Хаусхолдера и вращения Гивенса, с рассмотрения которых мы и начинаем эту главу. В § 5.2 обсуждается вычисление разложения  $A = QR$ , где  $Q$  – ортогональная,  $R$  – верхняя треугольная матрица. Оно сводится к нахождению ортонормированного базиса в  $\text{range}(A)$ . В § 5.3 показано, что  $QR$ -разложение может быть применено для решения задачи минимальных квадратов с матрицей полного ранга. Этот метод мы затем сравниваем с методом нормальных уравнений, предварительно развив соответствующую теорию возмущений.

В § 5.4 и § 5.5 мы рассматриваем методы, пригодные в сложной ситуации, когда матрица  $A$  неполноранговая (или почти неполноранговая). Здесь представлены  $QR$ -разложение с выбором ведущего элемента по столбцу и SVD.

В § 5.6 мы обсуждаем некоторые шаги, которые могут быть предприняты для улучшения качества вычисленного решения задачи минимальных квадратов. § 5.7 содержит несколько замечаний о недопределенных системах.

## 5.1. Матрицы Хаусхолдера и Гивенса

Напомним, что  $n \times n$ -матрица  $Q$  называется *ортогональной*, если  $Q^T Q = I_n$ . Ортогональные матрицы играют важную роль при решении задач на собственные значения и задач минимальных квадратов. В этом разделе мы вводим ключевые для эффективных вычислений с ортогональными матрицами преобразования: отражения Хаусхолдера и вращения Гивенса.

### 5.1.1. Двумерный случай

Полезно посмотреть, какой геометрический смысл имеют вращения и отражения в случае  $n = 2$ . Ортогональная  $2 \times 2$ -матрица  $Q$  называется *матрицей вращения*, если

$$Q = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix},$$

Если  $y = Q^T x$ , то  $y$  получается поворотом  $x$  на угол  $\theta$  против часовой стрелки.

Ортогональная  $2 \times 2$ -матрица  $Q$  называется *матрицей отражения*, если

$$Q = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}.$$

Если  $y = Q^T x$ ,  $x = Qy$ , то  $y$  получается отражением  $x$  относительно оси, определяемой как

$$S = \text{span} \left\{ \begin{bmatrix} \cos(\theta)/2 \\ \sin(\theta)/2 \end{bmatrix} \right\}.$$

Вращения и отражения привлекательны с вычислительной точки зрения, так как они легко строятся и могут быть использованы для обнуления отдельных элементов векторов при надлежащем выборе угла поворота или плоскости отражения.

**Пример 5.1.1.** Пусть  $x = (1, \sqrt{3})^T$ . Положим

$$Q = \begin{bmatrix} \cos(-60^\circ) & \sin(-60^\circ) \\ -\sin(-60^\circ) & \cos(-60^\circ) \end{bmatrix} = \begin{bmatrix} 1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & 1/2 \end{bmatrix}.$$

Тогда  $Q^T x = (2, 0)$ . Таким образом, поворот на  $(-60^\circ)$  обнуляет вторую компоненту  $X$ . Если

$$Q = \begin{bmatrix} \cos(30^\circ) & \sin(30^\circ) \\ \sin(30^\circ) & -\cos(30^\circ) \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 & 1/2 \\ 1/2 & -\sqrt{3}/2 \end{bmatrix},$$

то  $Q^T x = (2, 0)^T$ . Таким образом, вторую компоненту  $x$  можно обнулить, отражая  $x$  относительно прямой с наклоном  $30^\circ$ .

### 5.1.2. Отражения Хаусхолдера

Пусть  $v \in \mathbb{R}^n$  – ненулевой вектор.  $n \times n$ -Матрица  $P$  вида

$$P = I - 2vv^T/v^T v \quad (5.1.1)$$

называется *отражением Хаусхолдера* (матрицей Хаусхолдера, преобразованием Хаусхолдера). Вектор  $v$  называется *вектором Хаусхолдера*. При умножении матрицы  $P$  на вектор  $x$  последний отражается относительно гиперплоскости  $\text{span}\{v\}^\perp$ . Легко проверить, что матрицы Хаусхолдера симметричны и ортогональны.

Отражения Хаусхолдера похожи на введенные в § 3.2.1 преобразования Гаусса в двух отношениях: они являются одноранговыми модификациями единичной матрицы и они могут быть использованы для обнуления выбранных компонент вектора. В частности, предположим, что задан вектор  $0 \neq x \in \mathbb{R}^n$  и мы хотим, чтобы вектор  $Px$  был кратен  $e_1$  первому столбцу  $I_n$ . Для любого  $x \in \mathbb{R}^n$  имеем

$$Px = \left( I - \frac{2vv^T}{v^T v} \right) x = x - \frac{2v^T x}{v^T v} e_1,$$

и поэтому из  $Px \in \text{span}\{e_1\}$  следует, что  $v \in \text{span}\{x, e_1\}$ . Полагая  $v = x + \alpha e_1$ , получаем

$$v^T x = x^T x + \alpha x_1$$

и

$$v^T v = x^T x + 2\alpha x_1 + \alpha^2,$$

так что

$$Px = \left( 1 - 2 \frac{x^T x + \alpha x_1}{x^T x + 2\alpha x_1 + \alpha^2} \right) x - 2\alpha \frac{v^T x}{v^T v} e_1.$$

Чтобы коэффициент при  $x$  оказался нулевым, достаточно положить  $\alpha = \pm ||x||_2$ .

Тогда

$$v = x \pm \|x\|_2 e_1 \Rightarrow Px = \left( I - 2 \frac{vv^T}{v^T v} \right) x = \mp \|x\|_2 e_1. \quad (5.1.2)$$

Именно простота нахождения  $v$  делает отражения Хаусхолдера очень полезными.

**Пример 5.1.2.** Для  $x = (3, 1, 5, 1)^T$  и  $v = (9, 1, 5, 1)^T$  матрица

$$P = I - 2 \frac{vv^T}{v^T v} = \frac{1}{54} \begin{bmatrix} -27 & -9 & -45 & -9 \\ -9 & 53 & -5 & -1 \\ -45 & -5 & 29 & -5 \\ -9 & -1 & -5 & 53 \end{bmatrix}$$

удовлетворяет  $Px = (-6, 0, 0, 0)^T$ .

### 5.1.3. Вычисление вектора Хаусхолдера

С нахождением матрицы Хаусхолдера (или, что то же самое, вектора Хаусхолдера) связан ряд важных на практике деталей. Одна из них – выбор знака при определении  $v$  по формуле (5.1.2). Если  $x$  почти коллинеарен  $e_1$ , то вектор  $v = x - \text{sign}(x_1)\|x\|_2 e_1$  имеет малую норму. Вследствие этого возможно появление большой относительной ошибки при вычислении множителя  $\beta = 2/v^T v$ . Эту трудность можно обойти, просто взяв  $a$  с тем же знаком, что и первая компонента вектора  $x$ :

$$v = x + \text{sign}(x_1)\|x\|_2 e_1.$$

Это обеспечивает выполнение неравенства  $\|v\|_2 \geq \|x\|_2$  и гарантирует почти точную ортогональность вычисленной матрицы  $P$  (см. ниже). Заметим также, что  $|v_1| = \|v\|_\infty$ .

Менее критично, но полезно придерживаться в дальнейшем такой нормировки вектора  $v$ , что  $v(1) = 1$ . Это нестандартная нормировка, но она упрощает представление нескольких важных алгоритмов, в которых требуется хранить вектор Хаусхолдера. Мы будем называть  $v(2:n)$  *существенной*, или *значимой*, частью  $v$ . Заметим, что значимая часть  $v$  может быть записана на место обнуляемой части вектора  $x$ .

Следующая процедура вычисляет вектор Хаусхолдера:

**Алгоритм 5.1.1 (Вычисление вектора Хаусхолдера).** По  $n$ -вектору  $x$  данная функция вычисляет  $n$ -вектор  $v$ , такой, что  $v(1) = 1$  и все компоненты вектора  $(1 - 2vv^T/v^T v)x$ , кроме первой, равны нулю.

```
function: v = house(x)
  n = length(x); mu = ||x||_2; v = x
  if mu != 0
    beta = x(1) + sign(x(1))mu
    v(2:n) = v(2:n)/beta
  end
  v(1) = 1
end house
```

Этот алгоритм требует примерно  $3n$  флоков.

### 5.1.4. Умножение на матрицы Хаусхолдера

Применяя преобразование Хаусхолдера к матрицам, очень важно учесть специальную структуру матрицы Хаусхолдера. Пусть  $A$  – матрица,  $P = I - 2vv^T/v^T v$ .

Тогда

$$PA = \left( I - 2 \frac{vv^T}{v^T v} \right) A = A + v w^T,$$

где  $w = \beta A^T v$ ,  $\beta = -2/v^T v$ . Таким образом, хаусхолдерова модификация матрицы слагается из умножения матрицы на вектор и модификации внешним произведением векторов. Если бы мы не заметили этого и обращались бы с  $P$  как с матрицей общего вида, то объем работы возрос бы на порядок. Преобразование Хаусхолдера никогда не требует явного формирования матрицы Хаусхолдера. Следующие две функции формально подтверждают это.

**Алгоритм 5.1.2 (Умножение на матрицу Хаусхолдера слева).** Для данной  $m \times n$ -матрицы  $A$  и ненулевого  $m$ -вектора  $v$ , где  $v(1) = 1$ , следующий алгоритм строит на месте матрицы  $A$  матрицу  $PA$ , где  $P = I - 2vv^T/v^T v$ .

```
function: A = row.house(A, v)
    β = -2/v^T v
    w = β A^T v
    A = A + v w^T
end row.house
```

**Алгоритм 5.1.3 (Умножение на матрицу Хаусхолдера справа).** Для данной  $m \times n$ -матрицы  $A$  и ненулевого  $m$ -вектора  $v$ , где  $v(1) = 1$ , следующий алгоритм строит на месте матрицы  $A$  матрицу  $AP$ , где  $P = I - 2vv^T/v^T v$ .

```
function: A = col.house(A, v)
    β = -2/v^T v
    w = β A v
    A = A + w v^T
end col.house
```

Каждая из вышеприведенных процедур преобразования Хаусхолдера требует около  $4mn$  флопов. Заметим, что как в **row.house**, так и в **col.house** можно учесть наличие нормировки  $v(1) = 1$ . При малых  $m$  это может оказаться существенным для **row.house**, а при малых  $n$  – для **col.house**, поскольку экономит одно умножение.

В типичном случае функции **house**, **col.house** и **row.house** используются для обнуления заданной части строки или столбца матрицы. Пусть, к примеру, мы хотим записать на место матрицы  $A \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ) матрицу  $B = Q^T A$ , где  $Q$  – такая ортогональная матрица, что  $B(j+1:m, j) = 0$  для некоторого  $j$ ,  $1 \leq j \leq n$ . Допустим еще, что  $A(j:m, 1:j-1) = 0$  и мы хотим разместить значимую часть вектора Хаусхолдера на месте  $A(j+1:m, j)$ . Эта задача может быть решена следующим образом:

$$\begin{aligned} v(j:m) &= \text{house}(A(j:m, j)) \\ A(j:m, j:n) &= \text{row.house}(A(j:m, j:n), v(j:m)) \\ A(j+1:m, j) &= v(j+1:m) \end{aligned}$$

С вычислительной точки зрения мы подействовали матрицей Хаусхолдера  $\tilde{P} = I - 2vv^T/v^T v$  порядка  $m-j+1$  на нижние  $m-j+1$  строки матрицы  $A$ . Однако математически мы при этом также действовали  $m \times M$ -матрицей Хаусхолдера

$$P = \begin{bmatrix} I_{j-1} & 0 \\ 0 & \tilde{P} \end{bmatrix} = I - 2 \frac{vv^T}{v^T v}, \quad v = \begin{bmatrix} 0 \\ \tilde{v} \end{bmatrix},$$

на всю матрицу  $A$ . Невзирая на это, нетривиальная часть вектора Хаусхолдера была записана на месте обнуляемых элементов массива  $A$ .

### 5.1.5. Ошибки округления

Матрицы Хаусхолдера обладают очень хорошими свойствами в отношении ошибок округления. Уилкинсон (AEP, pp. 152–62) показывает, что вектор  $\hat{v}$ , вычисляемый функцией **house**, очень близок к точному вектору  $v$ . Если  $\hat{P} = I - 2vv^T/\hat{v}^T\hat{v}$ , то

$$\|\hat{P} - P\|_2 = O(\mathbf{u}).$$

Далее, вычисленные преобразования с матрицей  $\hat{P}$  близки к точным преобразованиям с матрицей  $P$ :

$$\begin{aligned} fl(\hat{P}A) &= P(A + E), & \|E\|_2 &= O(\mathbf{u}\|A\|_2), \\ fl(A\hat{P}) &= (A + E)P, & \|E\|_2 &= O(\mathbf{u}\|A\|_2). \end{aligned}$$

### 5.1.6. Факторизованное представление

Во многих из описываемых ниже алгоритмов разложения, основанных на преобразованиях отражения, вычисляется произведение матриц Хаусхолдера

$$Q = Q_1 Q_2 \dots Q_r, \quad Q_j = I - 2 \frac{v^{(j)} v^{(j)T}}{v^{(j)T} v^{(j)}}, \quad (5.1.3)$$

где  $r \leq n$  и каждое  $v^{(j)}$  имеет вид

$$v^{(j)} = (\underbrace{0, 0, \dots, 0}_{j-1}, v_{j+1}^{(j)}, \dots, v_n^{(j)})^T.$$

Явное вычисление  $Q$  обычно не требуется, даже если  $Q$  используется в последующих вычислениях. Например, если  $C \in \mathbf{R}^{n \times q}$  и нам нужно вычислить  $Q^T C$ , мы просто выполняем цикл

```
for j = 1:r
    C = Q_j C
end
```

Хранение хаусхолдеровых векторов  $v^{(1)}, \dots, v^{(r)}$  фактически равносильно факторизованному представлению матрицы  $Q$ . Для иллюстрации экономичности этого представления рассмотрим массив  $A$ , и пусть его элементы  $A(j+1:n, j)$  содержат значимые компоненты  $v^{(j)}(j+1:n)$   $j$ -го хаусхолдерова вектора. Тогда запись матрицы  $Q^T C$  на место  $C \in \mathbf{R}^{n \times q}$  может быть реализована следующим образом:

```
for j = 1:r
    v(j) = 1; v(j+1:n) = A(j+1:n, j)
    C(j:n,:) = row.house(C(j:n,:), v(j:n))
end
```

(5.1.4)

Здесь используется около  $2qr(2n - r)$  флопов. Если бы матрица  $Q$  была явно представлена как  $n \times n$ -матрица, то вычисление  $Q^T C$  потребовало бы  $2n^2 q$  флопов.

Конечно, в некоторых приложениях явное формирование матрицы  $Q$  или, возможно, некоторой ее части, все же необходимо. Для вычисления произведения  $Q$  хаусхолдеровых матриц из (5.1.3) возможны два алгоритма: *прямое накопление*

```
Q = I_n
for j = 1:r
    Q = Q Q_j
end
```

и обратное накопление

```

 $Q = I_n$ 
for  $j = r : -1 : 1$ 
 $Q = Q_j Q$ 
end

```

Вспомним, что ведущая часть  $Q_j$  размера  $(j-1) \times (j-1)$  является единичной матрицей. Поэтому в начале обратного накопления матрица  $Q$  «почти единичная» и заполняется постепенно в процессе итераций. Это обстоятельство можно использовать для сокращения требуемого количества флопов. Напротив, при прямом накоплении матрица  $Q$  становится полной уже после первого шага.

Ввиду этого прямой порядок перемножения матрицы  $Q$  менее экономичен. Обратное накопление является наилучшей стратегией и может быть детализировано следующим образом:

```

 $Q = I_n$ 
for  $j = r : -1 : 1$ 
 $v(j) = 1; v(j+1:n) = A(j+1:n, j)$ 
 $Q(j:n, j:n) = \text{row.house}(Q(j:n, j:n), v(j:n))$ 
end

```

(5.1.5)

При этом требуется около  $4(n^2r - nr^2 + r^3/3)$  флопов.

### 5.1.7. Блоchное представление

Пусть, как в (5.1.3),  $Q = Q_1 \dots Q_r$  – произведение  $n \times n$ -матриц Хаусхолдера. Так как каждая  $Q_j$  является одноранговой модификацией единичной матрицы, из структуры хаусхолдеровых векторов следует, что матрица  $Q$  является  $r$ -ранговой модификацией единичной матрицы и может быть представлена в виде

$$Q = I + WY^T, \quad (5.1.6)$$

где  $W$  и  $Y$  –  $n \times r$ -матрицы. Для вычисления блочного представления (5.1.6) ключевой является следующая лемма:

**Лемма 5.1.1.** Пусть  $Q = I + WY^T$  – ортогональная  $n \times n$ -матрица, где  $W, Y \in \mathbb{R}^{n \times r}$ . Если  $P = I - 2vv^T/v^Tv$ , где  $v \in \mathbb{R}^n$ ,  $z = -2Qv/v^Tv$ , то

$$Q_+ = QP = I + W_+ Y_+^T,$$

где  $W_+ = [W \ z]$ ,  $Y_+ = [Y \ v]$  –  $n \times (j+1)$ -матрицы.

*Доказательство.*

$$\begin{aligned}
QP &= (I + WY^T) \left( I - 2 \frac{vv^T}{v^Tv} \right) = I + WY^T - 2 \frac{Qvv^T}{v^Tv} \\
&= I + WY^T + zv^T = I + [W \ z] [Y \ v]^T. \quad \square
\end{aligned}$$

Многократно применяя эту лемму, мы можем получить блочное представление матрицы  $Q$  из (5.1.3) из факторизованного представления следующим образом:

**Алгоритм 5.1.4.** Пусть  $Q = Q_1 \dots Q_r$  – произведение  $n \times n$ -матриц Хаусхолдера, как в (5.1.3). Алгоритм вычисляет матрицы  $W, Y \in \mathbb{R}^{n \times r}$ , такие, что  $Q = I + WY^T$ .

```

 $Y = v^{(1)}$ 
 $W = -2v^{(1)}/[v^{(1)}]^T v^{(1)}$ 
for  $j = 2:r$ 
     $z = -2(I + WY^T)v^{(j)}/[v^{(j)}]^T v^{(j)}$ 
     $W = [W \ z]$ 
     $Y = [Y \ v^{(j)}]$ 
end

```

Этот алгоритм требует около  $2r^2n - 2r^3/3$  флопов, если учитывается наличие нулей в  $v^{(j)}$ . Заметим, что  $Y$  – просто матрица хаусхолдеровых векторов, являющаяся поэтому нижней треугольной с единичной диагональю. Ясно, что главной задачей в нахождении  $WY$ -представления (5.1.6) является вычисление матрицы  $W$ .

Блочное представление произведений хаусхолдеровых матриц привлекательно в тех случаях, когда требуется умножить  $Q$  на какую-либо матрицу. Пусть  $C \in \mathbb{R}^{n \times q}$ . Тогда преобразование

$$C \leftarrow Q^T C = (I + WY^T)^T C = C + Y(W^T C)$$

содержит много операций уровня 3. В то же время при факторизованном представлении матрицы  $Q$  вычисление  $Q^T C$  использует лишь операции уровня 2, а именно, умножение матрицы на вектор и модификацию внешним произведением. Конечно, разница между уровнями 2 и 3 сглаживается по мере того, как матрица  $C$  становится более узкой.

Отметим, что  $WY$ -представление не является с геометрической точки зрения обобщением преобразования Хаусхолдера. Блочные отражения имеют вид  $Q = I - 2VV^T$ , где  $V \in \mathbb{R}^{n \times r}$  удовлетворяет  $V^T V = I_r$ . См. Schreiber, Parlett (1987), Schreiber, Van Loan (1989).

**Пример 5.1.3.** Если  $n = 4$ ,  $r = 2$ ,  $(1, .6, 0, .8)^T$  и  $(0, 1, .8, .6)^T$  – хаусхолдеровы векторы, соответствующие матрицам  $Q_1$  и  $Q_2$  соответственно, то

$$Q_1 Q_2 = I + \begin{bmatrix} -1 & 1.080 \\ -0.6 & -0.352 \\ 0 & -0.800 \\ -0.8 & 0.264 \end{bmatrix} \begin{bmatrix} 1 & 0.6 & 0 & 0.8 \\ 0 & 1 & 0.8 & 0.6 \end{bmatrix}.$$

### 5.1.8. Вращения Гивенса

Отражения Хаусхолдера чрезвычайно полезны для крупномасштабных обнулений, как, например, аннулирование всех компонент вектора, кроме первой. Однако в тех случаях, когда необходимо избирательное зануление элементов, предпочтительнее использовать *вращения Гивенса*. Эти двухранговые модификации единичной матрицы имеют вид

$$G(i, k, \theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}_{\substack{i \\ k}} \quad (5.1.7)$$

где  $c = \cos(\theta)$ ,  $s = \sin(\theta)$  для некоторого  $\theta$ . Ясно, что вращения Гивенса – ортогональные матрицы.

Умножение слева на  $G(i, k, \theta)^T$  равносильно повороту на  $\theta$  радиан против часовой стрелки в координатной плоскости  $(i, k)$ . Действительно, если  $x \in \mathbb{R}^n$  и  $y = G(i, k, \theta)^T x$ , то

$$Y_j = \begin{cases} c x_i - s x_k, & j = i, \\ s x_i + c x_k, & j = k, \\ x_j & , \quad j \neq i, k. \end{cases}$$

Из этих соотношений ясно, что мы можем обратить  $y_k$  в нуль, положив

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}. \quad (5.1.8)$$

Таким образом, обнуление заданного элемента вектора с использованием вращения Гивенса является делом очень простым.

На практике существуют лучшие способы вычисления  $c$  и  $s$ , чем (5.1.8). Например, следующий алгоритм включает защиту от переполнения.

**Алгоритм 5.1.5.** По скалярам  $a$  и  $b$  эта функция определяет  $c = \cos(\theta)$  и  $s = \sin(\theta)$  так, что

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}.$$

```
function: [c, s] = givens(a, b)
if b = 0
    c = 1; s = 0
else
    if |b| > |a|
        τ = -a/b; s = 1/sqrt(1 + τ²); c = sτ
    else
        τ = -b/a; c = 1/sqrt(1 + τ²); s = cτ
    end
end
end givens
```

Этот алгоритм использует 5 флопов и одно извлечение квадратного корня. Заметим, что он вычисляет  $\theta$ , поэтому не приходится привлекать обратные тригонометрические функции.

**Пример 5.1.4.** Пусть  $x = (1, 2, 3, 4)^T$ ,  $\cos(\theta) = 1/\sqrt{5}$ ,  $\sin(\theta) = 2/\sqrt{5}$ . Тогда  $G(2, 4, \theta)^T x = (1, \sqrt{20}, 3, 0)^T$ .

### 5.1.9. Умножение на матрицы Гивенса

Простота строения  $G(i, k, \theta)$ , разумеется, должна быть использована при вычислении произведений матриц вида  $G(i, k, \theta)^T A$  и  $AG(i, k, \theta)$ . Заметим, что умножение справа затрагивает только строки  $i$  и  $k$ , а умножение слева – только столбцы  $i$  и  $k$ . Эти вычисления реализуются следующими функциями.

**Алгоритм 5.1.6.** Пусть  $A \in \mathbb{R}^{2 \times q}$ ,  $c = \cos(\theta)$ ,  $s = \sin(\theta)$ . Следующий алгоритм записывает на место  $A$  матрицу  $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A$ .

```

function:  $A = \text{row.rot}(A, c, s)$ 
     $q = \text{cols}(A)$ 
    for  $j = 1 : q$ 
         $\tau_1 = A(1, j); \tau_2 = A(2, j)$ 
         $A(1, j) = c\tau_1 - s\tau_2;$ 
         $A(2, j) = s\tau_1 + c\tau_2$ 
    end
end row.rot
```

Этот алгоритм требует  $6q$  флопов.

**Алгоритм 5.1.7.** Пусть  $A \in \mathbb{R}^{q \times 2}$ ,  $c = \cos(\theta)$ ,  $s = \sin(\theta)$ . Следующий алгоритм записывает на место  $A$  матрицу  $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A$ .

```

function:  $A = \text{col.rot}(A, c, s)$ 
     $q = \text{rows}(A)$ 
    for  $i = 1 : q$ 
         $\tau_1 = A(i, 1); \tau_2 = A(i, 2)$ 
         $A(i, 1) = c\tau_1 - s\tau_2;$ 
         $A(i, 1) = s\tau_1 + c\tau_2$ 
    end
end col.rot
```

Этот алгоритм требует  $6q$  флопов.

Вспоминая, что  $A([i \ k], :)$  означает строки  $i$  и  $k$  матрицы  $A$ , мы видим, что

$$A([i \ k], :) = \text{row.rot}(A([i \ k], :), c, s)$$

осуществляет преобразование  $A \leftarrow G(i, k, \theta)^T A$ . Аналогично

$$A(:, [i \ k]) = \text{col.rot}(A(:, [i \ k]), c, s)$$

осуществляет преобразование  $a \leftarrow A G(i, k, \theta)$ .

Для дальнейшей иллюстрации типичного применения процедур Гивенса допустим, что  $A - m \times n$ -матрица, такая, что  $A(i-1:i, 1:j-1) = 0$  и мы хотим обнулить  $A(i, j)$  вращением  $i$ -й и  $(i-1)$ -й строк. Вот как этого можно достигнуть:

$$\begin{aligned} [c, s] &= \text{givens}(A(i-1, j), A(i, j)), \\ A(i-1:i, j:n) &= \text{row.rot}(A(i-1:i, j:n), c, s). \end{aligned}$$

### 5.1.10. Ошибки округления

Численные свойства вращений Гивенса не менее хороши, чем для отражений Хаусхолдера. Можно, в частности, показать, что вычисленные функцией `givens`  $\hat{c}$  и  $\hat{s}$  удовлетворяют

$$\begin{aligned} \hat{c} &= c(1 + \varepsilon_c), & \varepsilon_c &= O(\mathbf{u}), \\ \hat{s} &= s(1 + \varepsilon_s), & \varepsilon_s &= O(\mathbf{u}). \end{aligned}$$

Если  $c$  и  $s$  в дальнейшем используются для нахождения преобразования Гивенса, то

вычисленное преобразование является точным преобразованием близкой матрицы:

$$\begin{aligned} f l [ \hat{G}(i, k, \theta)^T A ] &= G(i, k, \theta)^T (A + E), \quad \|E\|_2 \approx u \|A\|_2, \\ f l [ A \hat{G}(i, k, \theta) ] &= (A + E) G(i, k, \theta), \quad \|E\|_2 \approx u \|A\|_2. \end{aligned}$$

Детальный анализ ошибок для вращений Гивенса можно найти у Уилкинсона (АЕР, pp. 131–39).

### 5.1.11. Представление произведений матриц Гивенса

Пусть  $Q = G_1 \dots G_t$  – произведение матриц вращения Гивенса. Так же как и в случае отражений Хаусхолдера, ортогональную матрицу  $Q$  экономичнее хранить в факторизованном виде, а не вычислять явно произведение вращений. Прием, предложенный Стьюартом (1976), позволяет очень компактно это осуществить. Идея состоит в том, чтобы связать с каждым вращением единственное число с плавающей точкой  $\rho$ . Именно, если

$$Z = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}, \quad c^2 + s^2 = 1,$$

то определим скаляр  $\rho$  следующим образом:

```
if  $c = 0$ 
     $\rho = 1$ 
elseif  $|s| < |c|$ 
     $\rho = \text{sign}(c)s/2$ 
else
     $\rho = 2\text{sign}(s)/c$ 
end
```

(5.1.9)

По существу это означает, что мы храним  $s/2$ , если синус меньше косинуса, и  $2/c$  в противном случае. Восстановление матрицы  $\pm Z$  при этом производится следующим образом:

```
if  $\rho = 1$ 
     $c = 0; s = 1$ 
elseif  $|\rho| < 1$ 
     $s = 2\rho; c = \sqrt{1 - s^2}$ 
else
     $c = 2/\rho; s = \sqrt{1 - c^2}$ 
end
```

(5.1.10)

То что фактически может быть восстановлена матрица  $-Z$  (а не  $Z$ ), обычно несущественно, поскольку если  $Z$  обнуляет в матрице какой-нибудь элемент, то и  $-Z$  делает то же самое. Хранение меньшего из чисел  $c$  и  $s$  обусловлено тем, что формула  $\sqrt{1 - x^2}$  работает с плохой точностью, если  $x$  близок к единице. Другие детали см. Stewart (1976). Конечно, для воспроизведения  $G(i, k, \theta)$  нам необходимо знать  $i$  и  $k$ , помимо  $\rho$ . Это обычно не вызывает затруднений, как мы покажем в § 5.2.3.

### 5.1.12. Распространение ошибок

Здесь мы сделаем несколько замечаний о распространении ошибок округления в алгоритмах, использующих последовательности преобразований Хаусхолдера или Гивенса. Более точно, пусть дана матрица  $A = A_0 \in \mathbb{R}^{m \times n}$  и матрицы  $A_1, \dots, A_p = B$

вычисляются по формуле

$$A_k = f l(\hat{Q}_k A_{k-1} \hat{Z}_k), \quad k = 1 : p. \quad (5.1.11)$$

Предположим, что описанные выше алгоритмы Хаусхолдера и Гивенса используются как для получения, так и для применения матриц  $\hat{Q}_k$  и  $\hat{Z}_k$ . Пусть  $Q_k$  и  $Z_k$  – ортогональные матрицы, которые были бы получены в отсутствие ошибок округления. Можно показать, что

$$B = (Q_p \dots Q_1) (A + E) (Z_1 \dots Z_p), \quad (5.1.12)$$

где  $\|E\|_2 \leq c \|A\|_2$ , причем константа  $c$  не очень сильно зависит от  $n, m$  и  $p$ . Иными словами,  $B$  – результат точного ортогонального преобразования матрицы, близкой к  $A$ .

### 5.1.13. Быстрые вращения

Способность вращений Гивенса обнулять элементы матриц избирательно делает их важным инструментом в задачах с определенной структурой. Это стимулировало разработку процедур быстрых вращений. Идея быстрых вращений сводится к хитроумному представлению матрицы  $Q$ , являющейся произведением вращений Гивенса. Именно,  $Q$  представляется парой матриц  $(M, D)$ , где  $M^T M = D = \text{diag}(d_i)$  и все  $d_i$  положительны. Матрицы  $Q$ ,  $M$  и  $D$  связаны формулой

$$Q = M D^{-1/2} = M \text{diag}(1/\sqrt{d_i}). \quad (5.1.13)$$

Обратим внимание, что  $(M D^{-1/2})^T (M D^{-1/2}) = D^{-1/2} D D^{-1/2} = I$ , так что матрица  $M D^{-1/2}$  ортогональна.

Для разъяснения деталей лучше всего спуститься на уровень  $2 \times 2$ -матриц. Пусть заданы  $x = (x_1 \ x_2)^T$  и  $D = \text{diag}(d_1, d_2)$ . Допустим, что  $d_1$  и  $d_2$  положительны. Положим по определению

$$M_1 = \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix} \quad (5.1.14)$$

и заметим, что

$$M_1^T x = \begin{bmatrix} \beta_1 x_1 + x_2 \\ x_1 + \alpha_1 x_2 \end{bmatrix}$$

и

$$M_1^T D M_1 = \begin{bmatrix} d_2 + \beta_1^2 d_1 & d_1 \beta_1 + d_2 \alpha_1 \\ d_1 \beta_1 + d_2 \alpha_1 & d_1 + \alpha_1^2 d_2 \end{bmatrix} \equiv D_1.$$

Если  $x_2 \neq 0$ ,  $\alpha_1 = -x_1/x_2$ ,  $\beta_1 = -\alpha_1 d_2/d_1$ , то

$$M_1^T x = \begin{bmatrix} x_2(1 + \gamma_1) \\ 0 \end{bmatrix},$$

$$M_1^T D M_1 = \begin{bmatrix} d_2(1 + \gamma_1) & 0 \\ 0 & d_1(1 + \gamma_1) \end{bmatrix},$$

где  $\gamma_1 = -\alpha_1 \beta_1 = (d_2/d_1)(x_1/x_2)^2$ .

Аналогично, предположив  $x_1 \neq 0$  и определив  $M_2$  как

$$M_2 = \begin{bmatrix} 1 & \alpha_2 \\ \beta_2 & 1 \end{bmatrix}, \quad (5.1.15)$$

где  $\alpha_2 = -x_2/x_1$  и  $\beta_2 = -(d_1/d_2)\alpha_2$ , получим

$$M_2^T x = \begin{bmatrix} x_1(1 + \gamma_2) \\ 0 \end{bmatrix}$$

и

$$M_2^T D M_2 = \begin{bmatrix} d_1(1 + \gamma_2) & 0 \\ 0 & d_2(1 + \gamma_2) \end{bmatrix} \equiv D_2,$$

где  $\gamma_2 = -\alpha_2\beta_2 = (d_1/d_2)(x_2/x_1)^2$ .

Легко показать, что для  $i = 1, 2$  матрица  $j = D^{1/2} M_i D_i^{-1/2}$  ортогональна, причем вторая компонента вектора  $J^T(D^{-1/2} x)$  равна нулю. (Фактически  $J$  может оказаться отражением, так что популярный термин «быстрые вращения» верен лишь отчасти.)

Заметим, что  $\gamma_i$  удовлетворяет соотношению  $\gamma_1\gamma_2 = 1$ . Поэтому в вышеприведенных формулах всегда можно выбрать  $M_i$  таким образом, что «коэффициент роста»  $(1 + \gamma_i)$  будет ограничен числом 2. Матрицы вида

$$M_1 = \begin{bmatrix} \beta_1 & 1 \\ 1 & \alpha_1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & \alpha_2 \\ \beta_2 & 1 \end{bmatrix},$$

удовлетворяющие  $-1 \leq \alpha_i, \beta_i \leq 0$ , суть быстрые преобразования Гивенса размера  $2 \times 2$ . Обратим внимание, что умножение справа на матрицу быстрого вращения требует только половину того количества умножений, которое необходимо для осуществления обычного преобразования Гивенса. Кроме того, обнуление выполняется без явного вычисления квадратного корня, что может оказаться существенным при проектировании СБИС.

В  $n \times n$ -случае требуемые изменения почти такие же, как для обычных вращений Гивенса. Преобразования первого типа имеют вид

$$F(i, k, \alpha, \beta) = \left[ \begin{array}{ccccccccc} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \beta & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & 1 & \cdots & \alpha & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{array} \right]_{\begin{array}{c} i \\ k \end{array}}, \quad (5.1.16)$$

а структура преобразований второго типа такова:

$$F(i, k, \alpha, \beta) = \left[ \begin{array}{ccccccccc} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 1 & \cdots & \alpha & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \beta & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{array} \right]_{\begin{array}{c} i \\ k \end{array}}. \quad (5.1.17)$$

Таким образом, если  $M \in \mathbf{R}^{n \times n}$  и  $D = \text{diag}(d_i)$  удовлетворяют соотношению  $M^T M = D$  и  $F - n \times n$ -матрица быстрого вращения, где  $F^T D F = D_{\text{new}}$  диагональна, то  $M_{\text{new}}^T M_{\text{new}} = D_{\text{new}}$ , где  $M_{\text{new}} = MF$ . Тем самым представление  $(M, D)$  для быстрого вращения можно модифицировать, получив  $(M_{\text{new}}, D_{\text{new}})$ .

Функции **givens** будет соответствовать

**Алгоритм 5.1.8.** По  $x \in \mathbf{R}^2$  и положительному  $d \in \mathbf{R}^2$  следующий алгоритм вычисляет  $2 \times 2$ -матрицу быстрого вращения  $M$ , такую, что вторая компонента вектора  $M^T x$  нулевая и матрица  $M^T D M = D_1$  диагональна, где  $D = \text{diag}(d_1, d_2)$ . Если  $\text{type} = 1$ , то  $M$  имеет вид (5.1.14), а при  $\text{type} = 2$  – вид (5.1.15). Диагональные элементы матрицы  $D_1$  записываются на место вектора  $d$ .

```
function: [α, β, type] = fast.givens (x,d)
    if x(2) ≠ 0
        α = -x(1)/x(2); β = -α d(2)/d(1); γ = -α β
        if γ ≤ 1
            type = 1
            τ = d(1); d(1) = (1 + γ) d(2);
            d(2) = (1 + γ) τ
        else
            type = 2
            α = 1/α; β = 1/β; γ = 1/γ
            d(1) = (1 + γ) d(1); d(2) = (1 + γ) d(2)
        end
    else
        type = 2
        α = 0; β = 0
    end
end fast.givens
```

Алгоритм применения быстрого преобразования Гивенса схож с **row.rot** и **col.rot**. Для преобразования строк мы имеем:

**Алгоритм 5.1.9.** По  $A \in \mathbf{R}^{2 \times q}$  и матрице быстрого вращения  $M$  типа 1 или 2, следующий алгоритм записывает  $M^T A$  на место  $A$ .

```
function: A = row.fast.rot (A, α, β, type)
    if type = 1
        for j = 1 : q
            τ₁ = A(1, j); τ₂ = A(2, j)
            A(1, j) = β τ₁ + τ₂
            A(2, j) = τ₁ + α τ₂
        end
    else
        for j = 1 : q
            τ₁ = A(1, j); τ₂ = A(2, j)
            A(1, j) = τ₁ + β τ₂
            A(2, j) = α τ₁ + τ₂
        end
    end
end row.fast.rot
```

Этот алгоритм требует  $4q$  флоков. Заметим, что формула

$$A([i \ k], :) = \text{row.fast.rot} (A([i \ k], :), \alpha, \beta, \text{type})$$

осуществляет преобразование  $A \leftarrow F(i, k, \alpha, \beta)^T A$  соответствующего типа.

Хотя `fast.givens` и выбирает подходящий тип преобразования, коэффициент роста  $1 + \gamma$  все же может достигать двух. Тем самым после  $s$  преобразований элементы матриц  $D$  и  $M$  могут увеличиться в  $2^s$  раз. Это означает, что по ходу процедуры быстрых вращений необходимо все время отслеживать значения диагональных элементов  $D$ , чтобы избежать переполнения. Такие нетривиальные дополнительные расходы способны ухудшить производительность. Тем не менее рост элементов  $M$  и  $D$  не является неуправляемым, поскольку всегда матрица  $M\hat{D}^{-1/2}$  ортогональна. Свойства быстрых вращений по отношению к ошибкам округления соответствуют нашим ожиданиям от использования матриц Гивенса. Например, если вычислить  $\hat{Q} = fI(\hat{M}\hat{D}^{-1/2})$ , где  $\hat{M}$  и  $\hat{D}$  – вычисленные  $M$  и  $D$ , то  $\hat{Q}$  ортогональна с рабочей точностью:  $\|\hat{Q}^T \hat{Q} - I\|_2 \approx u$ .

### Задачи

**5.1.1.** Выполните `house` для  $x = (1, 7, 2, 3, -1)^T$ .

**5.1.2.** Пусть  $x$  и  $y$  – ненулевые векторы из  $\mathbb{R}^n$ . Дайте алгоритм нахождения такой матрицы Хаусхолдера  $P$ , что вектор  $Px$  кратен  $y$ .

**5.1.3.** Пусть  $x \in \mathbb{C}^n$  и  $x_1 = |x_1|e^{i\theta}, \theta \in \mathbb{R}$ . Предположим, что  $x \neq 0$ , и определим  $u = x + e^{i\theta} \|x\|_2 e_1$ . Покажите, что матрица  $P = I - 2uu^H/u^H$  – унитарна и  $Px = -e^{i\theta} \|x\|_2 e_1$ .

**5.1.4.** Используя матрицы Хаусхолдера, покажите, что  $\det(I + xy^T) = 1 + x^Ty$ , где  $x$  и  $y$  – заданные  $n$ -векторы.

**5.1.5.** Пусть  $x \in \mathbb{C}^2$ . Дайте алгоритм нахождения унитарной матрицы вида

$$Q = \begin{bmatrix} c & \bar{s} \\ -s & c \end{bmatrix}, \quad c \in \mathbb{R}, \quad c^2 + |s|^2 = 1,$$

такой, что вторая компонента вектора  $Q^H x$  нулевая.

**5.1.6.** Пусть  $x$  и  $y$  – единичные векторы в  $\mathbb{R}^n$ . Используя преобразования Гивенса, постройте алгоритм, вычисляющий такую ортогональную матрицу  $Q$ , что  $Q^T x = y$ .

**5.1.7.** Найдите  $c = \cos(\theta)$  и  $s = \sin(\theta)$ , такие, что

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} 5 \\ 12 \end{bmatrix} = \begin{bmatrix} 13 \\ 0 \end{bmatrix}.$$

**5.1.8.** Допустим, что матрица  $Q = I + YTY^T$  ортогональна, где  $Y \in \mathbb{R}^{n \times j}$ ;  $T \in \mathbb{R}^{j \times j}$  – верхняя треугольная матрица. Покажите, что если  $Q_+ = QP$ , где  $R = I - 2vv^T/v^T$  – матрица Хаусхолдера, то  $Q_+$  может быть представлена в виде  $Q_+ = I + Y_+ T_+ Y_+$ , где  $Y_+ \in \mathbb{R}^{n \times (j+1)}$ ;  $T_+ \in \mathbb{R}^{(j+1) \times (j+1)}$  – верхняя треугольная.

**5.1.9.** Опишите детальную реализацию алгоритма 5.1.4, предположив, что значимая часть  $j$ -го хаусхолдерова вектора,  $v^{(j)}(j+1:n)$ , хранится в  $A(j+1:n, j)$ . Поскольку  $Y$  эффективно представляется в  $A$ , ваша процедура должна лишь построить матрицу  $W$ .

**5.1.10.** Покажите, что если матрица  $S$  кососимметрична ( $S^T = -S$ ), то  $Q = (I + S)(I - S)^{-1}$  ортогональна. ( $Q$  называется *преобразованием Кэли* матрицы  $S$ .) Постройте такую матрицу  $S$  ранга 2, что для любого вектора  $x$  лишь первая компонента  $Q^T x$  будет, возможно, отлична от нуля.

### Замечания и литература к § 5.1

Описанные в этом разделе преобразования широко используются как блоки, из которых собираются более сложные алгоритмы. Поэтому важно, чтобы на этом элементарном уровне вычисления выполнялись аккуратно. См. статью

Lawson C. L., Hanson R. J., Kincaid D. R., and Krogh F. T. (1979). "Basic Linear Algebra Subprograms for Fortran Usage", ACM Trans. Math. Soft. 5, 308–23.

Элементарные процедуры линейной алгебры (BLAS) детально описаны в пакете Linpack.

Хаусхолдеровы матрицы названы так в честь А. Хаусхольдера, который первым стал широко использовать их в численном анализе. Однако свойства этих матриц были известны достаточно давно. См.

Turnbull H. W. and Aitken A. C. (1961). *An Introduction to the Theory of Canonical Matrices*, Dover Publications, New York, pp. 102–5.

Другая литература по преобразованиям Хаусхольдера включает

Cuppen J. J. M. (1984). "On Updating Triangular Products of Householder Matrices", Numer. Math. 45, 403–10.

Gourlay A. R. (1970). "Generalization of Elementary Hermitian Matrices", Comp. J. 13, 411–12.

Kaufman L. (1987). "The Generalized Householder Transformation and Sparse Matrices", Lin. Alg. and Its Appl. 90, 221–234.

Parlett B. N. (1971). "Analysis of Algorithms for Reflections in Bisectors", SIAM Review 13, 197–208.

Tsao N. K. (1975). "A Note on Implementing the Householder Transformations", SIAM J. Numer. Anal. 12, 53–58.

Детальный анализ ошибок округления для преобразований Хаусхольдера проведен в книге Лоусона и Хенсона (SLS, pp. 83–89).

Вот основные статьи по блочным преобразованиям Хаусхольдера и относящимся к ним вычислениям:

Bischof C. H. and Van Loan C. (1987). "The WY Representation for Products of Householder Matrices", SIAM J. Sci. and Stat. Comp. 8, s2–s13.

Higham N. J. and Schreiber R. S. (1988). "Fast Polar Decomposition of an Arbitrary Matrix", Report 88–942, Dept. of Computer Science, Cornell University, Ithaca, NY 14853.

Schreiber R. and Parlett B. N. (1987). "Block Reflectors: Theory and Computation", SIAM J. Numer. Anal. 25, 189–205.

Schreiber R. and Van Loan C. (1989). "A Storage Efficient WY Representation for Products of Householder Transformations", SIAM J. Sci. and Stat. Comp. to appear.

Вращения Гивенса, названные в честь У. Гивенса, именуются также вращениями Якоби. Основываясь на этих преобразованиях, Якоби в 1846 г. разработал алгоритм для решения симметричной проблемы собственных значений. Обсужденная в нашем тексте схема хранения вращений Гивенса описана в деталях в статье

Stewart G. W. (1976). "The Economical Storage of Plane Rotations", Numer. Math. 25, 137–138.

Быстрые вращения называются также преобразованиями Гивенса без вычисления квадратного корня. (Вспомним, что стандартный порядок вычисления преобразования Гивенса включает вычисление квадратного корня.) Быстрые вращения могут быть организованы несколькими способами. См.

Gentleman M. (1973). "Least Squares Computations by Givens Transformations without Square Roots", J. Inst. Math. Appl. 12, 329–336.

Hammarling S. (1974). "A Note on Modifications to the Givens Plane Rotation", J. Inst. Math. Appl. 13, 215–218.

Van Loan C. F. (1973). "Generalized Singular Values With Algorithms and Applications", Ph. D. thesis, University of Michigan, Ann Arbor.

Wilkinson J. H. (1977). "Some Recent Advances in Numerical Linear Algebra", in *The State of the Art in Numerical Analysis*, ed. D. A. H. Jacobs, Academic Press, New York, pp. 1–53.

Мы хотим еще раз повторить, что успешная реализация быстрых вращений требует нетривиальных затрат на контроль за процессом.

## 5.2. QR-разложение

В этом разделе мы покажем, как применить преобразования Гивенса и Хаусхольдера для получения различных разложений матрицы. Мы начинаем с QR-разложе-

ния, которое для  $m \times n$ -матрицы  $A$  выглядит так:

$$A = Q R,$$

где  $Q \in \mathbf{R}^{m \times m}$  – ортогональная,  $R \in \mathbf{R}^{m \times n}$  – верхняя треугольная матрица. В этом разделе мы предполагаем  $m \geq n$ . Если  $A$  имеет полный столбцовый ранг, то первые  $n$  столбцов матрицы  $Q$  образуют ортонормированный базис подпространства  $\text{range}(A)$ . Тем самым QR-разложение дает один из способов получения ортонормированного базиса для набора векторов. Существует несколько путей вычисления QR-разложения. Мы изложим методы, основанные на преобразованиях Хаусхолдера и Гивенса, а также на быстрых вращениях и блочных отражениях. Кроме того, мы обсуждаем процесс ортогонализации Грама–Шмидта и его модификацию, отличающуюся большей численной устойчивостью.

### 5.2.1. QR-разложение: преобразования Хаусхолдера

Мы начинаем с метода, использующего для получения QR-разложения преобразования Хаусхолдера. Суть алгоритма может быть пояснена на простом примере. Пусть  $m = 6$ ,  $n = 5$  и матрицы Хаусхолдера  $H_1$  и  $H_2$  таковы, что

$$H_2 H_1 A = \begin{bmatrix} \times & \times & \cdot & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \end{bmatrix}.$$

Займемся теперь элементами, выделенными жирным шрифтом. Найдем хаусхолдерову матрицу  $\tilde{H}_3 \in \mathbf{R}^{4 \times 4}$ , такую, что

$$\tilde{H}_3 \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix} = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Если  $H_3 = \text{diag}(I_2, \tilde{H}_3)$ , то

$$H_3 H_2 H_1 A = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}$$

Выполнив  $n$  таких шагов, мы получаем верхнюю треугольную матрицу  $H_n H_{n-1} \dots H_1 A = R$ , так что, положив  $Q = H_1 \dots H_n$ , имеем  $A = Q R$ . Запишем изложенное формально.

**Алгоритм 5.2.1 (QR-разложение: преобразования Хаусхолдера).** По матрице  $A \in \mathbf{R}^{m \times n}$ ,  $m \geq n$ , следующий алгоритм находит хаусхолдеровы матрицы  $H_1, \dots, H_n$ , такие, что при  $Q = H_1 \dots H_n$  матрица  $Q^T A = R$  верхняя треугольная. При этом верхний треугольник матрицы  $R$  записывается на место верхнего треуголь-

ника матрицы  $A$ , а компоненты  $j+1:m$   $j$ -го хаусхолдерова вектора хранятся в  $A(j+1:m, j)$ ,  $j < m$ .

```

for  $j = 1:n$ 
     $v(j:m) = \text{house}(A(j:m, j))$ 
     $A(j:m, j:n) = \text{row.house}(A(j:m, j:n), v(j:m))$ 
    if  $j < m$ 
         $A(j+1:m, j) = v(j+1:m)$ 
    end
end
```

Этот алгоритм требует  $2n^2(m - n/3)$  флопов.

Покажем наглядно, что именно записывается на место матрицы  $A$ . Если

$$v^{(j)} = [\underbrace{0, \dots, 0}_{j-1}, 1, v_{j+1}^{(j)}, \dots, v_m^{(j)}]^T$$

есть  $j$ -й хаусхолдеров вектор, то по завершении алгоритма

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ v_2^{(1)} & r_{22} & r_{23} & r_{24} & r_{25} \\ v_3^{(1)} & v_3^{(2)} & r_{33} & r_{34} & r_{35} \\ v_4^{(1)} & v_4^{(2)} & v_4^{(3)} & r_{44} & r_{45} \\ v_5^{(1)} & v_5^{(2)} & v_5^{(3)} & v_5^{(4)} & r_{55} \\ v_6^{(1)} & v_6^{(2)} & v_6^{(3)} & v_6^{(4)} & v_6^{(5)} \end{bmatrix}.$$

Если необходимо явно получить матрицу  $Q = H_1 \dots H_n$ , то это можно делать по ходу алгоритма, используя (5.1.5). Дополнительные затраты составляют  $4(m^2n - mn^2 + n^3/3)$  флопов.

Вычисленная верхняя треугольная матрица  $\hat{R}$  есть точная  $R$  для некоторой близкой к  $A$  матрицы, в том смысле что  $Z^T(A + E) = \hat{R}$ , где  $Z$  – некоторая ортогональная (точно) матрица, и  $\|E\|_2 \approx \|A\|_2 u$ .

## 5.2.2. QR-разложение: метод блочных отражений

В алгоритме 5.2.1 много операций уровня 2: умножений матриц на векторы и модификаций внешним произведением. Реорганизация вычислений с использованием введенного в § 5.1.7 блочного представления произведений хаусхолдеровых матриц позволяет получить процедуру уровня 3. Идея состоит в применении преобразований Хаусхолдера группами, или кластерами, представленными в  $WY$ -форме (см. § 5.1.7).

Для иллюстрации этой основной идеи рассмотрим маленький пример. Пусть  $n = 12$  и «блочный параметр»  $r = 3$ . На первом шаге генерируются матрицы Хаусхолдера  $H_1$ ,  $H_2$  и  $H_3$ , как в алгоритме 5.2.1. Но в отличие от алгоритма 5.2.1, где  $H_i$  применяются сразу ко всей матрице  $A$ , мы применим  $H_1$ ,  $H_2$  и  $H_3$  лишь к  $A(:, 1:3)$ . После этого мы строим блочное представление  $H_1 H_2 H_3 = I + W_1 Y_1^T$  и осуществляем модификацию уровня 3:

$$A(:, 4:12) = (I + W_1 Y_1^T) A(:, 4:12).$$

Далее генерируем, как в алгоритме 5.2.1,  $H_4$ ,  $H_5$  и  $H_6$ . Однако эти преобразования применяются к  $A(:, 7:12)$  лишь после нахождения блочного представления

$H_4 H_5 H_6 = I + W_2 Y_2^T$ . Общая схема выглядит так:

$\lambda = 1; k = 0$

**while**  $\lambda \leq n$

$\tau = \min(\lambda + r - 1, n); k = k + 1$

Используя алгоритм 5.2.1, привести к верхнему треугольному виду

$$A(\lambda:m, \lambda:n), \text{ построив хаусхолдеровы матрицы } H_\lambda, \dots, H_t. \quad (5.2.1)$$

Используя алгоритм 5.1.4, получить блочное представление  $I + W_k Y_k = H_\lambda \dots H_t$

$$A(\lambda:m, \tau+1:n) = (I + W_k Y_k^T)^T A(\lambda:m, \tau+1:n)$$

$$\lambda = \tau + 1$$

**end**

Структура нулей хаусхолдеровых векторов, задающих матрицы  $H_\lambda, \dots, H_t$ , влечет за собой равенство нулю первых  $\lambda - 1$  строк матриц  $W_k$  и  $Y_k$ . Этот факт следует использовать в практической реализации.

Запись (5.2.1) естественно рассматривать, разбив матрицу  $A$  на блочные столбцы

$$A = [A_1, \dots, A_N], \quad N = \text{ceil}(n/r),$$

где блочный столбец  $A_k$  обрабатывается на  $k$ -м шагу. На  $k$ -м шагу (5.2.1) строится блочное преобразование Хаусхолдера, обнуляющее поддиагональную часть  $A_k$ . После этого модифицируются остающиеся блочные столбцы.

Свойства (5.2.1) по отношению к ошибкам округления по существу те же, что и для алгоритма 5.2.1. Требуемое количество флопов несколько увеличивается вследствие необходимости вычислять матрицу  $W$ . Однако в результате нашей группировки почти все флопы связаны с перемножением матриц. Именно, доля операций уровня 3 в (5.2.1) составляет примерно  $1 - 2/N$ . Другие детали см. Bischof, Van Loan (1987).

### 5.2.3. QR-разложение: преобразования Гивенса

Вращения Гивенса также могут быть использованы для вычисления QR-разложения. Проиллюстрируем общую идею на примере  $4 \times 3$ -матриц:

$$\begin{array}{c} \left[ \begin{array}{ccc} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{array} \right] \xrightarrow{(3,4)} \left[ \begin{array}{ccc} x & x & x \\ x & x & x \\ x & x & x \\ 0 & x & x \end{array} \right] \xrightarrow{(2,3)} \left[ \begin{array}{ccc} x & x & x \\ x & x & x \\ 0 & x & x \\ 0 & x & x \end{array} \right] \xrightarrow{(1,2)} \\ \left[ \begin{array}{ccc} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \end{array} \right] \xrightarrow{(3,4)} \left[ \begin{array}{ccc} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & 0 & x \end{array} \right] \xrightarrow{(2,3)} \left[ \begin{array}{ccc} x & x & x \\ 0 & x & x \\ 0 & 0 & x \\ 0 & 0 & x \end{array} \right] \xrightarrow{(3,4)} R \end{array}$$

Здесь мы показали 2-векторы, определяющие соответствующие вращения Гивенса. Ясно, что если обозначить  $G_j$ -е вращение в этом приведении, то матрица  $Q^T A = R$  будет верхней треугольной, где  $Q = G_1 \dots G_t$  и  $t$  – общее количество вращений. Для произвольных  $m$  и  $n$  имеем:

**Алгоритм 5.2.2 (QR-разложение: преобразования Гивенса).** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ , следующий алгоритм записывает на место  $A$  верхнюю треугольную матрицу  $R = Q^T A$ , где  $Q$  ортогональна.

```

for  $j = 1:n$ 
    for  $i = m:-1:j+1$ 
         $[c, s] = \text{givens}(A(i-1, j), A(i, j))$ 
         $A(i-1:i, j:n) = \text{row.rot}(A(i-1:i, j:n), c, s)$ 
    end
end

```

Этот алгоритм требует  $3n^2(m - n/3)$  флопов. Заметим, что пару  $(c, s)$  можно, согласно (5.1.9), закодировать одним числом  $\rho$ , которое может быть сохранено на месте обнуляемого элемента  $A(i, j)$ . Операция типа  $x \leftarrow Q^T x$  может тогда быть реализована с использованием (5.1.10) – нужно только позаботиться о соблюдении надлежащего порядка следования вращений.

Для приведения  $A$  к верхнему треугольному виду годятся и другие последовательности вращений. Например, заменив операторы **for** в алгоритме 5.2.2 на

```

for  $i = m:-1:2$ 
    for  $j = 1:\min\{i-1, n\}$ 

```

мы придем к построчному порядку появления нулей в матрице  $A$ .

Еще один изменяемый параметр нашей процедуры связан с плоскостями, в которых происходят обнуляющие каждый из элементов  $a_{ij}$  вращения. Например, вместо поворота строки  $i-1$  и  $i$  для обнуления  $a_{ij}$  в алгоритме 5.2.2 мы могли бы использовать строки  $j$  и  $i$ , т. е.

```

for  $j = 1:n$ 
    for  $i = m:-1:j+1$ 
         $[c, s] = \text{givens}(A(j, j), A(i, j))$ 
         $A([j \ i], j:n) = \text{row.rot}(A([j \ i], j:n), c, s)$ 
    end
end

```

#### 5.2.4. QR-разложение хессенберговой матрицы с использованием вращений

Как пример использования вращений Гивенса для матриц с определенной структурой мы рассмотрим их применение к вычислению QR-разложения верхней хессенберговой матрицы. Общую идею мы снова иллюстрируем на маленьком примере. Предположим, что  $n = 6$  и после двух шагов мы вычислили

$$G(2, 3, \theta_2)^T G(1, 2, \theta_1)^T A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

Тогда для обнуления элемента  $(4, 3)$  мы вычисляем  $G(3, 4, \theta_3)$ . Имеем

$$G(3, 4, \theta_3)^T G(2, 3, \theta_2)^T G(1, 2, \theta_1)^T A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

В целом мы имеем

**Алгоритм 5.2.3 (QR-разложение хессенберговой матрицы).** Если  $A \in \mathbb{R}^{n \times n}$  – верхняя хессенбергова матрица, то следующий алгоритм записывает на место  $A$  матрицу  $Q^T A = R$ , где  $Q$  – ортогональная,  $R$  – верхняя треугольная матрица.  $Q = G_1 \dots G_{n-1}$  есть произведение вращений Гивенса, причем  $G_j$  имеет вид  $G_j = G(j, j+1, \theta_j)$ .

```

for  $j = 1 : n - 1$ 
     $[c, s] = \text{givens}(A(j, j), A(j+1, j))$ 
     $A(j:j+1, j:n) = \text{row.rot}(A(j:j+1, j:n), c, s)$ 
end
```

Алгоритм требует около  $3n^2$  флопов.

### 5.2.5. QR-разложение: быстрые вращения

Описанные в § 5.1.13 быстрые вращения могут быть использованы для вычисления  $(M, D)$ -представления матрицы  $Q$ . Именно, если  $M$  – невырожденная,  $D$  – диагональная матрица, причем матрица  $M^T A = T$  – верхняя треугольная,  $M^T M = D$  – диагональная, то матрица  $Q = MD^{-1/2}$  ортогональная, а  $Q^T A = D^{-1/2} T \equiv R$  – верхняя треугольная. Вспоминая выписанные в § 5.1.13 функции `fast.givens` и `fast.row.rot`, получаем следующий аналог QR-разложения методом вращений:

**Алгоритм 5.2.4 (QR-разложение: быстрые вращения).** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ , следующий алгоритм вычисляет невырожденную матрицу  $M \in \mathbb{R}^{m \times m}$  и положительный вектор  $d(1:m)$ , такие, что матрица  $M^T A = T$  верхняя треугольная и  $M^T M = \text{diag}(d_1, \dots, d_m)$ . Матрица  $T$  записывается на место матрицы  $A$ .

```

for  $i = 1 : n$ 
     $d(i) = 1$ 
end
for  $j = 1 : n$ 
    for  $i = m : -1 : j + 1$ 
         $[\alpha, \beta, type] = \text{fast.givens}(A(i-1:i, j), d(i-1:i))$ 
         $A(i-1:i, j:n) = \text{fast.row.rot}(A(i-1:i, j:n), \alpha, \beta, type)$ 
    end
end
```

Этот алгоритм требует  $2n^2 (m - n/3)$  флопов. Как отмечалось в предыдущем разделе, в алгоритмах быстрых вращений (подобных вышеприведенному) необходимо принимать меры предосторожности против переполнения. Это означает, что если элементы матриц  $M$ ,  $D$  и  $A$  становятся большими, необходимо периодически проводить перемасштабирование.

Применение быстрых вращений является привлекательным, если необходимо получить QR-разложение ленточной матрицы с узкой лентой, ввиду отсутствия необходимости вычисления квадратных корней. (По этой же причине в § 4.3.7 мы предпочли  $LDL^T$  разложению Холецкого в случае матриц с узкой лентой.) Более конкретно, если  $A \in \mathbb{R}^{m \times n}$  имеет верхнюю ширину ленты  $q$  и нижнюю ширину ленты  $p$ , то  $Q^T A = R$  имеет верхнюю ширину ленты  $p + q$ . В этом случае метод вращений для QR-разложения требует около  $O(np(p+q))$  флопов и  $O(np)$  вычислений квадратных корней. Таким образом, если  $p, q \ll n$ , на вычисление квадратных корней приходится значительная доля общего количества вычислений.

### 5.2.6. Свойства QR-разложения

Описанные выше алгоритмы «доказывают» существование QR-разложения. Здесь мы выявим связь между столбцами матрицы  $Q$  и подпространствами  $\text{range}(A)$ ,  $\text{range}(A)^\perp$  и рассмотрим вопрос о единственности.

**Теорема 5.2.1.** Если  $A = QR$  есть QR-разложение матрицы  $A \in \mathbf{R}^{m \times n}$ , имеющей полный столбцовий ранг, и  $A = [a_1, \dots, a_n]$ ,  $Q = [q_1, \dots, q_m]$  – разбиения на блочные столбцы, то

$$\text{span} \{a_1, \dots, a_k\} = \text{span} \{q_1, \dots, q_k\}, \quad k = 1:n.$$

В частности, если  $Q_1 = Q(1:m, 1:n)$  и  $Q_2 = Q(1:m, n+1:m)$ , то

$$\text{range}(A) = \text{range}(Q_1),$$

$$\text{range}(A)^\perp = \text{range}(Q_2)$$

и  $A = Q_1 R_1$ , где  $R_1 = R(1:n, 1:n)$ .

*Доказательство.* Сравнивая  $k$ -е столбцы в  $A = QR$ , находим, что

$$a_k = \sum_{i=1}^k r_{ik} q_i \in \text{span} \{q_1, \dots, q_k\}. \quad (5.2.2)$$

Тем самым  $\text{span} \{a_1, \dots, a_k\} \subseteq \text{span} \{q_1, \dots, q_k\}$ . Однако из  $\text{rank}(A) = n$  следует, что  $\text{span} \{a_1, \dots, a_k\}$  имеет размерность  $k$ , и поэтому должно совпадать с  $\text{span} \{q_1, \dots, q_k\}$ . Доказательство остающейся части теоремы тривиально. Матрицы  $Q_1 = Q(1:m, 1:n)$  и  $Q_2 = Q(1:m, n+1:m)$  легко вычисляются по факторизованному представлению  $Q$ .

**Теорема 5.2.2.** Пусть  $A \in \mathbf{R}^{m \times n}$  имеет полный столбцовий ранг. Тогда «усеченное» QR-разложение

$$A = Q_1 R_1,$$

где  $Q_1 \in \mathbf{R}^{m \times n}$  имеет ортонормированные столбцы, а  $R_1$  верхняя треугольная матрица с положительными диагональными элементами, будет единственным. Далее  $R_1 = G^T$ , где  $G$  – нижнетреугольный множитель в разложении Холецкого матрицы  $A^T A$ .

*Доказательство.* Так как  $A^T A = (Q_1 R_1)^T (Q_1 R_1) = R_1^T R_1$ , мы видим, что  $G = R_1^T$  является множителем Холецкого матрицы  $A^T A$ . По теореме 4.2.5 этот множитель единствен. Поскольку  $Q_1 = A R_1^{-1}$ ,  $Q_1$  также единствен. Во всех случаях, когда нам необходимо обратиться к «усеченному» QR-разложению какой-нибудь матрицы  $A$ , как в теореме 5.2.2, мы будем записывать  $A = Q_1 R_1$ .

### 5.2.7. Классический метод Грама – Шмидта

Мы переходим к обсуждению двух альтернативных методов прямого вычисления разложения  $A = Q_1 R_1$ . Если  $\text{rank}(A) = n$ , то уравнение (5.2.2) может быть разрешено относительно  $q_k$ :

$$q_k = \left( a_k - \sum_{i=1}^{k-1} r_{ik} q_i \right) / r_{kk}.$$

Таким образом,  $q_k$  можно рассматривать как единичный в 2-норме вектор, направ-

ленный вдоль

$$z_k = a_k - \sum_{i=1}^{k-1} r_{ik} q_i,$$

где для обеспечения  $z_k \in \text{span}\{q_1, \dots, q_{k-1}\}^\perp$  мы выбираем

$$r_{ik} = q_i^T a_k, \quad i = 1 : k-1.$$

Это приводит к классическому алгоритму Грама – Шмидта (CGS) для вычисления  $A = Q_1 R_1$ .

```

for  $k = 1 : n$ 
     $R(1:k-1, k) = Q(1:m, 1:k-1)^T A(1:m, k)$ 
     $z = A(1:m, k) - Q(1:m, 1:k-1) R(1:k-1, k)$ 
     $R(k, k) = \|z\|_2$ 
     $Q(1:m, k) = z / R(k, k)$ 
end

```

На  $k$ -м шаге CGS генерируются  $k$ -е столбцы матриц  $Q$  и  $R$ .

### 5.2.8. Модифицированный метод Грама – Шмидта

К несчастью, классический метод Грама – Шмидта отличается крайне плохими численными свойствами. В типичном случае происходит сильная потеря ортогональности вычисленных векторов  $q_i$ . Любопытно, что другая организация этого же вычисления, известная под названием модифицированный метод Грама – Шмидта (MGS), дает куда лучшую с вычислительной точки зрения процедуру. На  $k$ -м шаге MGS находятся  $k$ -й столбец матрицы  $Q$  (обозначаемый  $q_k$ ) и  $k$ -я строка матрицы  $R$  (обозначаемая  $r_k^T$ ). Для вывода метода MGS определим матрицы  $A^{(k)} \in \mathbb{R}^{m \times (n-k+1)}$  как

$$A - \sum_{i=1}^{k-1} q_i r_i^T = \sum_{i=k}^n q_i r_i^T = [0 \quad A^{(k)}]. \quad (5.2.4)$$

Отсюда следует, что если

$$A^{(k)} = \begin{bmatrix} z & B \\ 1 & n-k \end{bmatrix},$$

то  $r_{kk} = \|z\|_2$ ,  $q_k = z / r_{kk}$  и  $(r_{k,k+1} \dots r_{kn}) = q_k^T B$ . Затем вычисляем внешнее произведение  $A^{(k+1)} = B - q_k (r_{k,k+1} \dots r_{kn})$  и переходим к следующему шагу. Этим  $k$ -й шаг MGS полностью описан.

**Алгоритм 5.2.5 (Модифицированный метод Грама – Шмидта).** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $\text{rank}(A) = n$ , следующий алгоритм вычисляет разложение  $A = Q_1 R_1$ , где столбцы матрицы  $Q_1 \in \mathbb{R}^{m \times n}$  ортонормированы, а матрица  $R_1 \in \mathbb{R}^{n \times n}$  верхняя треугольная.

```

for  $k = 1 : n$ 
     $R(k, k) = \|A(1:m, k)\|_2$ 
     $Q(1:m, k) = A(1:m, k) / R(k, k)$ 
    for  $j = k + 1 : n$ 
         $R(k, j) = Q(1:m, k)^T A(1:m, j)$ 
         $A(1:m, j) = A(1:m, j) - Q(1:m, k) R(k, j)$ 
    end
end

```

Алгоритм требует  $2mn^2$  флопов. Записать и  $Q_1$ , и  $R_1$  на место матрицы  $A$  невозможнo; как правило, вычисления организуют таким образом, что на место  $A$  записывается  $Q_1$ , а матрица  $R_1$  хранится в отдельном массиве.

### 5.2.9. Вычислительные затраты и точность

Если необходимо построить в  $\text{range}(A)$  ортонормированный базис, то при использовании метода Хаусхолдера требуется  $2mn^2 - 2n^3/3$  флопов для получения  $Q$  в факторизованном виде, и еще  $2mn^2 - 2n^3/3$  флопов для вычисления первых  $n$  столбцов  $Q$ . Таким образом, для задачи нахождения ортонормированного базиса в  $\text{range}(A)$  MGS примерно в два раза эффективнее, чем ортогонализация по Хаусхолдеру. Однако было показано (Björck 1967b), что вычисленная методом MGS матрица  $\hat{Q}_1 = [\hat{q}_1, \dots, \hat{q}_n]$  удовлетворяет

$$\hat{Q}_1^T \hat{Q}_1 = I + E_{\text{MGS}}, \quad \|E_{\text{MGS}}\|_2 \approx \mathbf{u} \mathbf{x}_2(A),$$

в то время как в методе Хаусхолдера соответствующий результат имеет вид

$$\hat{Q}_1^T \hat{Q}_1 = I + E_n, \quad \|E_n\|_2 \approx \mathbf{u}.$$

Поэтому если для нас критична ортонормированность, метод MGS следует применять для вычисления ортонормированного базиса лишь в тех случаях, когда исходные векторы достаточно независимы.

**Пример 5.2.1.** Применение модифицированного метода Грама – Шмидта к матрице

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \end{bmatrix}, \quad \mathbf{x}_2(A) \approx 1.4 \cdot 10^3,$$

с 6-разрядной десятичной арифметикой дает

$$[\hat{q}_1 \hat{q}_2] = \begin{bmatrix} 1.00000 & 0 \\ 0.001 & -707107 \\ 0 & 0.707100 \end{bmatrix}.$$

#### Задачи

**5.2.1.** Адаптируйте алгоритм QR-разложения методов Хаусхолдера на случай ленточной матрицы  $A \in \mathbb{R}^{m \times n}$  с нижней шириной ленты  $p$  и верхней шириной ленты  $q$ .

**5.2.2.** Модифицируйте алгоритм QR-разложения методом Хаусхолдера так, чтобы он вычислял разложение  $A = QL$ , где  $L$  нижняя треугольная, а  $Q$  ортогональная матрица. Считайте, что матрица  $A$  квадратная. Функция вычисления хаусхолдерова вектора  $v = \text{house}(x)$  должна быть при этом переписана так, чтобы все компоненты вектора  $(I - 2vv^T/v^T)v$ , кроме последней, равнялись нулю.

**5.2.3.** Модифицируйте алгоритм QR-разложения методом Гивенса так, чтобы нули вносились в матрицу по диагоналям. То есть элементы обнуляются в порядке  $(m, 1), (m-1, 1), (m, 2), (m-2, 1), (m-1, 2), (m, 3)$  и т. д.

**5.2.4.** Адаптируйте алгоритм QR-разложения методом быстрых вращений для случая трехдиагональной  $n \times n$ -матрицы  $A$ . Считайте, что поддиагональ, диагональ и наддиагональ матрицы  $A$  хранятся в  $e(1:n-1)$ ,  $a(1:n)$ ,  $f(1:n-1)$ , соответственно. Организуйте вычисления таким образом, чтобы на место этих векторов записалась бы ненулевая часть матрицы  $T$ .

**5.2.5.** Пусть  $L \in \mathbb{R}^{m \times n}$ ,  $m \geq n$  – нижняя треугольная матрица. Покажите, как можно использовать матрицы Хаусхолдера  $H_1, \dots, H_n$  для нахождения такой нижней треугольной матрицы  $L_1 \in \mathbb{R}^{n \times n}$ , что

$$H_n \dots H_1 L = \begin{bmatrix} L_1 \\ 0 \end{bmatrix}.$$

Подсказка: Для случая  $6 \times 3$  второй шаг связан с нахождением такой матрицы  $H_2$ , что

$$H_2 \begin{bmatrix} x & 0 & 0 \\ x & x & 0 \\ x & x & x \\ x & x & 0 \\ x & x & 0 \\ x & x & 0 \end{bmatrix} = \begin{bmatrix} x & 0 & 0 \\ x & x & 0 \\ x & x & x \\ x & 0 & 0 \\ x & 0 & 0 \\ x & 0 & 0 \end{bmatrix}$$

при условии, что строки 1 и 3 не изменяются.

### 5.2.6. Покажите, что если

$$A = \begin{bmatrix} R & w \\ 0 & v \\ k & n-k \end{bmatrix} \quad \begin{matrix} k \\ m \\ k \end{matrix}, \quad b = \begin{bmatrix} c \\ d \\ k \end{bmatrix} \quad \begin{matrix} k \\ m-k \end{matrix}$$

и матрица  $A$  имеет полный ранг, то  $\min \|Ax - b\|_2^2 = \|d\|_2^2 - (v^T d / \|v\|_2)^2$ .

**5.2.7.** Пусть  $A \in \mathbb{R}^{n \times n}$ ,  $D = \text{diag}(d_1, \dots, d_n) \in \mathbb{R}^{n \times n}$ . Укажите, как построить такую ортогональную матрицу  $Q$ , что матрица  $Q^T A - D Q^T = R$  является верхней треугольной. Не заботьтесь об эффективности – это просто упражнение на манипуляцию с QR-разложением.

### Замечания и литература к § 5.2

Идея использования преобразований Хаусхолдера для решения задачи наименьших квадратов была предложена в

Householder A.S. (1958). “Unitary Triangularization of a Nonsymmetric Matrix”, J. ACM 5, 339–342.

Проработка практических деталей была проведена в

Businger P. and Golub G.H. (1965). “Linear Least Squares Solutions by Householder Transformations”, Numer. Math. 7, 269–76. См. также HACLA, pp. 111–118.

Golub G.H. (1965). “Numerical methods for Solving Linear Least Squares Problems”, Numer. Math. 7, 206–216.

Основные ссылки по QR-разложению методом Гивенса включают

Gentleman M. (1973). “Error Analysis of QR Decompositions by Givens Transformations”, Lin. Alg. and Its Appl. 10, 189–197.

Givens W. (1958). “Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form”, SIAM J. App. Math. 6, 26–50.

Обсуждение использования QR-разложения для решения многих задач статистических вычислений можно найти в

Golub G.H. (1969). “Matrix Decompositions and Statistical Computation”, in Statistical Computation, ed. Milton R.C. and Nelder J.A., Academic Press, New York, pp. 365–97.

Как изменяется QR-разложение при возмущении матрицы  $A$ ? Этот вопрос исследован в

Stewart G.W. (1977). “Perturbation Bounds for the QR Factorization of a Matrix”, SIAM J. Numer. Anal. 14, 509–518.

Основной результат – изменения в  $Q$  и  $R$  ограничены произведением обусловленности  $A$  на относительное возмущение  $A$ .

Различные аспекты процесса Грама – Шмидта обсуждаются в

Abdelmalek N.N. (1971). “Roundoff Error Analysis for Gram-Schmidt Method and Solutions of Linear Least Squares Problems”, BIT 11, 345–368.

Björck A. (1976b). “Solving Linear Least Squares Problems by Gram-Schmidt Orthogonalization”, Math. Comp. 20, 325–328.

QR-разложение матрицы с определенной структурой обычно само структурировано. См.

Bojanczyk A.W., Brent R.P., and de Hoog F.R. (1986). “QR Factorization of Toeplitz Matrices”, Numer. Math. 49, 81–94.

## 5.3. Задача наименьших квадратов: случай полного ранга

Рассмотрим задачу нахождения вектора  $x$ , удовлетворяющего уравнению  $Ax = b$ , где *матрица данных*  $A \in \mathbb{R}^{m \times n}$  и *вектор наблюдений*  $b \in \mathbb{R}^m$  заданы, причем  $m \geq n$ . Когда уравнений больше, чем неизвестных, мы говорим, что система  $Ax = b$  *переопределена*. Обычно переопределенная система не имеет точного решения, поскольку  $b$  должен принадлежать  $\text{range}(A)$ , являющемуся собственным подпространством  $\mathbb{R}^m$ .

Чтобы все же как-то решить задачу, попытаемся минимизировать  $\|Ax - b\|_p$  для некоторого подходящего  $p$ . Оптимальные решения будут различными в разных нормах. Например, если  $A = [1 \ 1 \ 1]^T$ ,  $b = (b_1 \ b_2 \ b_3)^T$ , где  $b_1 \geq b_2 \geq b_3 \geq 0$ , то можно проверить, что

$$\begin{aligned} p = 1 &\Rightarrow x_{opt} = b_2, \\ p = 2 &\Rightarrow x_{opt} = (b_1 + b_2 + b_3)/3, \\ p = \infty &\Rightarrow x_{opt} = (b_1 + b_3)/2. \end{aligned}$$

Минимизация в 1-норме и в  $\infty$ -норме затруднена тем, что функция  $f(x) = \|Ax - b\|_p$  не дифференцируема для этих значений  $p$ . Несмотря на это, в этой области был достигнут существенный прогресс, и для минимизации в 1-норме и в  $\infty$ -норме имеется несколько хороших методик. См. Bartels, Conn, Sinclair (1978), Bartels, Conn, Charalambous (1978).

В отличие от минимизации в произвольной  $p$ -норме, задача *наименьших квадратов* (задача LS)

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 \quad (5.3.1)$$

лучше поддается решению по двум причинам:

- $\psi(x) = \frac{1}{2}\|Ax - b\|_2^2$  – дифференцируемая функция  $x$ , поэтому значение  $x$  в точке минимума удовлетворяет градиентному уравнению  $\nabla \psi(x) = 0$ . Это уравнение оказывается легко строящейся симметричной линейной системой, положительно определенной, если  $A$  имеет полный столбцовый ранг.
- 2-норма сохраняется ортогональными преобразованиями. Это означает, что мы имеем право искать такую ортогональную матрицу  $Q$ , что эквивалентная задача минимизации  $\|(Q^T A)x - (Q^T b)\|_2$  решается «легко».

В этом разделе мы проследим эти два подхода к решению для случая, когда  $A$  имеет полный столбцовый ранг. Будут подробно рассмотрены и сопоставлены методы, основанные на нормальных уравнениях и QR-разложении.

### 5.3.1. Следствия полноты ранга

Пусть  $x \in \mathbb{R}^n$ ,  $z \in \mathbb{R}^n$ ,  $\alpha \in \mathbb{R}$ . Рассмотрим равенство

$$\|A(x + \alpha z) - b\|_2^2 = \|Ax - b\|_2^2 + 2\alpha z^T A^T(Ax - b) + \alpha^2 \|Az\|_2^2,$$

где  $A \in \mathbb{R}^{m \times n}$ ,  $b \in \mathbb{R}^m$ . Если  $x$  является решением задачи LS (5.3.1), то должно выполняться  $A^T(Ax - b) = 0$ . В противном случае, взяв  $z = -A^T(Ax - b)$  и сделав  $\alpha$  достаточно малым, мы сможем получить противоречашее неравенство  $\|A(x + \alpha z) - b\|_2 \leq \|Ax - b\|_2$ . Можно также заключить, что если  $x$  и  $x + \alpha z$  являются решениями задачи LS, то  $z \in \text{null}(A)$ .

Таким образом, если  $A$  имеет полный столбцовый ранг, задача LS имеет

единственное решение  $x_{LS}$ , удовлетворяющее симметричной положительно определенной линейной системе

$$A^T A x_{LS} = A^T b.$$

Составляющие ее уравнения называются *нормальными уравнениями*. Поскольку  $\nabla \psi(x) = A^T(Ax - b)$ , где  $\psi(x) = \frac{1}{2}\|Ax - b\|_2^2$ , мы видим, что решение нормальных уравнений эквивалентно решению градиентного уравнения  $\nabla \psi = 0$ .

Назовем

$$r_{LS} = b - Ax_{LS}$$

*минимальной невязкой* и будем использовать обозначение

$$\rho_{LS} = \|Ax_{LS} - b\|_2$$

для ее величины. Заметим, что если  $\rho_{LS}$  мало, мы можем «предсказать»  $b$  по столбцам  $A$ .

До сих пор мы предполагали, что  $A \in \mathbb{R}^{m \times n}$  имеет полный столбцовый ранг. Это предположение будет отброшено в § 5.5. Однако даже если  $\text{rank}(A) = n$ , мы можем ожидать неприятностей в наших процедурах в случае, когда  $A$  почти неполноранговая.

Оценивая качество вычисленного решения задачи LS, следует иметь в виду два важных вопроса:

- Насколько близко  $\hat{x}_{LS}$  к  $x_{LS}$ ?
- Насколько мало  $r_{LS} = b - A\hat{x}_{LS}$  по сравнению с  $r_{LS} = b - Ax_{LS}$ ?

Относительная важность этих двух критериев меняется от приложения к приложению. Во всяком случае, важно понимать, как возмущения  $A$  и  $b$  воздействуют на  $x_{LS}$  и  $r_{LS}$ . Интуиция подсказывает нам, что если столбцы  $A$  почти линейно зависимы, эти величины могут оказаться весьма чувствительными к возмущениям. Чтобы внести ясность в эти вопросы, нам нужно распространить понятие обусловленности на прямоугольные матрицы.

### 5.3.2. Обусловленность прямоугольных матриц

При анализе задачи наименьших квадратов естественно работать в 2-норме. Напомним (см. § 2.7.3), что в 2-норме обусловленность квадратной невырожденной матрицы равна отношению наибольшего сингулярного числа к наименьшему. Для прямоугольных матриц с полным столбцовым рангом мы примем это свойство за определение:

$$A \in \mathbb{R}^{m \times n}, \quad \text{rank}(A) = n \Rightarrow \kappa_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}.$$

Если столбцы  $A$  почти линейно зависимы, то  $\kappa_2(A)$  велико. Следующий пример дает понять, что для плохо обусловленных задач LS их решения и минимальные невязки весьма чувствительны к возмущениям.

**Пример 5.3.1.** Пусть

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 10^{-6} \\ 0 & 0 \end{bmatrix}, \quad \delta A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 10^{-8} \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \delta b = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

и пусть  $x_{LS}$  и  $\hat{x}_{LS}$  минимизируют  $\|Ax - b\|_2$  и  $\|(A + \delta A)x - (b + \delta b)\|_2$  соответственно. Тогда

$$x_{LS} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \hat{x}_{LS} = \begin{bmatrix} 1 \\ 0.9999 \cdot 10^4 \end{bmatrix}, \quad r_{LS} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \hat{r}_{LS} = \begin{bmatrix} 0 \\ -0.9999 \cdot 10^{-2} \\ 0.9999 \cdot 10^0 \end{bmatrix}.$$

Поскольку  $\kappa_2(A) = 10^6$ , мы имеем

$$\frac{\|\hat{x}_{LS} - x_{LS}\|_2}{\|x_{LS}\|_2} \approx 0.9999 \cdot 10^4 \leq \kappa_2(A)^2 \frac{\|\delta A\|_2}{\|A\|_2} = 10^{12} \cdot 10^{-8}$$

и

$$\frac{\|\hat{r}_{LS} - r_{LS}\|_2}{\|b\|_2} \approx 0.7070 \cdot 10^{-2} \leq \kappa_2(A) \frac{\|\delta A\|_2}{\|A\|_2} = 10^6 \cdot 10^{-8}.$$

На примере 5.3.1 мы наблюдаем, что чувствительность  $x_{LS}$  к возмущениям зависит, видимо, от  $\kappa_2(A)^2$ . В конце этого раздела мы разовьем теорию возмущений для задачи LS и применим ее к оценке процедур решения, основанных на нормальных уравнениях и QR-разложении. К описанию этих процедур мы сейчас переходим.

### 5.3.3. Метод нормальных уравнений

Для решения задачи LS полного ранга наиболее часто используемая процедура – это метод нормальных уравнений.

**Алгоритм 5.3.1 (Нормальные уравнения).** По матрице  $A \in \mathbb{R}^{m \times n}$ , такой, что  $\text{rank}(A) = n$ , и вектору  $b \in \mathbb{R}^m$  алгоритм вычисляет решение  $x_{LS}$  задачи LS  $\min \|Ax - b\|_2$ , где  $b \in \mathbb{R}^m$ .

Вычислить нижний треугольник матрицы  $C = A^T A$ .

$$d = A^T b$$

Вычислить разложение Холецкого  $C = GG^T$ .

Решить систему  $Gy = d$  и  $G^T x_{LS} = y$ .

Алгоритм требует  $(m + n/3)n^2$  флопов. Подход с использованием нормальных уравнений удобен тем, что он опирается на стандартные алгоритмы: разложение Холецкого, перемножение матриц, умножение матрицы на вектор. В случае  $m \gg n$  привлекательно также ужимание  $m \times n$ -матрицы данных  $A$  в существенно меньшую  $n \times n$ -матрицу  $C = A^T A$ .

Займемся вопросом о точности вычисленного методом нормальных уравнений решения  $\hat{x}_{LS}$ . Для ясности предположим, что ошибки округления отсутствуют при формировании  $c = A^T A$  и  $d = A^T b$ . (В типичном случае используемые в этих вычислениях скалярные произведения вычисляются в режиме накопления, так что наше допущение не является резким отклонением от истины.) Из уже известных нам сведений о численных свойствах разложения Холецкого (ср. § 4.2.7) следует, что

$$(A^T A + E) \hat{x}_{LS} = A^T b,$$

где  $\|E\|_2 \approx u \|A^T\|_2 \|A\|_2 \approx u \|A^T A\|_2$ , так что мы можем ожидать

$$\frac{\|\hat{x}_{LS} - x_{LS}\|_2}{\|x_{LS}\|_2} \approx u \kappa_2(A^T A) = u \kappa_2(A)^2. \quad (5.3.2)$$

Другими словами, точность вычисленного методом нормальных уравнений решения зависит от квадрата числа обусловленности. Это, как видно, согласуется с примером 5.3.1. Дальнейшие уточнения см § 5.3.9.

**Пример 5.3.2.** Следует иметь в виду, что формирование  $A^T A$  может повлечь за собой серьезные потери информации. Пусть

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 10^{-3} \\ 10^{-3} \end{bmatrix}.$$

Тогда  $x_2(A) \approx 1.4 \cdot 10^3$ ,  $x_{LS} = (1 \ 1)^T$ ,  $\rho_{LS} = 0$ . Выполнение метода нормальных уравнений в десятичной арифметике с усечением ( $t = 6$ ) вызывает ошибку при делении во время процесса, так как матрица

$$f l(A^T \ A) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

является вырожденной. С другой стороны, в семиразрядной арифметике с усечением  $\hat{x}_{LS} = (2.000001, 0)^T$  и  $\|\hat{x}_{LS} - x_{LS}\|_2 / \|x_{LS}\|_2 \approx u x_2(A)^2$ .

### 5.3.4. Решение задачи LS через QR-разложение

Пусть задана матрица  $A \in \mathbb{R}^{m \times n}$ ,  $m \times n$ , и вектор  $b \in \mathbb{R}^m$ . Предположим, что вычислена такая ортогональная матрица  $Q \in \mathbb{R}^{m \times m}$ , что матрица

$$Q^T A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \quad \begin{matrix} n \\ m-n \end{matrix} \quad (5.3.3)$$

является верхней треугольной. Если

$$Q^T b = \begin{bmatrix} c \\ d \end{bmatrix} \quad \begin{matrix} n \\ m-n \end{matrix},$$

то

$$\|Ax - b\|_2^2 = \|Q^T Ax - Q^T b\|_2^2 = \|R_1 x - c\|_2^2 + \|d\|_2^2$$

для любого  $x \in \mathbb{R}^n$ . Ясно, что если  $\text{rank}(A) = \text{rank}(R_1) = n$ , то  $x_{LS}$  определяется из верхней треугольной системы  $R_1 x_{LS} = c$ . Заметим, что

$$\rho_{LS} = \|d\|_2.$$

Мы делаем вывод, что задача LS полного ранга может быть легко решена, если мы вычислили QR-разложение матрицы  $A$ . Детали зависят от того, какая именно процедура QR-разложения была использована. Если используются матрицы Хаусхолдера и  $Q^T$  применяется к  $b$  в факторизованном виде, то мы получаем

**Алгоритм 5.3.2 (Решение задачи LS преобразованиями Хаусхолдера).** По матрице  $A \in \mathbb{R}^{m \times n}$  полного столбцового ранга и вектору  $b \in \mathbb{R}^m$  следующий алгоритм вычисляет вектор  $x_{LS} \in \mathbb{R}^n$ , такой, что  $\|Ax_{LS} - b\|_2$  достигает минимума.

Применив алгоритм 5.2.1, записать на место  $A$  ее QR-разложение.

**for**  $j = 1:n$

{Применить  $j$ -е преобразование Хаусхолдера.}

$v(j) = 1; \quad v(j+1:m) = A(j+1:m, j)$

$b(j:m) = \text{row.house}(b(j:m), v(j:m))$

**end**

Решить систему  $R(1:n, 1:n)x_{LS} = b(1:n)$  обратной подстановкой.

Этот метод решения задачи LS полного ранга требует  $2n^2(m - n/3)$  флопов. При этом  $O(mn)$  флопов на модификацию  $b$  и  $O(n^2)$  флопов на обратную подстановку несущественны по сравнению с объемом работы, требующимся для факторизации матрицы  $A$ .

Можно показать, что вычисленное  $\hat{x}_{LS}$  является точным решением задачи

$$\min \| (A + \delta A)x - (b + \delta b) \|_2, \quad (5.3.4)$$

где

$$\| \delta A \|_F \leq (6m - 3n + 41) n \mathbf{u} \| A \|_F + O(\mathbf{u}^2) \quad (5.3.5)$$

и

$$\| \delta b \|_2 \leq (6m - 3n + 40) n \mathbf{u} \| b \|_2 + O(\mathbf{u}^2). \quad (5.3.6)$$

Эти неравенства устанавливаются в книге Лоусона и Хенсона (SLS, pp. 90 и далее). Они показывают, что  $\hat{x}_{LS}$  удовлетворяет «близкой» задаче LS. (Мы не можем обратиться к относительной ошибке в  $\hat{x}_{LS}$ , не имея теории возмущений для задачи LS, о которой (теории) мы будем говорить ниже.) Отметим, что аналогичные результаты имеют место при использовании преобразований Гивенса для QR-разложения матрицы.

### 5.3.5. Срыв в случае «почти неполноранговости»

Так же как и метод нормальных уравнений, метод Хаусхолдера решения задачи LS аварийно обрывается на стадии обратной подстановки, если  $\text{rank}(A) < n$ . С численной точки зрения, неприятностей можно ожидать при  $\kappa_2(A) = \kappa_2(R) \approx 1/\mathbf{u}$ . Напомним, что в методе нормальных уравнений завершение факторизации Холецкого проблематично уже при  $\kappa_2(A)$  порядка  $1/\sqrt{\mathbf{u}}$ . (См. пример 5.3.2.) Отсюда вывод у Лоусона и Хенсона (SLS, pp 126–127), что при фиксированной машинной точности ортогонализация Хаусхолдера позволяет решать более широкий класс задач LS.

### 5.3.6. О методе MGS

Обратим внимание на то, что в отличие от ортогонализации Хаусхолдера метод MGS не строит ортонормированный базис для  $\text{range}(A)$ . Тем не менее добываемой в нем информации достаточно для решения задачи LS. Именно, MGS вычисляет разложение  $A = Q_1 R_1$ , где  $Q_1 \in \mathbb{R}^{m \times n}$  имеет ортонормированные столбцы, а  $R_1 \in \mathbb{R}^{n \times n}$  верхняя треугольная. Тем самым система нормальных уравнений  $(A^T A)x = A^T b$  приводится к верхнетреугольному виду  $R_1 x = Q_1^T b$ .

Метод MGS может быть легко приспособлен к решению задачи LS и так же устойчив, как и метод Хаусхолдера. См. Björck (1967b). Однако затраты в MGS чуть выше, поскольку он все время работает с  $m$ -векторами, в то время как в QR-разложении Хаусхолдера длина вектора уменьшается по ходу процесса.

### 5.3.7. Решение задачи LS методом быстрых вращений

Быстрые вращения также могут быть использованы для решения задачи LS. Допустим, что матрица  $M^T M = D$  диагональная, а

$$M^T A = \begin{bmatrix} S_1 \\ 0 \end{bmatrix} \quad \begin{matrix} n \\ m-n \end{matrix}$$

верхняя треугольная. Если

$$M^T b = \begin{bmatrix} c \\ d \end{bmatrix} \quad \begin{matrix} n \\ m-n \end{matrix},$$

то

$$\begin{aligned} \|Ax - b\|_2^2 &= \|D^{-1/2} M^T Ax - D^{-1/2} M^T b\|_2^2 = \\ &= \left\| D^{-1/2} \left( \begin{bmatrix} S_1 \\ 0 \end{bmatrix} x - \begin{bmatrix} c \\ d \end{bmatrix} \right) \right\|_2^2 \end{aligned}$$

для любого  $x \in \mathbb{R}^n$ . Ясно, что  $x_{LS}$  получается решением невырожденной верхней треугольной системы  $S_1 x = c$ .

Можно показать, что вычисленное таким путем решение  $\hat{x}_{LS}$  является точным решением близкой задачи LS в смысле (5.3.4)–(5.3.6). Это может показаться удивительным, поскольку по ходу вычислений могут появиться большие числа. Элемент масштабирующей матрицы  $D$  может удваиваться после каждого шага метода быстрых вращений. Однако увеличение элементов  $D$  должно быть точно скомпенсировано увеличением элементов  $M$ , так как матрица  $D^{-1/2} M$  ортогональна на протяжении всего вычисления. Именно этот феномен позволяет благополучно провести анализ ошибок.

Однако, как обсуждалось в § 5.1.13, возможность роста элементов требует постоянного контроля за матрицей  $D$  для избежания переполнения. Ввиду нетривиальных издержек такого контроля итоговый алгоритм будет, вероятно, медленнее, чем подход Хаусхолдера, и, безусловно, более сложен в реализации.

### 5.3.8. Чувствительность задачи LS к возмущениям

Здесь мы построим теорию возмущений, которая поможет нам провести сравнение метода нормальных уравнений и QR-разложения для решения задачи LS. Нижеследующая теорема изучает действие изменений в  $A$  и  $b$  на решение задачи LS и соответствующую невязку. Тем самым выявляется обусловленность задачи LS.

В ходе анализа нам понадобятся два легко устанавливаемых факта.

$$\begin{aligned} \|A\|_2 \|(A^T A)^{-1} A^T\|_2 &= \kappa_2(A), \\ \|A\|_2^2 \|(A^T A)^{-1}\|_2 &= \kappa_2(A)^2. \end{aligned} \tag{5.3.7}$$

Эти равенства можно проверить, воспользовавшись сингулярным разложением.

**Теорема 5.3.1.** Пусть  $x, r, \hat{x}, \hat{r}$  удовлетворяют

$$\begin{aligned} \|Ax - b\|_2 &= \min, & r &= b - Ax, \\ \|(A + \delta A)\hat{x} - (b + \delta b)\|_2 &= \min, & \hat{r} &= (b + \delta b) - (A + \delta A)\hat{x}, \end{aligned}$$

где  $A$  и  $\delta A$  принадлежат  $\mathbb{R}^{m \times n}$ ,  $m \geq n$ ,  $0 \neq b$  и  $\delta b$  принадлежат  $\mathbb{R}^m$ . Если

$$\varepsilon = \max \left\{ \frac{\|\delta A\|_2}{\|A\|_2}, \frac{\|\delta b\|_2}{\|b\|_2} \right\} < \frac{\sigma_n(A)}{\sigma_1(A)}$$

и

$$\sin(\theta) = \frac{\rho_{LS}}{\|b\|_2} \neq 1,$$

где  $\rho_{LS} = \|Ax_{LS} - b\|_2$ , то

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \varepsilon \left\{ \frac{2\kappa_2(A)}{\cos(\theta)} + \tan(\theta)\kappa_2(A)^2 \right\} + O(\varepsilon^2), \quad (5.3.8)$$

$$\frac{\|\hat{r} - r\|_2}{\|b\|_2} \leq \varepsilon(1 + 2\kappa_2(A)) \min(1, m-n) + O(\varepsilon^2). \quad (5.3.9)$$

*Доказательство.* Определим  $E$  и  $f$  равенствами  $E = \delta A / \varepsilon$ ,  $f = \delta b / \varepsilon$ . По предположению  $\|\delta A\|_2 < \sigma_n(A)$ , так что по теореме 2.5.2 имеем  $\text{rank}(A + tE) = n$  для всех  $t \in [0, \varepsilon]$ . Отсюда следует, что решение  $x(t)$  уравнения

$$(A + tE)^T(A + tE)x(t) = (A + tE)^T(b + tf) \quad (5.3.10)$$

непрерывно дифференцируемо для всех  $t \in [0, \varepsilon]$ . Поскольку  $x = x(0)$ ,  $\dot{x} = x(\varepsilon)$ , имеем

$$\dot{x} = x + \varepsilon \dot{x}(0) + O(\varepsilon^2).$$

Предположения  $b \neq 0$  и  $\sin(\theta) \neq 1$  гарантируют, что  $x$  отличен от нуля, так что

$$\frac{\|\dot{x} - x\|_2}{\|x\|_2} = \varepsilon \frac{\|\dot{x}(0)\|_2}{\|x\|_2} + O(\varepsilon^2). \quad (5.3.11)$$

Чтобы оценить  $\|\dot{x}(0)\|_2$ , продифференцируем (5.3.10) и положим  $t = 0$ . Это дает

$$E^T Ax + A^T Ex + A^T A \dot{x}(0) = A^T f + E^T b,$$

т. е.

$$\dot{x}(0) = (A^T A)^{-1} A^T (f - Ex) + (A^T A)^{-1} E^T b. \quad (5.3.12)$$

Подставив это в (5.3.11), беря соответствующие нормы и используя легко проверяемые неравенства  $\|f\|_2 \leq \|b\|_2$ ,  $\|E\|_2 \leq \|A\|_2$ , получаем

$$\begin{aligned} \frac{\|\dot{x} - x\|_2}{\|x\|_2} &\leq \varepsilon \left\{ \|A\|_2 \|(A^T A)^{-1} A^T\|_2 \left( \frac{\|b\|_2}{\|A\|_2 \|x\|_2} + 1 \right) + \right. \\ &\quad \left. + \frac{\rho_{LS}}{\|A\|_2 \|x\|_2} \|A\|_2^2 \|(A^T A)^{-1}\|_2 \right\} + O(\varepsilon^2). \end{aligned}$$

Так как  $A^T(Ax - b) = 0$ ,  $Ax$  ортогонален к  $Ax - b$ , поэтому

$$\|b - Ax\|_2^2 + \|Ax\|_2^2 = \|b\|_2^2.$$

Таким образом,

$$\|A\|_2^2 \|x\|_2^2 \geq \|b\|_2^2 - \rho_{LS}^2,$$

откуда, используя (5.3.7),

$$\frac{\|\dot{x} - x\|_2}{\|x\|_2} \leq \varepsilon \left\{ \kappa_2(A) \left( \frac{1}{\cos(\theta)} + 1 \right) + \kappa_2(A)^2 \frac{\sin(\theta)}{\cos(\theta)} \right\} + O(\varepsilon^2).$$

Тем самым (5.3.8) доказано.

Чтобы доказать (5.3.9), введем дифференцируемую вектор-функцию  $r(t)$ :

$$r(t) = (b + tf) - (A + tE)x(t)$$

и заметим, что  $r = r(0)$  и  $\dot{r} = r(\varepsilon)$ . Используя (5.3.12), можно показать, что

$$\dot{r}(0) = (I - A(A^T A)^{-1} A^T)(f - Ex) - A(A^T A)^{-1} E^T b.$$

Поскольку  $\|\hat{r} - r\|_2 = \varepsilon \|\dot{r}(0)\|_2 + O(\varepsilon^2)$ , имеем

$$\begin{aligned} \frac{\|\hat{r} - r\|_2}{\|b\|_2} &= \varepsilon \frac{\|\dot{r}(0)\|_2}{\|b\|_2} + O(\varepsilon^2) \leqslant \\ &\leqslant \varepsilon \left\{ \|I - A(A^T A)^{-1} A^T\|_2 \left( 1 + \frac{\|A\|_2 \|x\|_2}{\|b\|_2} \right) + \right. \\ &\quad \left. + \|A(A^T A)^{-1}\|_2 \|A\|_2 \frac{\rho_{LS}}{\|b\|_2} \right\} + O(\varepsilon^2). \end{aligned}$$

Неравенство (5.3.9) следует теперь из того, что

$$\|A\|_2 \|x\|_2 = \|A\|_2 \|A^+ b\|_2 \leqslant \kappa_2(A) \|b\|_2,$$

$$\rho_{LS} = \|(I - A(A^T A)^{-1} A^T)b\|_2 \leqslant \|(I - A(A^T A)^{-1} A^T)\|_2 \|b\|_2$$

и

$$\|(I - (A^T A)^{-1} A^T)\|_2 = \min(m - n, 1).$$

Интересной отличительной чертой оценки (5.3.8) является множитель

$$\operatorname{tg}(\theta) \kappa_2(A)^2 = \frac{\rho_{LS}}{\sqrt{\|b\|_2^2 - \rho_{LS}^2}} \kappa_2(A)^2.$$

Таким образом, в задачах с ненулевой невязкой мерой чувствительности  $\kappa_{LS}$  к возмущениям служит квадрат числа обусловленности. По контрасту с этим чувствительность самой невязки зависит от  $\kappa_2(A)$  только лишь линейно. Все это подтверждается примером 5.3.1.

### 5.3.9. Сравнение метода нормальных уравнений и QR-разложения

Поучительно провести сравнение метода нормальных уравнений и QR-разложения для решения задачи LS. Вспомним ряд важных моментов проведенного обсуждения:

- Чувствительность решения задачи LS к возмущениям примерно пропорциональна величине  $\kappa_2(A) + \rho_{LS} \kappa_2(A)^2$ .
- Метод нормальных уравнений дает  $\hat{x}_{LS}$ , относительная ошибка которого зависит от квадрата числа обусловленности.
- Подход, использующий QR-разложение преобразованиями Хаусхолдера, решает близкую задачу LS и поэтому дает решение, относительная ошибка которого примерно равна  $\mathbf{u}(\kappa_2(A) + \rho_{LS} \kappa_2(A)^2)$ .

Таким образом, мы можем заключить, что если  $\rho_{LS}$  мало, а  $\kappa_2(A)$  велико, метод нормальных уравнений не решает близкую задачу и поэтому полученное им решение будет, как правило, менее точным, чем при хаусхолдеровом подходе. Напротив, оба метода дадут примерно в одинаковой степени неточные результаты при применении к плохо обусловленным задачам с большими невязками.

Упомянем напоследок два других фактора, играющих роль в споре QR против нормальных уравнений:

- Арифметические затраты на решение нормальных уравнений при  $m \gg n$  примерно вдвое меньше; слабее также и требования к объему памяти.

- Хаусхолдерово QR-разложение применимо к более широкому классу матриц, потому что процесс Холецкого для  $A^T A$  срывается еще до начала обратной подстановки для  $Q^T A = R$ .

Уж по крайней мере это обсуждение должно вас убедить, как труден бывает выбор «верного» алгоритма!

### Задачи

- 5.3.1.** Пусть  $A^T A x = A^T b$ ,  $(A^T A + F) \hat{x} = A^T b$ ,  $2 \|F\|_2 \leq \sigma_n(A)^2$ . Покажите, что если  $r = b - Ax$ ,  $\hat{r} = b - A\hat{x}$ , то  $\hat{r} - r = (I - A(A^T A + F)^{-1} A^T)Ax$ , и, как следствие этого,

$$\|\hat{r} - r\|_2 \leq 2\kappa_2(A) \frac{\|F\|_2}{\|A\|_2} \|x\|_2.$$

- 5.3.2.** Пусть  $A \in \mathbb{R}^{m \times n}$ ,  $m > n$ ,  $y \in \mathbb{R}^m$ . Построим матрицу  $\bar{A} = [A \quad y] \in \mathbb{R}^{m \times (n+1)}$ . Покажите, что  $\sigma_1(\bar{A}) \geq \sigma_1(A)$  и  $\sigma_{n+1}(\bar{A}) \leq \sigma_n(A)$ . Тем самым число обусловленности разве что возрастает при добавлении к матрице еще одного столбца.

- 5.3.3.** Пусть  $A \in \mathbb{R}^{m \times n}$ ,  $w \in \mathbb{R}^n$ . Определим

$$B = \begin{bmatrix} A \\ w^T \end{bmatrix}.$$

Покажите, что  $\sigma_n(B) \geq \sigma_n(A)$  и  $\sigma_1(B) \leq \sqrt{\|A\|_2^2 + \|w\|_2^2}$ . Таким образом, при добавлении еще одной строки число обусловленности матрицы может как увеличиться, так и уменьшиться.

- 5.3.4.** Постройте алгоритм решения задачи LS, использующий метод MGS. Дополнительных двумерных массивов не потребуется, если записывать  $r_{ji}$  на место  $a_{ij}$  при  $i \geq j$  и вычислять  $q_i^T b$  по мере построения ортонормированных  $q_i$ .

- 5.3.5** (Cline 1973). Пусть  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $n$ , и гауссово исключение с частичным выбором ведущего элемента используется для вычисления разложения  $PA = LU$ , где  $L \in \mathbb{R}^{m \times n}$  нижняя треугольная матрица с единичной диагональю,  $U \in \mathbb{R}^{n \times n}$  верхняя треугольная матрица,  $P \in \mathbb{R}^{m \times m}$  – матрица перестановок. Объясните, как можно использовать разложение из 5.2.5 для нахождения вектора  $z \in \mathbb{R}^n$ , минимизирующего  $\|Lz - Pb\|_2$ . Покажите, что если  $Ux = z$ , то  $\|Ax - b\|_2$  достигает минимума. Покажите, что этот метод решения задачи LS с точки зрения количества операций лучше, чем хаусхолдерово QR-разложение, если  $m \leq 5/3n$ .

- 5.3.6.** Матрица  $C = (A^T A)^{-1}$ , где  $\text{rank}(A) = n$ , часто возникает во многих статистических приложениях и известна под названием ковариационной матрицы. Допустим, что получено разложение  $A = QR$ . (а) Докажите  $C = (R^T R)^{-1}$ . (б) Дайте алгоритм вычисления диагонали матрицы  $C$ , требующий  $n^3/3$  флопов. (в) Покажите, что

$$R = \begin{bmatrix} \alpha & v^T \\ 0 & S \end{bmatrix} \Rightarrow C = (R^T R)^{-1} = \begin{bmatrix} (1 + v^T C_1 v)/\alpha & -v^T C_1/\alpha \\ -C_1 v/\alpha & C_1 \end{bmatrix},$$

где  $C_1 = (S^T S)^{-1}$ . (д) Используя (в), постройте алгоритм, записывающий верхнюю треугольную часть  $C$  на место верхней треугольной части  $R$ . Ваш алгоритм должен требовать  $2n^3/3$  флопов.

### Замечания и литература к § 5.3

Мы ограничились аппроксимацией наименьшими квадратами, но это не значит, что мы против минимизации в других нормах. Бывают случаи, когда целесообразно минимизировать  $\|Ax - b\|_p$  для  $p = 1$  и  $\infty$ . Некоторые алгоритмы для этого описаны в

Barrodale I. and Roberts F. D. K. (1973). “An Improved Algorithm for Discrete  $L_1$  Linear Approximation”, SIAM J. Num. Anal. 10, 839–848.

Barrodale I. and Phillips C. (1975). “Algorithm 495: Solution of an Overdetermined System of Linear Equations in the Chebychev Norm”, ACM Trans. Math. Soft. 1, 264–270.

Bartels R. H., Conn A. R., and Sinclair J. W. (1978). “Minimization Techniques for Piecewise Diffe-

- rentiable Functions: The  $L_1$  Solution of an Overdetermined Linear System", SIAM J. Num. Anal. 15, 224–241.
- Bartels R. H., Conn A. R. and Charalambous C. (1978). "On Cline's Direct Method for Solving Overdetermined Linear Systems in the  $L_\infty$  Sense", SIAM J. Num. Anal. 15, 255–270.
- Cline A. K. (1976a). "A Descent Method for the Uniform Solution to Overdetermined Systems of Equations", SIAM J. Num. Anal. 13, 293–309.

Две замечательные книги по линейной задаче наименьших квадратов – это Лоусон и Хенсон (SLS) и

- Björck A. (1988). Least Squares Methods: Handbook of Numerical Analysis Vol. 1, Solutions of Equations in  $\mathbb{R}^N$ , Elsevier North Holland.

Гауссовые преобразования для решения задачи LS заслужили некоторое внимание ввиду относительной дешевизны их использования по сравнению с матрицами Хаусхолдера и Гибенса. См.

- Cline A. K. (1973). "An Elimination Method for the Solution of Linear Least Squares Problems", SIAM J. Num. Anal. 10, 283–289.

- Peters G. and Wilkinson J. H. (1970). "The Least Squares Problem and Pseudo-Inverses", Comp. J. 13, 309–316.

- Plemmons R. J. (1974). "Linear Least Squares by Elimination and MGS", J. Assoc. Comput. Mach. 21, 581–585.

Вопросы обусловленности задачи LS обсуждаются в

- Golub G. H. and Wilkinson J. H. (1966). "Note on the Iterative Refinement of Least Squares Solution", Numer. Math. 9, 139–148.

- Saad Y. (1986). "On the Condition Number of Some Gram Matrices Arising from Least Squares Approximation in the Complex Plane", Numer. Math. 48, 337–348.

- Van der Sluis A. (1975). "Stability of the Solutions of Linear Least Squares Problem", Numer. Math. 23, 241–554.

См. также

- Björck A. (1987). "Stability Analysis of the Method of Seminormal Equations", Lin. Alg. and Its Applic. 88/89, 31–48.

«Полунормальные» уравнения – это  $R^T R x = A^T b$ , где  $A = QR$ . В этой статье показано, что решение полунормальных уравнений дает приемлемое решение задачи LS, если выполнить один шаг итеративного уточнения с фиксированной точностью.

Фортранные программы для решения задачи LS преобразованиями Хаусхолдера можно найти в Linpack, гл. 9. Реализация метода MGS решения задачи LS на алголе приведена в

- Bauer F. L. (1965). "Elimination with Weighted Row Combinations for Solving Linear Equations and Least Squares Problems", Numer. Math. 7, 338–52. См. также НАCLA, 119–133.

Задачи наименьших квадратов часто имеют специальную структуру, которую, конечно же, следует использовать. Из многих статей на эту тему отметим две:

- Cox M. G. (1981) "The Least Squares Solution of Overdetermined Linear Equations having Band or Augmented Band Structure", IMA. J. Numer. Anal. 1, 3–22.

- Cybenko G. (1984). "The Numerical Stability of the Lattice Algorithm for Least Squares Linear Prediction Problems", BIT 24, 441–455.

Использовать матрицы Хаусхолдера для решения разреженных задач LS следует с осторожностью, не допуская чрезмерного заполнения. Литература на эту тему включает

- Duff I. S. and Reid J. K. (1976). "A Comparison of Some Methods for the Solution of Sparse Overdetermined Systems of Linear Equations", J. Inst. Math. Applic. 17, 267–280.

- Gill P. E. and Murray W. (1976). "The Orthogonal Factorization of a Large Sparse Matrix", in Sparse Matrix Computations, ed Bunch J. R. and Rose D. J., Academic Press, New York, pp. 177–200.

- Kaufman L. (1979). "Application of Dense Householder Transformations to a Sparse Matrix", ACM Trans. Math. Soft. 5, 442–451.

- Reid J. K. (1967). "A Note on the Least Squares Solution of a Band System of Linear Equations by Householder Reductions", Comp. J. 10, 188–189.

Хотя отражения Хаусхолдера более эффективны для вычисления QR-разложения, в некоторых контекстах выгоднее использовать матрицы Гивенса. Если, к примеру,  $A$  разрежена, то аккуратное применение вращений Гивенса может свести заполнение к минимуму. См.

Duff I. S. (1974). "Pivot Selection and Row Ordering in Givens Reduction on Sparse Matrices", Computing 13, 239–248.

George J. A. and Heath M. T. (1980). "Solution of Sparse Linear Least Squares Problems Using Givens Rotations", Lin. Alg. and Its Applic. 34, 69–83.

## 5.4. Другие ортогональные разложения

Если матрица  $A$  неполного ранга, то QR-разложение необязательно дает базис подпространства  $\text{range}(A)$ . Эта проблема может быть решена, если перед QR-разложением переставить столбцы матрицы  $A$ , т. е.  $A\Pi = QR$ , где  $\Pi$  – матрица перестановок.

Данные в матрице  $A$  могут быть сконцентрированы еще больше, если разрешить умножение справа на подходящую ортогональную матрицу  $Z: Q^T A Z = T$ . Для выбора  $Q$  и  $Z$  имеются интересные возможности, которые, наряду с QR-разложением с выбором ведущего столбца, будут рассмотрены в этом разделе.

### 5.4.1. Случай неполного ранга: QR с выбором ведущего столбца

Если  $A \in \mathbb{R}^{m \times n}$  и  $\text{rank}(A) < n$ , то QR-разложение необязательно строит ортонормированный базис подпространства  $\text{range}(A)$ . Например, если  $A$  состоит из трех столбцов и

$$A = [a_1 \ a_2 \ a_3] = [q_1 \ q_2 \ q_3] \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

–ее QR-разложение, то  $\text{rank}(A) = 2$ , но  $\text{range}(A)$  не совпадает ни с одним из подпространств  $\text{span}\{q_1, q_2\}$ ,  $\text{span}\{q_1, q_3\}$ ,  $\text{span}\{q_2, q_3\}$ .

К счастью, существует простая модификация хаусхолдеровой процедуры QR-разложения (алгоритм 5.2.1), дающая ортонормированный базис подпространства  $\text{range}(A)$ . Модифицированный алгоритм вычисляет разложение

$$Q^T A \Pi = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \begin{matrix} r \\ m-r \end{matrix}, \quad (5.4.1)$$

где  $r = \text{rank}(A)$ ,  $Q$  ортогональна,  $R_{11}$  – верхняя треугольная невырожденная матрица,  $\Pi$  – матрица перестановок. Если ввести разбиения по столбцам  $A\Pi = [a_{c_1}, \dots, a_{c_n}]$  и  $Q = [q_1, \dots, q_m]$ , то для  $k = 1:n$  имеем:

$$a_{c_k} = \sum_{i=1}^{\min\{r,k\}} r_{ik} q_i \in \text{span}\{q_1, \dots, q_r\},$$

откуда  $\text{range}(A) = \text{span}\{q_1, \dots, q_r\}$ .

Матрицы  $Q$  и  $\Pi$  являются соответственно произведениями матриц Хаусхолдера и матриц элементарных перестановок. Допустим, что для некоторого  $k$  мы уже вычислили матрицы Хаусхолдера  $H_1, \dots, H_{k-1}$  и матрицы перестановок  $\Pi_1, \dots,$

$\Pi_{k-1}$ , такие, что

$$(H_{k-1} \dots H_1) A (\Pi_1 \dots \Pi_{k-1}) = \quad (5.4.2)$$

$$= R^{(k-1)} = \begin{bmatrix} R_{11}^{(k-1)} & R_{12}^{(k-1)} \\ 0 & R_{22}^{(k-1)} \\ k-1 & n-k+1 \end{bmatrix} \begin{matrix} k-1 \\ m-k+1 \end{matrix},$$

где  $R_{11}^{(k-1)}$  – невырожденная верхняя треугольная матрица. Пусть теперь  $R_{22}^{(k-1)} = [z_k^{(k-1)}, \dots, z_n^{(k-1)}]$  – разбиение по столбцам, и пусть  $p$  – наименьший индекс, удовлетворяющий  $k \leq p \leq n$ , такой, что

$$\|z_p^{(k-1)}\|_2 = \max \{\|z_k^{(k-1)}\|_2, \dots, \|z_n^{(k-1)}\|_2\}. \quad (5.4.3)$$

Заметим, что если  $k-1 = \text{rank}(A)$ , то этот максимум равен нулю, и процесс закончен. В противном случае пусть  $\Pi_k$  – единичная  $n \times n$ -матрица с переставленными  $p$ -м и  $k$ -м столбцами, и найдем такую хаусхолдерову матрицу  $H_k$ , что из  $R^{(k)} = H_k R^{(k-1)} \Pi_k$  следует  $R^{(k)}(k+1:m, k) = 0$ . Иными словами,  $\Pi_k$  переставляет наибольший столбец из  $R_{22}^{(k-1)}$  в ведущую позицию, а  $H_k$  обнуляет все его поддиагональные компоненты.

Нормы столбцов не нужно будет вычислять заново на каждом шагу, если воспользоваться свойством

$$Q^T z = \begin{bmatrix} \alpha \\ W \end{bmatrix} \begin{matrix} 1 \\ s-1 \end{matrix} \Rightarrow \|w\|_2^2 = \|z\|_2^2 - \alpha^2,$$

выполняющимся для любой ортогональной матрицы  $Q \in \mathbb{R}^{s \times s}$ . Это уменьшает накладные расходы, связанные с выбором ведущего столбца, с  $O(mn^2)$  до  $O(mn)$  флопов, так как мы сможем вычислять новые нормы столбцов через старые, например,  $\|z^{(j)}\|_2^2 = \|z^{(j-1)}\|_2^2 - r_{kj}^2$ .

Собирая вместе полученные результаты, мы приходим к следующему алгоритму (Businger, Golub (1965)):

**Алгоритм 5.4.1 (Хаусхолдерово QR-разложение с выбором ведущего столбца).** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $m \geq n$ , следующий алгоритм вычисляет  $r = \text{rank}(A)$  и разложение (5.4.1), где  $Q = H_1 \dots H_r$ ,  $\Pi = \Pi_1 \dots \Pi_r$ . На место верхнего треугольника матрицы  $A$  записывается верхний треугольник матрицы  $R$ ; компоненты  $j+1:m$   $j$ -го хаусхолдерова вектора хранятся в  $A(j+1:m, j)$ . Матрица перестановок  $\Pi$  закодирована в целочисленном векторе  $piv$  таким образом, что  $\Pi_j$  есть единичная матрица с переставленными строками  $j$  и  $piv(j)$ .

```

for  $j = 1:n$ 
     $c(j) = A(1:m, j)^T A(1:m, j)$ 
end
 $r = 0; \tau = \max \{c(j), \dots, c(n)\}$ 
Найти наименьшее  $k$ ,  $1 \leq k \leq n$ , такое, что  $c(k) = \tau$ 
while  $\tau > 0$ 
     $r = r + 1$ 
     $piv(r) = k; A(1:m, r) \leftrightarrow A(1:m, k); c(r) \leftrightarrow c(k)$ 
     $v(r:m) = \text{house}(A(r:m, r))$ 
     $A(r:m, r:n) = \text{row.house}(A(r:m, r:n), v(r:m))$ 
     $A(r+1:m, r) = v(r+1:m)$ 
    for  $i = r+1:n$ 
         $c(i) = c(i) - A(r, i)^2$ 
    end
```

```

if  $r < n$ 
     $\tau = \max \{ c(r+1), \dots, c(n) \}$ 
Найти наименьшее  $k$ ,  $r+1 \leq k \leq n$ , такое, что  $c(k) = \tau$ .
else
     $\tau = 0$ 
end
end

```

Этот алгоритм требует  $4mn - 2r^2 (m+n) + 4r^3/3$  флопов, где  $r = \text{rank}(A)$ . Как и в процедуре без выбора ведущего столбца, т. е. в алгоритме 5.2.1, ортогональная матрица  $Q$  хранится в факторизованном виде в поддиагональной части  $A$ .

**Пример 5.4.1.** Если алгоритм 5.4.1 применить к

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 5 & 6 \\ 1 & 8 & 9 \\ 1 & 11 & 12 \end{bmatrix},$$

то  $\Pi = [e_3 \ e_2 \ e_1]$ , и с точностью до трех значащих цифр мы получаем

$$A\Pi = Q R = \begin{bmatrix} -0.182 & -0.816 & 0.514 & 0.191 \\ -0.365 & 0.408 & -0.827 & 0.129 \\ 0.548 & 0.000 & 0.113 & -0.829 \\ -0.730 & 0.408 & 0.200 & 0.510 \end{bmatrix} \begin{bmatrix} -16.4 & -14.600 & -1.820 \\ 0.0 & 0.816 & -0.816 \\ 0.0 & 0.000 & 0.000 \end{bmatrix}.$$

## 5.4.2. Полные ортогональные разложения

Вычисленная алгоритмом 5.4.1 матрица  $R$  может подвергнуться дальнейшему приведению, если на нее умножить подходящую последовательность матриц Хаусхолдера. В частности, можно использовать алгоритм 5.2.1 для вычисления

$$Z_r \dots Z_1 = \begin{bmatrix} R_{11}^T \\ R_{12}^T \end{bmatrix} = \begin{bmatrix} T_{11}^T \\ 0 \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix}, \quad (5.4.4)$$

где  $Z_i$  – преобразования Хаусхолдера,  $T_{11}^T$  – верхняя треугольная матрица. Отсюда следует, что

$$Q^T A Z = T = \begin{bmatrix} T_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} r \\ m-r \end{matrix}, \quad (5.4.5)$$

где  $Z = \Pi Z_1 \dots Z_r$ . Всякие разложения такого рода мы будем называть *полными ортогональными разложениями*. Заметим, что  $\text{null}(A) = \text{range}(Z(1:n, r+1:n))$ . См. задачу Р5.2.5 по поводу учета структуры в (5.4.4).

## 5.4.3. Двухдиагонализация

Пусть  $A \in \mathbf{R}^{m \times n}$ ,  $m \geq n$ . Мы покажем, как вычислить такие ортогональные матрицы  $U(m \times m)$  и  $V(n \times n)$ , что

$$U^T A V = \begin{bmatrix} d_1 & f_1 & 0 & \dots & 0 \\ 0 & d_2 & f_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & & d_{n-1} & f_{n-1} \\ 0 & \dots & 0 & & d_n \end{bmatrix}. \quad (5.4.6)$$

Каждая из матриц  $U = U_1 \dots U_n$  и  $V = V_1 \dots V_{n-2}$  может быть найдена как произведение матриц Хаусхолдера:

$$\begin{array}{c} \left[ \begin{array}{ccccc} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{array} \right] \xrightarrow{U_1} \left[ \begin{array}{cccc} x & x & x & x \\ 0 & x & x & x \end{array} \right] \xrightarrow{V_1} \\ \left[ \begin{array}{cccc} x & x & 0 & 0 \\ 0 & x & x & x \end{array} \right] \xrightarrow{U_2} \left[ \begin{array}{cccc} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{array} \right] \xrightarrow{V_2} \\ \left[ \begin{array}{cccc} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{array} \right] \xrightarrow{U_3} \left[ \begin{array}{cccc} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & x \end{array} \right] \\ \xrightarrow{U_4} \left[ \begin{array}{cccc} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 \end{array} \right]. \end{array}$$

В общем,  $U_k$  вносит нули в  $k$ -й столбец, а  $V_k$  обнуляет нужные элементы  $k$ -й строки. В итоге мы имеем

**Алгоритм 5.4.2 (Двухдиагонализация Хаусхолдера).** По матрице  $A \in \mathbb{R}^{m+n}$ ,  $m \geq n$ , следующий алгоритм переписывает две верхние диагонали  $A$  двумя диагоналями матрицы  $U^T A V = B$ , где  $B$ —верхняя двухдиагональная матрица,  $U = U_1 \dots U_n$ ,  $V = V_1 \dots V_{n-2}$ . Значимая часть хаусхолдерова вектора матрицы  $U_j$  хранится в  $A(j+1:m, j)$ , а матрицы  $V_j$  в  $A(j, j+2:n)$ .

```

for j = 1:n
    v(1:m) = house(A(j:m, j))
    A(j:m, j:n) = row.house (A(j:m, j:n), v(j:m))
    A(j+1:m, j) = v(j+1:m)
    if j <= n - 2
        v(j+1:n) = house (A(j, j+1:n)^T)
        A(j:m, j+1:n) = col.house (A(j:m, j+1:n), v(j+1:n))
        A(j, j+2:n) = v(j+2:n)^T
    end
end

```

Этот алгоритм требует  $4mn^2 - 4n^3/3$  флопов. Впервые двухдиагонализация была описана в Golub, Kahan (1965) с использованием именно этой методики. Если желательно получить матрицы  $U_B$  и  $V_B$  в явном виде, то их можно накопить, затратив соответственно  $4m^2n - 4n^3/3$  и  $4n^3/3$  флопов (ср. § 5.1.6). Упомянем еще, что двухдиагонализация матрицы  $A$  имеет отношение к трехдиагонализации матрицы  $A^T A$ . См. § 8.2.1.

**Пример 5.4.2.** Применение алгоритма 5.4.2 к матрице

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

дает с точностью до трех значащих цифр

$$\hat{B} = \begin{bmatrix} 12.8 & 21.8 & 0 \\ 0 & 2.24 & -0.613 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \hat{V}_B = \begin{bmatrix} 1.00 & 0.00 & 0.00 \\ 0.00 & -0.667 & -0.745 \\ 0.00 & -0.745 & 0.667 \end{bmatrix},$$

$$\hat{U}_B = \begin{bmatrix} -0.0776 & -0.833 & 0.392 & -0.383 \\ -0.3110 & -0.451 & -0.238 & 0.802 \\ -0.5430 & -0.069 & 0.701 & -0.457 \\ -0.7760 & 0.312 & 0.547 & 0.037 \end{bmatrix}.$$

#### 5.4.4. R-двуходиагонализация

При  $m \gg n$  более быстрый метод двухдиагонализации получится, если перед применением алгоритма 5.4.2 привести матрицу  $A$  к верхнетреугольному виду. Конкретно, допустим, что мы вычислили такую ортогональную матрицу  $Q \in \mathbb{R}^{m \times m}$ , что матрица

$$Q^T A = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

верхняя треугольная. Мы двухдиагонализуем квадратную матрицу  $R_1$ ,

$$U_R^T R_1 V = B_1.$$

Здесь  $U_R$  и  $V$  – ортогональные  $n \times n$ -матрицы,  $B_1$  – верхняя двухдиагональная  $n \times n$ -матрица. Обозначив  $U = Q \operatorname{diag}(U_R, I_{m-n})$ , получим, что

$$U^T A V = \begin{bmatrix} B_1 \\ 0 \end{bmatrix} = B$$

есть двухдиагонализация матрицы  $A$ .

Идея этого метода двухдиагонализации упомянута у Лоусона и Хенсона (SLS, р. 119) и проанализирована более детально в Chan (1982a). Мы будем именовать этот метод  $R$ -двуходиагонализацией. Сравнивая его затраты флопов ( $2mn^2 + 2n^3$ ) с затратами алгоритма 5.4.2, мы видим, что при  $m \geq 5n/3$  он требует меньшего (приблизительно) количества вычислений.

### 5.4.5. Сингулярное разложение и его вычисление

Следующий за двухдиагонализацией матрицы  $A$  шаг алгоритма Голуба – Райнша вычисления SVD заключается в аннулировании наддиагональных элементов матрицы  $B$ . Это итерационный процесс, осуществляемый алгоритмом, описанным в Golub, Kahan (1965). К сожалению, мы должны отложить обсуждение этой итерации до § 8.3, так как оно требует понимания симметричной проблемы собственных значений. Здесь достаточно сказать, что этот итерационный процесс вычисляет ортогональные матрицы  $U_\Sigma$  и  $V_\Sigma$ , такие, что

$$U_\Sigma^T B V_\Sigma = \Sigma = \text{diag}(\sigma_1, \dots, \sigma_n) \in \mathbb{R}^{m \times n}.$$

Положив  $U = U_B U_\Sigma$  и  $V = V_B V_\Sigma$ , мы видим, что  $U^T A V = \Sigma$  есть сингулярное разложение матрицы  $A$ . Объем вычислений, связанных с этой частью алгоритма, зависит от того, насколько много мы хотим знать о SVD. Например, при решении задачи LS никогда не требуется явно формировать  $U^T$ ; достаточно лишь умножать ее на  $b$  по ходу процесса. Для других приложений требуется только матрица  $U_1 = U(:, 1:n)$ . Всего возможно шесть вариантов; суммарные затраты SVD-алгоритма для каждого случая приведены в таблице ниже. Поскольку возможны две схемы двухдиагонализации, мы приводим подсчет числа операций в двух столбцах. При двухдиагонализации по алгоритму 5.4.2 получается SVD-алгоритм из Golub, Reinsch (1970), а при использовании  $R$ -двухдиагонализации мы получаем алгоритм  $R$ -SVD, детально описанный в Chan (1982a). Сравнивая столбцы нашей таблицы, содержащие приближенные оценки объема работы, мы заключаем, что подход  $R$ -SVD более эффективен, кроме случая  $m \approx n$ .

Требуется	SVD Голуба–Рейнча	$R$ -SVD
$\Sigma$	$4mn^2 - 4n^3/3$	$2mn^2 + 2n^3$
$\Sigma, V$	$4mn^2 + 8n^3$	$2mn^2 + 11n^3$
$\Sigma, U$	$4m^2n - 8mn^2$	$4m^2n + 13n^3$
$\Sigma, U_1$	$14mn^2 - 2n^3$	$6mn^2 + 11n^3$
$\Sigma, U, V$	$4m^2n + 8mn^2 + 9n^3$	$4m^2n + 22n^3$
$\Sigma, U_1, V$	$14mn^2 + 8n^3$	$6mn^2 + 20n^3$

#### Задачи

**5.4.1.** Пусть  $A \in \mathbb{R}^{m \times n}$ , где  $m < n$ . Приведите алгоритм вычисления разложения

$$U^T A V = [B \quad 0],$$

где  $B$  – верхняя двухдиагональная  $m \times n$ -матрица. (Подсказка: используя матрицы Хаусхолдера, приведите  $A$  к виду

$$\left[ \begin{array}{cccccc} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \end{array} \right],$$

а затем вытесните элемент  $(m, m+1)$  вверх по  $(m+1) = u$  столбцу, действуя справа вращениями Гивенса.)

**5.4.2.** Найдите эффективный способ двухдиагонализации верхней треугольной  $n \times n$ -матрицы с использованием вращений Гивенса.

**5.4.3.** Покажите, как привести к верхнему двухдиагональному виду трехдиагональную матрицу  $T \in \mathbb{R}^{n \times n}$ , используя вращения Гивенса.

**5.4.4.** Пусть  $A \in \mathbb{R}^{m \times n}$ , и пусть  $0 \neq v$  удовлетворяет  $\|Av\|_2 = \delta_n(A)\|v\|_2$ . Пусть  $\Pi$  – матрица перестановок, такая, что из  $\Pi^T v = w$  следует  $|w_n| = \|w\|_\infty$ . Покажите, что если  $A\Pi = QR$  – QR-разложение матрицы  $A\Pi$ , то  $|r_{nn}| \leq \sqrt{n}\sigma_n(A)$ . Таким образом, всегда существует такая перестановка  $\Pi$ , что QR-разложение матрицы  $A\Pi$  «проявляет» близость к неполноте ранга.

**5.4.5** Пусть даны  $x, y \in \mathbb{R}^m$  и  $Q \in \mathbb{R}^{m \times m}$ , причем  $Q$  ортогональна. Покажите, что если

$$Q^T x = \begin{bmatrix} \alpha \\ u \end{bmatrix} \quad \frac{1}{m-1}, \quad Q^T y = \begin{bmatrix} \beta \\ v \end{bmatrix} \quad \frac{1}{m-1},$$

то  $u^T v = x^T y - \alpha\beta$ .

**5.4.6.** Пусть даны  $A = [a_1, \dots, a_n] \in \mathbb{R}^{m \times n}$  и  $b \in \mathbb{R}^m$ . Для любого подмножества столбцов  $A \{a_{c_1}, \dots, a_{c_k}\}$  определим

$$\text{res}[a_{c_1}, \dots, a_{c_k}] = \min_{x \in \mathbb{R}^k} \| [a_{c_1}, \dots, a_{c_k}] x - b \|_2.$$

Опишите альтернативную процедуру выбора ведущего столбца для алгоритма 5.4.1, такую, что если в окончательном разложении  $QR = A\Pi = [a_{c_1}, \dots, a_{c_n}]$ , то для  $k = 1:n$

$$\text{res}[a_{c_1}, \dots, a_{c_n}] = \min_{i \geq k} \text{res}[a_{c_1}, \dots, a_{c_{k-1}}, a_{c_i}].$$

#### Замечания и литература к § 5.4

Аспекты полного ортогонального разложения обсуждаются в

Golub G. H. and Pereyra V. (1976). “Differentiation of Pseudo-Inverses, Separable Nonlinear Least Squares Problems and Other Tales”, in Generalized Inverses and Applications, ed. M. Z. Nashed, Academic Press, New York, pp. 303–324.

Hanson R. J. and Lawson C. L. (1969). “Extensions and Applications of the Householder Algorithm for Solving Linear Squares Problems”, Math. Comp. 23, 787–812.

Wedin P. A. (1973). “On the Almost Rank-Deficient Case of the Least Squares Problem”, BIT 13, 344–354.

Вычисление этого разложения детально описано в

Businger P. A. and Golub G. H. (1969). “Algorithm 358: Singular value Decomposition of the Complex Matrix”, Comm. ACM 12, 564–565.

Chan T. F. (1982). “An Improved Algorithm for Computing the Singular Value Decomposition”, ACM Trans. Math. Soft. 8, 72–83.

Golub G. H. and Kahan W. (1965). “Calculating the Singular Values and Pseudo-Inverse of a Matrix”, SIAM J. Num. Anal. 2, 205–224.

Golub G. H. and Reinsch C. (1970). “Singular Value Decomposition and Least Squares Solutions”, Numer. Math. 14, 403–420. См. также HACLA, pp. 134–151.

Любопытный подход к оценке ранга, использующий QR с выбором ведущего столбца и оценку обусловленности, обсуждается в

Chan T. F. (1987). “Rank Revealing QR Factorizations”, Lin. Agl. and Its Applic. 88/89, 67–82.

## 5.5. Задача LS неполного ранга

Если матрица  $A$  неполного ранга, то задача LS имеет бесконечное множество решений, и мы вынуждены прибегать к специальным приемам, направленным на решение трудной задачи численного определения ранга.

После некоторых предварительных замечаний, связанных с SVD, мы показываем, как можно использовать QR-разложение с выбором ведущего столбца для

нахождения решения  $x_B$  с тем свойством, что  $Ax_B$  есть линейная комбинация  $r = \text{rank}(A)$  столбцов. Затем мы обсуждаем решение с минимальной 2-нормой, которое может быть получено через SVD.

### 5.5.1. Решение с минимальной нормой

Пусть  $a \in \mathbb{R}^{m \times n}$  и  $\text{rank}(A) = r < n$ . Задача LS неполного ранга имеет бесконечное множество решений, так как если  $x$  – решение и  $z \in \text{null}(A)$ , то  $x + z$  – также решение. Множество всех решений

$$\chi = \{x \in \mathbb{R}^n : \|Ax - b\|_2 = \min\}$$

выпукло, так как если  $x_1, x_2 \in \chi, \lambda \in [0, 1]$ , то

$$\|A(\lambda x_1 + (1 - \lambda)x_2) - b\|_2 \leq \lambda \|Ax_1 - b\|_2 + (1 - \lambda) \|Ax_2 - b\|_2 = \min \|Ax - b\|_2.$$

Тем самым  $\lambda x_1 + (1 - \lambda)x_2 \in \chi$ . Отсюда следует, что  $\chi$  имеет единственный элемент с минимальной 2-нормой, который мы обозначим  $x_{LS}$ . (Заметим, что в случае полного ранга решение задачи LS единственno, поэтому его 2-норма, естественно, минимальна. Поэтому наши нынешние обозначения согласуются с введенными в § 5.3.)

### 5.5.2. Полное ортогональное разложение и $x_{LS}$

Для вычисления  $x_{LS}$  годится любое полное ортогональное разложение. Действительно, если  $Q$  и  $Z$  – ортогональные матрицы, такие, что

$$Q^T A Z = T = \begin{bmatrix} T_{11} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ r & n-r \end{bmatrix} \begin{matrix} r \\ m-r \end{matrix},$$

то

$$\|Ax - b\|_2^2 = \|((Q^T A Z) Z^T x - Q^T b)\|_2^2 = \|T_{11} w - c\|_2^2 + \|d\|_2^2,$$

где

$$Z^T x = \begin{bmatrix} w \\ y \end{bmatrix} \begin{matrix} r \\ n-r \end{matrix}, \quad Q^T b = \begin{bmatrix} c \\ d \end{bmatrix} \begin{matrix} r \\ m-r \end{matrix}.$$

Если мы хотим, чтобы  $x$  минимизировало сумму квадратов, то ясно, что мы должны иметь  $w = T_{11}^{-1} c$ . Чтобы 2-норма  $x$  была минимальна,  $y$  должен быть нулевым, так что

$$x_{LS} = Z \begin{bmatrix} T_{11}^{-1} c \\ 0 \end{bmatrix}.$$

### 5.5.3. Сингулярное разложение и задача LS

Среди всех полных ортогональных разложений SVD, безусловно, дает особенно много информации. Оно доставляет удобное выражение как для  $x_{LS}$ , так и для нормы минимальной невязки  $\rho_{LS} = \|Ax_{LS} - x\|_2$ .

**Теорема 5.5.1.** Пусть  $U^T A V = \Sigma$  есть сингулярное разложение матрицы  $A \in \mathbb{R}^{m \times n}$ ,  $r = \text{rank}(A)$ . Если  $U = [u_1, \dots, u_m]$  и  $V = [v_1, \dots, v_n]$  – разбиения по столбцам и

$b \in \mathbb{R}^m$ , то

$$x_{LS} = \sum_{i=1}^r \frac{u_i^T b}{\sigma_i} v_i \quad (5.5.1)$$

доставляет минимум функционалу  $\|Ax - b\|_2$  и имеет наименьшую 2-норму среди всех таких  $x$ . Кроме того,

$$\rho_{LS}^2 = \|Ax_{LS} - b\|_2^2 = \sum_{i=r+1}^m (u_i^T b)^2. \quad (5.5.2)$$

*Доказательство.* Для любого  $x \in \mathbb{R}^n$  имеем:

$$\begin{aligned} \|Ax - b\|_2^2 &= \|(U^T A V)(V^T x) - U^T b\|_2^2 = \|\Sigma \alpha\|_2^2 = \\ &= \sum_{i=1}^r (\sigma_i a_i - u_i^T b)^2 + \sum_{i=r+1}^m (u_i^T b)^2, \end{aligned}$$

где  $\alpha = V^T x$ . Ясно, что если  $x$  – решение задачи LS, то  $a_i = (u_i^T b / \sigma_i)$  для  $i = 1:r$ . Если мы положим  $\alpha_{(r+1:n)} = 0$ , то получившееся  $x$  будет, очевидно, иметь минимальную 2-норму.

#### 5.5.4. Псевдообратная матрица

Заметим, что если определить матрицу  $A^+ \in \mathbb{R}^{n \times m}$  соотношением  $A^+ = V \Sigma^+ U^T$ , где

$$\Sigma^+ = \text{diag} \left( \frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0 \right) \in \mathbb{R}^{n \times m},$$

то  $x_{LS} = A^+ b$  и  $\rho_{LS} = \|(I - A A^+) b\|_2$ . Матрица  $A^+$  называется *псевдообратной* к  $A$ . Она является единственным минимальным в  $F$ -норме решением задачи

$$\min_{X \in \mathbb{R}^{n \times m}} \|AX - I_m\|_F. \quad (5.5.3)$$

Если  $\text{rank}(A) = n$ , то  $A^+ = (A^T A)^{-1} A^T$ , а если  $m = n = \text{rank}(A)$ , то  $A^+ = A^{-1}$ . Обычно  $A^+$  определяют как единственную матрицу  $X \in \mathbb{R}^{n \times m}$ , удовлетворяющую четырем условиям *Мура–Пенроуза*:

- |                  |                       |
|------------------|-----------------------|
| (i) $AXA = A$ ,  | (iii) $(AX)^T = AX$ , |
| (ii) $XAX = X$ , | (iv) $(XA)^T = XA$ .  |

Эти условия сводятся к требованию, чтобы  $AA^+$  и  $A^+A$  были ортогональными проекторами на  $\text{range}(A)$  и  $\text{range}(A^T)$  соответственно. Действительно,  $AA^+ = U_1 U_1^T$ , где  $U_1 = U(1:m, 1:r)$ , и  $A^+A = V_1 V_1^T$ , где  $V_1 = V(1:n, 1:r)$ .

#### 5.5.5. Некоторые вопросы чувствительности к возмущениям

В § 5.3 мы исследовали чувствительность к возмущениям задачи LS полного ранга. Поведение  $x_{LS}$  в этой ситуации описано в теореме 5.3.1. Если отбросить предположение о полноте ранга, то  $x$  не является даже непрерывной функцией данных и малые изменения в  $A$  и  $b$  могут вызвать произвольно большие изменения в  $x_{LS} = A^+ b$ . Проще всего это можно увидеть, изучив поведение псевдообратной матрицы. Если  $A$  и  $\delta A$  принадлежат  $\mathbb{R}^{m \times n}$ , то показано (Wedin (1973), Stewart

(1975а)), что

$$\|(A - \delta A)^+ - A^+\|_F \leq 2 \|\delta A\|_F \max \{ \|A^+\|_2^2, \|(A + \delta A)^+\|_2^2 \}$$

Это неравенство обобщает теорему 2.3.4, в которой оцениваются возмущения в обратной матрице. Однако в отличие от случая квадратных невырожденных матриц верхняя граница неизбежно стремится к нулю при стремлении к нулю  $\delta A$ . Если

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

то

$$A^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad (A + \delta A)^+ = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1/\varepsilon & 0 \end{bmatrix}$$

и  $\|A^+ - (A + \delta A)^+\|_2 = 1/\varepsilon$ . Численное нахождение решения задачи LS в присутствии подобных нарушений непрерывности является серьезным испытанием.

### 5.5.6. QR с выбором ведущего столбца и основные решения

Пусть  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $r$ . QR-разложение с выбором ведущего столбца (алгоритм 5.4.1) дает разложение  $A \Pi = Q R$ , где

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \\ r & n-r \end{bmatrix} \quad \begin{matrix} r \\ m-r \end{matrix}.$$

С помощью этого приведения легко получить решение задачи LS. Действительно, для любого  $x \in \mathbb{R}^n$  имеем:

$$\begin{aligned} \|Ax - b\|_2^2 &= \|(Q^T A \Pi)(\Pi^T x) - (Q^T b)\|_2^2 \\ &= \|R_{11}y - (c - R_{12}z)\|_2^2 + \|d\|_2^2, \end{aligned}$$

где

$$\Pi^T x = \begin{bmatrix} y \\ x \end{bmatrix} \quad \begin{matrix} r \\ n-r \end{matrix}, \quad Q^T b = \begin{bmatrix} c \\ d \end{bmatrix} \quad \begin{matrix} r \\ m-r \end{matrix}.$$

Так что если  $x$  доставляет минимум задаче LS, должно быть

$$x = \Pi \begin{bmatrix} R_{11}^{-1} & (c - R_{12}z) \\ z & \end{bmatrix}.$$

Положив в этом выражении  $z$  равным нулю, получим *основное решение*

$$x_B = \Pi \begin{bmatrix} R_{11}^{-1} c \\ 0 \end{bmatrix}.$$

Заметим, что  $x_B$  содержит максимум  $r$  ненулевых компонент, так что в  $Ax_B$  участвует только подмножество столбцов  $A$ .

Основное решение не является минимальным в 2-норме, если только блок  $R_{12}$  не

равен нулю, поскольку

$$\|x_{LS}\|_2 = \min_{z \in \mathbb{R}^{n-r}} \|x_B - \Pi \begin{bmatrix} R_{11}^{-1} & R_{12} \\ -I_{n-r} & \end{bmatrix} z\|_2. \quad (5.5.4)$$

В самом деле, с помощью этой характеристики  $\|x_{LS}\|_2$  можно показать, что

$$1 \leq \frac{\|x_B\|_2}{\|x_{LS}\|_2} \leq \sqrt{1 + \|R_{11}^{-1} R_{12}\|_2^2}. \quad (5.5.5)$$

Детали см. в Golub, Pereyra (1976).

### 5.5.7. Численное нахождение ранга с помощью $A\Pi = QR$

Если для вычисления  $x_B$  используется алгоритм 5.4.1, то необходимо проявлять внимание при определении  $\text{rank}(A)$ . Чтобы почувствовать трудность этой задачи, допустим, что

$$fl(H_k \dots H_1 A \Pi_1 \dots \Pi_k) = \hat{R}^{(k)} = \begin{bmatrix} \hat{R}_{11}^{(k)} & \hat{R}_{12}^{(k)} \\ 0 & \hat{R}_{22}^{(k)} \\ k & n-k \end{bmatrix}$$

есть матрица, вычисленная после  $k$  шагов алгоритма, выполненных в режиме с плавающей точкой. Предположим, что  $\text{rank}(A) = k$ . Ввиду наличия ошибок округления  $\hat{R}_{22}^{(k)}$  не будет в точности нулевой. Однако если  $\hat{R}_{22}^{(k)}$  достаточно мала по норме, разумно завершить приведение и объявить, что ранг  $A$  равен  $k$ . Типичный критерий остановки мог бы выглядеть так:

$$\|\hat{R}_{22}^{(k)}\|_2 \leq \varepsilon_1 \|A\|_2 \quad (5.5.6)$$

для некоторого малого машинно-зависимого параметра  $\varepsilon_1$ . Из свойств матриц Хаусхолдера по отношению к ошибкам округления (ср. § 5.1.12) мы знаем, что  $\hat{R}^{(k)}$  есть точный треугольный множитель для матрицы  $A + E_k$ , где

$$\|E_k\|_2 \leq \varepsilon_2 \|A\|_2, \quad \varepsilon_2 = O(u).$$

Используя теорему 2.5.2, имеем:

$$\sigma_{k+1}(A + E_k) = \sigma_{k+1}(R^{(k)}) \leq \|\hat{R}_{22}^{(k)}\|_2.$$

Так как  $\sigma_{k+1}(A) \leq \sigma_{k+1}(A + E_k) + \|E_k\|_2$ , отсюда следует, что

$$\sigma_{k+1}(A) \leq (\varepsilon_1 + \varepsilon_2) \|A\|_2.$$

Иными словами, относительное возмущение порядка  $O(\varepsilon_1 + \varepsilon_2)$  в матрице  $A$  может дать матрицу ранга  $k$ . С этим критерием остановки мы заключаем, что QR с выбором ведущего столбца обнаруживает неполноту ранга, если по ходу приведения  $\hat{R}_{22}^{(k)}$  оказывается мала для некоторого  $k < n$ .

К сожалению, это происходит не всегда. Матрица  $A$  может быть очень близка к неполноранговой, и ни один из блоков  $\hat{R}_{22}^{(k)}$  не будет маленьким. Таким образом, QR с выбором ведущего столбца сам по себе не является вполне надежным методом обнаружения неполноты ранга. Однако если получить хорошую оценку обусловленности  $R$ , то практически невозможно, чтобы близость к неполноте ранга осталась незамеченной.

**Пример 5.5.1.** Пусть  $T_n(c)$  имеет вид

$$T_n(c) = \text{diag}(1, s, \dots, s^{n-1}) \begin{bmatrix} 1 & -c & -c & \dots & -c \\ 0 & 1 & -c & \dots & -c \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & -c & \\ & & & 1 & 1 \end{bmatrix},$$

где  $c^2 + s^2 = 1$ ,  $c, s > 0$ . (См. Лоусон и Хенсон [SLS, п. 31].) Эти матрицы не изменяются алгоритмом 5.4.1, поэтому  $\|R_{22}^{(k)}\|_2 \geq s^{n-1}$  для  $k = 1:n-1$ . Это неравенство означает, в частности, что матрица  $T_{100}$  (0.2) не имеет особенно малых главных подматриц, так как  $s^{99} \approx 0.13$ . Однако можно показать, что  $\sigma_n = O(10^{-8})$ .

### 5.5.8. Численный ранг и SVD

Мы сосредоточиваем наше внимание на SVD, которое способно управляться с проблемой неполного ранга в условиях влияния ошибок округления. Это относится как к версии Голуба–Райнша, так и к версии Чана. Вспомним, что если  $A = U \Sigma V^T$  есть сингулярное разложение матрицы  $A$ , то

$$x_{LS} = \sum_{i=1}^r \frac{u_i^T b}{\sigma_i} v_i, \quad (5.5.7)$$

где  $r = \text{rank}(A)$ . Обозначим реально вычисленные  $U$ ,  $V$  и  $\Sigma = \text{diag}(\sigma_i)$  через  $\hat{U}$ ,  $\hat{V}$  и  $\hat{\Sigma} = \text{diag}(\hat{\sigma}_i)$ . Допустим, что обе последовательности сингулярных чисел изменяются от наибольшего к наименьшему. Можно показать, что

$$\hat{U} = W + \Delta U; \quad W^T W = I_m; \quad \|\Delta U\|_2 \leq \varepsilon, \quad (5.5.8)$$

$$\hat{V} = Z + \Delta V; \quad Z^T Z = I_n; \quad \|\Delta V\|_2 \leq \varepsilon, \quad (5.5.9)$$

$$\hat{\Sigma} = W^T (A + \Delta A) Z; \quad \|\Delta A\|_2 \leq \varepsilon \|A\|_2, \quad (5.5.10)$$

где  $\varepsilon$  – малое кратное машинной точности  $\mathbf{u}$ . Другими словами, алгоритм SVD вычисляет сингулярные числа «близкой» матрицы  $A + \Delta A$ .

Заметим, что  $\hat{U}$  и  $\hat{V}$  не обязательно близки к соответствующим точным матрицам. Однако мы можем показать, что  $\hat{\sigma}_k$  не сильно отличается от  $\sigma_k$ . Используя (5.5.10) и теорему 2.5.2, имеем

$$\sigma_k = \min_{\text{rank}(B)=k-1} \|A - B\|_2 =$$

$$= \min_{\text{rank}(B)=k-1} \|(\hat{\Sigma} - B) - W^T (\Delta A) Z\|_2.$$

Поскольку  $\|W^T (\Delta A) Z\|_2 \leq \varepsilon \|A\|_2 = \varepsilon \sigma_1$  и

$$\min_{\text{rank}(B)=k-1} \|\hat{\Sigma} - B\|_2 = \hat{\sigma}_k$$

мы получаем, что  $|\sigma_k - \hat{\sigma}_k| \leq \varepsilon \sigma_1$  для  $k = 1:n$ . Тем самым, если ранг матрицы  $A$  равен  $r$ , мы можем ожидать, что  $n-r$  из вычисленных сингулярных чисел будут малы. Близость матрицы  $A$  к неполноранговой не может остаться незамеченной, если вычислить SVD матрицы  $A$ .

**Пример 5.5.2.** Применение алгоритма SVD из пакета Linpack к матрице  $T_{100}$  (0.2) из примера 5.5.1 дает при машинной точности  $10^{-17}$

$$\sigma_n = 0.367805646308792467 \cdot 10^{-8}.$$

Один из возможных путей оценки  $r = \text{rank}(A)$  исходя из вычисленных сингулярных чисел – завести некий допуск  $\delta > 0$  и условиться, что численный ранг матрицы  $A$  равен  $r$ , если сингулярные числа  $\hat{\sigma}_i$  удовлетворяют

$$\hat{\sigma}_1 \geq \dots \geq \hat{\sigma}_r > \delta \geq \hat{\sigma}_{r+1} \geq \dots \geq \hat{\sigma}_n.$$

Допуск  $\delta$  должен быть увязан с машинной точностью, например  $\delta = u \|A\|_\infty$ . Однако если общий уровень относительных ошибок в данных превышает  $u$ ,  $\delta$  должен быть соответственно больше, например  $\delta = 10^{-2} \|A\|_\infty$ , если элементы матрицы верны с точностью до двух значащих цифр.

Если принять  $r$  за численный ранг, то можно рассматривать

$$x_{\hat{r}} = \sum_{i=1}^{\hat{r}} \frac{\hat{u}_i^T b}{\hat{\sigma}_i} \hat{v}_i$$

как приближение к  $x_{LS}$ . Такие как  $\|x_{\hat{r}}\|_2 \approx 1/\hat{\sigma}_{\hat{r}} \leq 1/\delta$ ,  $\delta$  можно выбирать также из соображений получения приближенного решения задачи LS с разумно малой нормой. Более тонкие методы этого будут обсуждаться в § 12.1.

Если  $\hat{\sigma}_{\hat{r}} \gg \delta$ , у нас есть основания быть довольными найденным  $x_{\hat{r}}$ , так как тогда  $A$  может с надежностью рассматриваться как матрица ранга  $r$ .

С другой стороны, множество  $\{\hat{\sigma}_1, \dots, \hat{\sigma}_n\}$  может и не разбиться на четко определенные подмножества больших и малых сингулярных чисел, что делает определение  $r$  описанным методом отчасти произвольным. Это ведет к более сложным методам оценки ранга, которые мы здесь обсудим в контексте задачи LS.

Допустим, например, что  $r = n$ , и предположим на время, что в (5.5.10)  $\Delta A = 0$ . Тогда  $\sigma_i = \hat{\sigma}_i$  для  $i = 1:n$ . Обозначим  $i$ -е столбцы матриц  $\hat{U}$ ,  $W$ ,  $\hat{V}$  и  $Z$  соответственно  $u_i$ ,  $w_i$ ,  $v_i$ ,  $z_i$ . Вычитая  $x_{\hat{r}}$  из  $x_{LS}$  и переходя к нормам, получаем

$$\|x_{\hat{r}} - x_{LS}\|_2 \leq \sum_{i=1}^{\hat{r}} \frac{\|(w_i^T b)z_i - (u_i^T b)v_i\|_2}{\sigma_i} + \sqrt{\sum_{i=\hat{r}+1}^n \left( \frac{w_i^T b}{\sigma_i} \right)^2}.$$

Используя (5.5.8), (5.5.9), легко проверить, что

$$\|(w_i^T b)z_i - (u_i^T b)v_i\|_2 \leq 2(1 + \varepsilon)\varepsilon \|b\|_2 \quad (5.5.11)$$

и поэтому

$$\|x_{\hat{r}} - x_{LS}\|_2 \leq \frac{\hat{r}}{\sigma_{\hat{r}}} 2(1 + \varepsilon)\varepsilon \|b\|_2 + \sqrt{\sum_{i=\hat{r}+1}^n \left( \frac{w_i^T b}{\sigma_i} \right)^2}.$$

Параметр  $\hat{r}$  можно найти как целое число, минимизирующее эту оценку сверху. Обратим внимание, что первый член оценки возрастает с  $\hat{r}$ , а второй убывает.

В тех случаях, когда важнее минимизировать невязку, чем получить точное решение, мы можем определить  $\hat{r}$ , основываясь на предположительной степени близости  $\|b - Ax_{\hat{r}}\|_2$  к истинному минимуму. По аналогии с проведенным выше анализом можно показать, что

$$\|b - Ax_{\hat{r}}\|_2 - \|b - Ax_{LS}\|_2 \leq (n - \hat{r}) \|b\|_2 + \varepsilon \|b\|_2 \left( \hat{r} + \frac{\hat{\sigma}_1}{\hat{\sigma}_{\hat{r}}} (1 + \varepsilon) \right).$$

Снова можно выбрать  $\hat{r}$  так, чтобы оно минимизировало верхнюю оценку.

Описанные приемы могут быть включены в компьютерную программу, где в качестве приближений  $\sigma_i$  и  $w_i$  берутся соответственно  $\hat{\sigma}_i$  и  $\hat{w}_i$ . Проведенный анализ может быть распространен на случай  $\Delta A \neq 0$ . Детали см. в Varah (1973).

### 5.5.9. Некоторые сравнения

Как уже упоминалось, при решении задачи LS посредством SVD достаточно вычислить только  $\Sigma$  и  $V$ . Следующая таблица сравнивает эффективность этого подхода с другими рассмотренными алгоритмами.

LS-Алгоритм	Число флоков
Нормальные уравнения	$mn^2 + n^3/3$
Ортогонализация Хаусхолдера	$2mn^2 - 2n^3/3$
Модифицированный Грамма-Шмидта	$2mn^2$
Ортогонализация Гибенса	$3mn^2 - n^3$
Бидиагонализация Хаусхолдера	$4mn^2 - 4n^3/2$
R-бидиагонализация	$2mn^2 + 2n^3$
SVD Голуба-Рейнча	$4mn^2 + 8n^3$
SVD Шана	$2mn^2 + 11n^3$

### Задачи

5.5.1. Покажите, что если

$$A = \begin{bmatrix} T & S \\ 0 & 0 \\ r & n-r \end{bmatrix} \quad \begin{matrix} r \\ m-r \end{matrix},$$

где  $r = \text{rank}(A)$  и  $T$  невырожденна, то матрица

$$X = \begin{bmatrix} T^{-1} & 0 \\ 0 & 0 \\ r & m-r \end{bmatrix} \quad \begin{matrix} r \\ n-r \end{matrix}$$

удовлетворяет соотношениям  $AXA = A$  и  $(AX)^T = AX$ . В этом случае будем называть  $X$  (1,3)-псевдообратной к матрице  $A$ . Покажите, что для матрицы  $A$  общего вида  $X_B = Xb$ , где  $X$  – (1,3)-псевдообратная к  $A$ .

5.5.2. Определим  $B(\lambda) \in \mathbf{R}^{n \times m}$  как  $B(\lambda) = (A^T A + \lambda I)^{-1} A^T$ , где  $\lambda > 0$ . Покажите, что

$$\|B(\lambda) - A^+\|_2 = \frac{\lambda}{\sigma_r(A)[\sigma_r(A)^2 + \lambda]}, \quad r = \text{rank}(A),$$

так что  $B(\lambda) \rightarrow A^+$  при  $\lambda \rightarrow 0$ .

5.5.3. Рассмотрим задачу LS неполного ранга

$$\min_{\substack{u \in \mathbf{R}^r \\ z \in \mathbf{R}^n}}, \left\| \begin{bmatrix} R & S \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} - \begin{bmatrix} c \\ d \end{bmatrix} \right\|_2,$$

где  $R \in \mathbf{R}^{r \times r}$ ,  $S \in \mathbf{R}^{r \times (n-r)}$ ,  $y \in \mathbf{R}^r$ ,  $z \in \mathbf{R}^{n-r}$ . Предположим, что  $R$  – невырожденная верхняя тре-

угольная матрица. Покажите, как найти для этой задачи решение минимальной нормы, вычисляя подходящее QR-разложение без выбора ведущего столбца и решая систему для подходящих  $u$  и  $z$ .

**5.5.4** Покажите, что если  $A_k \rightarrow A$  и  $A_k^+ \rightarrow A^+$ , то существует такое целое  $k_0$ , что  $\text{rank}(A_k)$  постоянен для всех  $k \geq k_0$ .

**5.5.5.** Покажите, что если  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $n$ , то такой же ранг имеет и матрица  $A + E$ , если выполнено неравенство  $\|E\|_2 \|A^+\|_2 < 1$ .

### Замечания и литература к § 5.5

Фортранную процедуру решения задачи LS посредством SVD или QR-разложения с выбором ведущего столбца можно найти в пакете Linpack.

Литература по псевдообратным матрицам весьма обширна, как свидетельствуют 1775 ссылок, приведенных в

Nashed M. Z. (1976). Generalized Inverses and Applications, Academic Press, New York.

Дальнейшее обсуждение дифференцирования псевдообратной матрицы см. в

Lawson C. L. and Hanson R. J. (1969). “Extensions and Applications of the Householder Algorithm for Solving Linear Least Squares Problems”, Math. Comp. 23, 787–812.

Golub G. H. and Pereyra V. (1973). “The Differentiation of Pseudo-Inverses and Nonlinear Least Squares Problems Whose Variables Separate”, SIAM J. Num. Anal. 10, 413–432.

Обзорное изложение теории возмущений для задачи LS есть у Лоусона и Хенсона (SLS, ch. 7–9), а также в

Wedin P. A. (1973). “Perturbation Theory for Pseudo-Inverses”, BIT 13, 217–232.

Stewart G. W. (1977a). “On the Perturbation of Pseudo-Inverses, Projections, and Linear Least Squares”, SIAM Review 19, 634–662.

QR-разложение с выбором ведущего столбца было впервые рассмотрено в

Businger P. and Golub G. H. (1965). “Linear Least Squares Solutions by Householder Transformations”, Numer. Math. 7, 269–276. См. также HACLA, pp. 111–118.

Даже для задачи полного ранга выбор ведущего столбца, по-видимому, приводит к более точным результатам. Объяснить, почему это так, пытаются анализ ошибок, проведенный в

Jennings L. S. and Osborne M. R. (1974). “A Direct Error Analysis for Least Squares”, Numer. Math. 22, 322–332.

Уметь вовремя остановиться в алгоритме 5.4.1 трудно, но важно. В проблеме неполного ранга бывает полезно получить информацию о наименьшем сингулярном числе верхней треугольной матрицы  $R$ . Это можно сделать, применив приемы из § 3.5.4 или из статей

Anderson N. and Karasalo I. (1975). “On Computing Bounds for the Least Singular Value of a Triangular Matrix”, BIT 15, 1–4.

Karasalo I. (1974). “A Criterion for Truncation of the QR Decomposition Algorithm for the Singular Linear Least Squares Problem”, BIT 14, 156–166.

См. также Лоусон и Хенсон (SLS, гл. 6). Различные другие аспекты неполноты ранга обсуждаются в

Foster L. V. (1986). “Rank and Null Space Calculations Using Matrix Decomposition without Column Interchanges”, Lin. Alg. and Its Applic. 74, 47–71.

Hansen P. C. (1987). “The Truncated SVD as a Method for Regularization”, BIT 27, 534–553.

Stewart G. W. (1984). “Rank Degeneracy”, SIAM J. Sci. and Stat. Comp. 5, 403–413.

Stewart G. W. (1987). “Collinearity and Least Squares Regression”, Statistical Science 2, 68–100.

Varah J. M. (1973). “On the Numerical Solution of Ill-Conditioned Linear Systems with Applications to Ill-Posed Problems”, SIAM J. Num. Anal. 10, 257–267.

Мы еще вернемся к этим вопросам в § 12.1 и § 12.2.

## 5.6. Взвешивание и итерационное уточнение

В § 3.5 в контексте квадратных линейных систем были введены понятия масштабирования и итерационного уточнения. Здесь мы обобщаем эти идеи на случай задачи наименьших квадратов.

### 5.6.1. Взвешивание по столбцам

Пусть  $G \in \mathbf{R}^{n \times n}$  – невырожденная матрица. Решение задачи LS

$$\min \|Ax - b\|_2, \quad A \in \mathbf{R}^{m \times n}, \quad b \in \mathbf{R}^m \quad (5.6.1)$$

может быть получено, если найти минимальное в 2-норме решение  $y_{LS}$  задачи

$$\min \|(AG)y - b\|_2, \quad (5.6.2)$$

а затем положить  $x_G = Gy_{LS}$ . Если  $\text{rank}(A) = n$ , то  $x_G = x_{LS}$ . В противном случае  $x_G$  есть решение задачи (5.6.1), минимальное в  $G$ -норме, которая определяется как  $\|z\|_G = \|G^{-1}z\|_2$ .

Выбор матрицы  $G$  важен. Иногда он может быть сделан на основе априорной информации о неопределенностях в матрице  $A$ . См. Лоусон и Хенсон (SLS, пр. 185–188). В других случаях может оказаться желательным нормировать столбцы  $A$ , положив

$$G = G_0 \equiv \text{diag}(1/\|A(:, 1)\|_2, \dots, 1/\|A(:, n)\|_2).$$

Показано, что (Van der Sluis (1969))

$$\mu = \min_{G \text{ диагональна}} \kappa_2(AG) \Rightarrow \mu \leq \kappa_2(AG_0) \leq \sqrt{n}\mu.$$

Поскольку реально достигнутая в  $v_{LS}$  точность зависит от  $\kappa_2(AG)$ , постолку это довоd в пользу того, чтобы положить  $G = G_0$ .

Заметим, что взвешивание по столбцам воздействует на сингулярные числа. Вследствие этого методы определения численного ранга, будучи примененными к  $A$  и  $AG$ , могут возвращать различные оценки. См. Stewart (1984b).

### 5.6.2. Взвешивание по строкам

Пусть  $D = \text{diag}(d_1, \dots, d_m)$  – невырожденная матрица. Рассмотрим *взвешенную задачу наименьших квадратов*

$$\min \|D(Ax - b)\|_2, \quad A \in \mathbf{R}^{m \times n}, \quad b \in \mathbf{R}^m. \quad (5.6.3)$$

Допустим, что  $\text{rank}(A) = n$  и  $x_D$  есть решение задачи (5.6.3). Тогда решение  $x_{LS}$  задачи (5.6.1) удовлетворяет

$$x_D - x_{LS} = (A^T D^2 A)^{-1} A^T (D^2 - I)(b - Ax_{LS}). \quad (5.6.4)$$

Это показывает, что взвешивание по строкам в задаче LS изменяет ее решение. (Важное исключение – случай  $b \in \text{range}(A)$ , так как тогда  $x_D = x_{LS}$ .)

Один из способов определения  $D$  – положить  $d_k$  равным некоторой мере неопределенности в  $b_k$ , например, обратному к стандартному отклонению  $b_k$ . Значения  $r_k = e_k^T(b - Ax_D)$  имеют тенденцию быть маленькими при больших  $d_k$ . Более точно воздействие  $d_k$  на  $r_k$  можно прояснить следующим образом. Определим

$$D(\delta) = \text{diag}(d_1, \dots, d_{k-1}, d_k \sqrt{1 + \delta}, d_{k+1}, \dots, d_m),$$

где  $\delta > -1$ . Если  $x(\delta)$  минимизирует  $\|D(\delta)(Ax - b)\|_2$  и  $r_k(\delta)$  есть  $k$ -я компонента

$b - Ax(\delta)$ , то можно показать, что

$$r_k(\delta) = \frac{r_k}{1 + \delta d_k^2 e_k^T A (A^T D^2 A)^{-1} A^T e_k}. \quad (5.6.5)$$

Это явное выражение показывает, что  $r_k(\delta)$  – монотонно убывающая функция  $\delta$ . Конечно, при вариации всех весов  $r_k$  изменяется значительно более сложным образом.

Дальнейшее обсуждение взвешивания по строкам можно найти у Лоусона и Хенсона (SLS, pp. 183–185).

**Пример 5.6.1.** Пусть

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Если  $D = I_4$ , то  $x_D = (-1, 0.85)^T$  и  $r = b - Ax_D = (0.3, -0.4, -0.1, 0.2)^T$ . С другой стороны, если  $D = \text{diag}(1000, 1, 1, 1)$ , то мы имеем  $x_D \approx (-1.43, 1.21)^T$  и  $r = b - Ax_D = [0.000428, -0.571428, -0.142853, 0.285714]^T$ .

### 5.6.3. Обобщенные наименьшие квадраты

Во многих задачах на оценивание вектор наблюдений  $b$  связан с  $x$  уравнением

$$b = Ax + w, \quad (5.6.6)$$

где вектор шума  $w$  имеет нулевое среднее значение и симметричную положительно определенную ковариационную матрицу  $\sigma^2 W$ . Предположим, что  $W$  известна и  $W = BB^T$  для некоторого  $B \in \mathbb{R}^{m \times m}$ . Матрица  $B$  может быть задана или она может вычисляться как треугольный множитель Холецкого матрицы  $W$ . Для того чтобы все уравнения (5.6.6) в равной степени влияли на определение  $x$ , статистики часто решают задачу наименьших квадратов

$$\min \|B^{-1}(Ax - b)\|_2. \quad (5.6.7)$$

Очевидный подход к численному решению этой задачи – сформировать  $\tilde{A} = B^{-1}A$  и  $\tilde{b} = B^{-1}b$ , а затем применить любую из описанных выше методик к минимизации  $\|\tilde{A}\tilde{x} - \tilde{b}\|_2$ . К сожалению, результат будет неважным, в случае если матрица  $B$  плохо обусловлена.

Гораздо более надежный метод решения (5.6.7) с использованием ортогональных преобразований был предложен в Paige (1979a, 1979b). Он основан на той идее, что (5.6.7) эквивалентно обобщенной задаче наименьших квадратов

$$\min_{\substack{v \\ b = Ax + Bv}} v^T v. \quad (5.6.8)$$

Обратим внимание, что эта задача определена, даже если обе матрицы  $A$  и  $B$  не полного ранга. Хотя техника Пейджа проходит и в этом случае, мы ее изложим в предположении, что обе эти матрицы имеют полный ранг.

Первый шаг заключается в вычислении QR-разложения матрицы  $A$ :

$$Q^T A = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad Q = \begin{bmatrix} Q_1 & Q_2 \\ n & m-n \end{bmatrix}.$$

Затем определяется ортогональная матрица  $Z \in \mathbb{R}^{m \times n}$ , такая, что

$$Q_2^T BZ = \begin{bmatrix} 0 & S \\ n & m-n \end{bmatrix}, \quad Z = \begin{bmatrix} Z_1 & Z_2 \\ n & m-n \end{bmatrix},$$

где матрица  $S$  верхняя треугольная. С использованием этих ортогональных матриц условие из (5.6.8) трансформируется в

$$\begin{bmatrix} Q_1^T & b \\ Q_2^T & b \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} x + \begin{bmatrix} Q_1^T BZ_1 & Q_1^T BZ_2 \\ 0 & S \end{bmatrix} \begin{bmatrix} Z_1^T & v \\ Z_2^T & v \end{bmatrix}.$$

Заметим, что «нижняя половина» этого уравнения определяет  $v$ ,

$$Su = Q_2^T b, \quad v = Z_2 u, \quad (5.6.9)$$

а «верхняя половина» отвечает за  $x$ :

$$R_1 x = Q_1^T b - (Q_1^T BZ_1 Z_1^T + Q_1^T BZ_2 Z_2^T)v = Q_1^T b - Q_1^T BZ_2 u. \quad (5.6.10)$$

Привлекательность этого метода в том, что вся потенциально плохая обусловленность собрана в треугольных системах (5.6.9) и (5.6.10). Более того, Paige (1979b) показал, что описанная процедура численно устойчива, чего нельзя сказать ни об одном из методов, явно формирующих  $B^{-1}A$ .

#### 5.6.4. Итерационное уточнение

Один из приемов уточнения приближенного решения задачи LS проанализирован в Bjorck (1967, 1968). Он основан на той идее, что если

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m, \quad (5.6.11)$$

то  $\|b - Ax\|_2 = \min$ . Это следует из того, что из  $r + Ax = b$  и  $A^T r = 0$  следует  $A^T Ax = A^T b$ . Выписанная выше расширенная система невырождена, если  $\text{rank}(A) = n$ , что мы и будем в дальнейшем предполагать.

Эта формулировка задачи LS через квадратную линейную систему дает возможность применить схему итерационного уточнения (3.5.5):

$$\begin{aligned} r^{(0)} &= 0; \quad x^{(0)} = 0 \\ \text{for } k &= 0, 1, \dots \\ \begin{bmatrix} f^{(k)} \\ g^{(k)} \end{bmatrix} &= \begin{bmatrix} b \\ 0 \end{bmatrix} - \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r^{(k)} \\ x^{(k)} \end{bmatrix} \\ \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} p^{(k)} \\ z^{(k)} \end{bmatrix} &= \begin{bmatrix} f^{(k)} \\ g^{(k)} \end{bmatrix} \\ \begin{bmatrix} r^{(k+1)} \\ x^{(k+1)} \end{bmatrix} &= \begin{bmatrix} r^{(k)} \\ x^{(k)} \end{bmatrix} + \begin{bmatrix} p^{(k)} \\ z^{(k)} \end{bmatrix} \end{aligned}$$

end

Невязки  $f^{(k)}$  и  $g^{(k)}$  должны вычисляться в режиме с повышенной точностью, и для этой цели надо сохранять исходную матрицу  $A$ .

Если имеется QR-разложение матрицы  $A$ , то найти решение расширенной системы достаточно легко. Именно, если  $A = QR$  и  $R_1 = R(1:n, 1:n)$ , то система вида

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} p \\ z \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

преобразуется к

$$\begin{bmatrix} I_n & 0 & R_1 \\ 0 & I_{m-n} & 0 \\ R_1 & 0 & 0 \end{bmatrix} \begin{bmatrix} h \\ f_2 \\ z \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ g \end{bmatrix},$$

где

$$Q^T f = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix}, \quad Q^T p = \begin{bmatrix} h \\ f_2 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix}.$$

Таким образом,  $p$  и  $z$  могут быть найдены решением треугольных систем  $R_1^T h = g$  и  $R_1 z = f_1 - h$  и вычислением  $p = Q \begin{bmatrix} h \\ f_2 \end{bmatrix}$ . В предположении что  $Q$  хранится в факторизованном виде, каждая итерация требует  $8mn - 2n^2$  флопов.

Ключом к успеху итерационного процесса является уточнение как невязки задачи LS, так и ее решения (а не только решения). Bjorck (1968) показал, что если  $\kappa_2(A) \approx \beta^q$  и используется  $t$ -разрядная арифметика по основанию  $\beta$ , то  $x^{(k)}$  имеет примерно  $k(t-q)$  верных цифр по основанию  $\beta$ , при условии что невязки вычисляются с двойной точностью. Заметим, что в этой оценке фигурирует  $\kappa_2(A)$ , а не  $\kappa_2(A)^2$ .

### Задачи

**5.6.1.** Проверьте (5.6.4).

**5.6.2.** Пусть  $A \in \mathbb{R}^{m \times n}$  имеет полный ранг. Определим диагональную матрицу

$$\Delta = \text{diag} \underbrace{(1, \dots, 1)}_{k-1}, \underbrace{(1 + \delta, 1, \dots, 1)}_{m-k},$$

где  $\delta > -1$ . Обозначим решение задачи LS  $\min \| \Delta (Ax - b) \|_2$  через  $x(\delta)$ , а его невязку через  $r(\delta) = b - Ax(\delta)$ .

(a) Покажите, что

$$r(\delta) = \left( I - \delta \frac{A(A^T A)^{-1} A^T e_k e_k^T}{1 + \delta e_k^T A(A^T A)^{-1} A^T e_k} \right) r(0).$$

(b) Пусть  $r_k(\delta)$  –  $k$ -я компонента  $r(\delta)$ . Покажите, что

$$r_k(\delta) = \frac{r_k(0)}{1 + \delta e_k^T A(A^T A)^{-1} A^T e_k}.$$

(c) Используя (b), докажите (5.6.5).

**5.6.3.** Покажите, как использовать SVD для решения обобщенной задачи LS в случае, когда матрицы  $A$  и  $B$  (5.6.8) неполного ранга.

**5.6.4.** Пусть  $A \in \mathbb{R}^{m \times n}$  имеет ранг  $n$ . Для  $a \geq 0$  определим

$$M(a) = \begin{bmatrix} aI_m & A \\ A^T & 0 \end{bmatrix}.$$

Покажите, что

$$\delta_{m+n}(M(a)) = \min \left\{ a, -\frac{a}{2} + \sqrt{\delta_n(A)^2 + \left(\frac{a}{2}\right)^2} \right\}$$

и найдите значение  $a$ , минимизирующее  $\kappa_2(M(a))$ .

**5.6.5.** Еще один метод итерационного уточнения для задачи LS заключается в следующем:

```

 $x^{(0)} = 0$ 
for  $k = 0, 1, \dots$ 
     $r^{(k)} = b - Ax^{(k)}$  (с двойной точностью)
     $\|Az^{(k)} - r^{(k)}\|_2 = \min$ 
     $x^{(k+1)} = x^{(k)} + z^{(k)}$ 
end

```

(а) Предполагая, что имеется QR-разложение матрицы  $A$ , дайте оценку требуемого количества флотов на итерацию. (б) Покажите, что приведенный выше итерационный процесс получится, если положить  $g^{(k)} = 0$  в схеме итерационного уточнения из § 5.6.4.

### Замечания и литература к § 5.6

Взвешивание по строкам и столбцам для задачи LS обсуждается у Лоусона и Хенсона (SLS, pp. 180–188). Различные эффекты масштабирования обсуждаются в

Van der Sluis A. (1969). “Condition Numbers and Equilibration of Matrices”, Numer. Math. 14, 14–23.

Stewart G. W. (1984b). “On the Asymptotic Behavior of Scaled Singular Value and QR Decompositions”, Math. Comp. 43, 483–490.

Теоретические и вычислительные аспекты обобщенной задачи наименьших квадратов представлены в

Kourouklis S. and Paige C. C. (1981). “A Constrained Least Squares Approach to the General Gauss-Markov Linear Model”, J. Amer. Stat. Assoc. 76, 620–625.

Paige C. C. (1979a). “Computer Solution and Perturbation Analysis of Generalized Least Squares Problems”, Math. Comp. 33, 171–184.

Paige C. C. (1979b). “Fast Numerically Stable Computations for Generalized Linear Least Squares Problems”, SIAM J. Num. Anal. 16, 165–171.

Paige C. C. (1985). “The General Limit Model and the Generalized Singular Value Decomposition”, Lin. Alg. and Its. Applic. 70, 269–284.

Итерационное уточнение для задачи LS обсуждается в

Björck A. (1967a). “Iterative Refinement of Linear Least Squares Solutions I”, BIT 7, 257–278.

Björck A. (1968). “Iterative Refinement of Linear Least Squares Solutions II”, BIT 8, 8–30.

Björck A. (1987). “Stability Analysis of the Method of Seminormal Equations for Linear Least Squares Problems”, Lin. Alg. and Its Applic. 88/89, 31–48.

Björck A. and Golub G. H. (1967). “Iterative Refinement of Linear Least Squares Solutions by Householder Transformation”, BIT 7, 322–337.

Golub G. H. and Wilkinson J. H. (1966). “Note on Iterative Refinement of Least Squares Solutions”, Numer. Math. 9, 139–148.

## 5.7. Квадратные и недоопределенные системы

Развитые в этой главе методы ортогонализации могут быть применены к решению квадратных систем, а также систем, в которых уравнений меньше, чем неизвестных. В этом коротком разделе мы обсудим некоторые из возможностей.

### 5.7.1. Использование QR и SVD для решения квадратных систем

Основанные на QR-разложении либо на SVD методы решения задачи наименьших квадратов могут быть применены для решения квадратных линейных систем: достаточно положить  $m = n$ . Однако с точки зрения количества операций гауссово исключение остается самым дешевым способом решения квадратной линейной системы, как свидетельствует следующая таблица:

Таблица 5.7.1

Метод	Флопы
Гауссово исключение	$2n^3/3$
Ортогонализация	$4n^3/3$
Хаусхолдера	
Модифицированный	
Грамма – Шмидта	$2n^3$
Двухдиагонализация	$8n^3/3$
SVD	$12n^3$

Тем не менее методы ортогонализации заслуживают внимания по целым трем причинам:

- При подсчете количества флопов наблюдается тенденция к преувеличению преимущества гауссова исключения. С учетом издержек векторизации и обменов с памятью QR-подход сравним по эффективности.
- Гарантирана устойчивость методов ортогонализации; в отличие от гауссова исключения, нет нужды беспокоиться о «коэффициенте роста».
- В случаях плохой обусловленности ортогональные методы особенно ясно демонстрируют свою надежность. Очень хорошо ведет себя QR с оценкой обусловленности, и, конечно, непревзойденным методом получения осмысленного решения почти вырожденной системы является SVD.

Мы не заявляем о категорическом предпочтении ортогональных методов, мы только предлагаем конкурентоспособные альтернативы гауссову исключению.

Заметим еще, что число операций для SVD в табл. 5.7.1 предполагает, что вектор  $b$  известен во время разложения. В противном случае требуемое число операций возрастает до  $20n^3$ , так как становится необходимым явное формирование матрицы  $U$ .

## 5.7.2. Недоопределенные системы

Будем называть линейную систему

$$Ax = b, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m, \quad (5.7.1)$$

недоопределенной, если  $m < n$ . Отметим, что такая система либо не имеет решений, либо имеет бесконечное множество решений. Во втором случае важно различать алгоритмы, находящие минимальное в 2-норме решение и не находящие. Первый из представляемых нами алгоритмов относится к этой последней категории.

**Алгоритм 5.7.1.** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $\text{rank}(A) = m$  и вектору  $b \in \mathbb{R}^m$  следующий алгоритм находит  $x \in \mathbb{R}^n$ , такое, что  $Ax = b$ .

$Q^T A \Pi = R$  (QR с выбором ведущего столбца)

Решить систему  $R(1:m, 1:m)z = Q^T b$ .

Положить  $x = \Pi \begin{bmatrix} z \\ 0 \end{bmatrix}$ .

Этот алгоритм требует  $2m^2n - m^3/3$  флопов.

Если мы будем работать с  $A^T$ , мы сможем найти решение минимальной нормы.

**Алгоритм 5.7.1.** По матрице  $A \in \mathbb{R}^{m \times n}$ ,  $\text{rank}(A) = m$ , и вектору  $b \in \mathbb{R}^m$  следующий алгоритм находит минимальное в 2-норме решение системы  $Ax = b$ .

$$A^T = QR \text{ (QR-разложение)}$$

Решить систему  $R(1:m, 1:m)^T z = b$ .

$$x = Q(:, 1:m)z$$

Алгоритм требует максимум  $2m^2n - 2m^3/3$  флопов.

Для вычисления решения минимальной нормы недоопределенной системы можно также использовать SVD. Если

$$A = \sum_{i=1}^r \sigma_i u_i v_i^T r = \text{rank}(A)$$

сингулярное разложение матрицы  $A$ , то

$$x = \sum_{i=1}^r \frac{u_i^T b}{\sigma_i} v_i.$$

Как и в задаче наименьших квадратов, использование SVD желательно во всех случаях, когда  $A$  почти неполного ранга.

### 5.7.3. Возмущенные недоопределенные системы

В заключение мы приводим теорему о возмущениях для недоопределенных систем полного ранга.

**Теорема 5.7.1.** Пусть  $\text{rank}(A) = m \leq n$  и  $A \in \mathbb{R}^{m \times n}$ ,  $\delta A \in \mathbb{R}^{m \times n}$ ,  $0 \neq b \in \mathbb{R}^m$  и  $\delta b \in \mathbb{R}^m$  удовлетворяют

$$\varepsilon = \max \{\varepsilon_A, \varepsilon_b\} < \sigma_m(A),$$

где  $\varepsilon_A = \|\delta A\|_2 / \|A\|_2$  и  $\varepsilon_b = \|\delta b\|_2 / \|b\|_2$ . Если  $x$  и  $\hat{x}$  – решения с минимальной нормой, удовлетворяющие

$$Ax = b, (A + \delta A)\hat{x} = b + \delta b,$$

то

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \kappa_2(A)(\varepsilon_A \min\{2, n-m+1\} + \varepsilon_b) + O(\varepsilon^2).$$

*Доказательство.* Определим  $E$  и  $f$  как  $\delta A/\varepsilon$  и  $\delta b/\varepsilon$ . Заметим, что  $\text{rank}(A + tE) = m$  для всех  $0 < t < \varepsilon$  и что

$$x(t) = (A + tE)^T ((A + tE)(A + tE)^T)^{-1} (b + tf)$$

удовлетворяет  $(A + tE)x(t) = b + tf$ . Дифференцируя это выражение по  $t$  и полагая  $t = 0$ , получаем

$$\dot{x}(0) = (I - A^T (AA^T)^{-1} A) E^T (AA^T)^{-1} b + A^T (AA^T)^{-1} (f - Ex).$$

Так как

$$\begin{aligned} \|x\|_2 &= \|A^T (AA^T)^{-1} b\|_2 \geq \sigma_m(A) \|(AA^T)^{-1} b\|_2, \\ \|I - A^T (AA^T)^{-1} A\|_2 &= \min(1, n-m) \end{aligned}$$

и

$$\frac{\|f\|_2}{\|x\|_2} \leq \frac{\|f\|_2 \|A\|_2}{\|b\|_2},$$

то мы имеем

$$\begin{aligned} \frac{\|\hat{x} - x\|_2}{\|x\|_2} &= \frac{x(\varepsilon) - x(0)}{\|x\|_2} = \varepsilon \frac{\|\dot{x}(0)\|_2}{\|x\|_2} + O(\varepsilon^2) \\ &\leq \varepsilon_{\min}(1, n-m) \left\{ \frac{\|E\|_2}{\|A\|_2} + \frac{\|f\|_2}{\|b\|_2} + \frac{\|E\|_2}{\|A\|_2} \right\} \kappa_2(A) + O(\varepsilon^2), \end{aligned}$$

откуда следует утверждение теоремы.  $\square$

Обратим внимание на отсутствие множителя  $\kappa_2(A)^2$  по сравнению со случаем переопределенных систем.

### Задачи

**5.7.1.** Выведите выписанное выше выражение для  $\dot{x}(0)$ .

**5.7.2.** Найдите решение минимальной нормы для системы  $Ax = b$ , где  $A = [1 \ 2 \ 3]$  и  $b = 1$ .

### Замечания и литература к § 5.7

Некоторые интересные моменты, связанные с вырожденными системами, рассматриваются в

Chan T. F. (1984). "Deflated Decomposition Solutions of Nearly Singular Systems", SIAM J. Numer. Anal. 21, 738–754.

Golub G. H. and Meyer C. D. (1986). "Using the QR Factorization and Group Inversion to Compute, Differentiate, and Estimate the Sensitivity of Stationary Probabilities for Markov Chains", SIAM J. Alg. and Dis. Methods 7, 273–281.

Вот несколько статей по недоопределенным системам:

Arioli M. and Laratta A. (1985). "Error Analysis of an Algorithm for Solving an Underdetermined System", Numer. Math. 46, 255–268.

Cline R. E. and Plemmons R. J. (1976). " $L_2$ -Solutions to Underdetermined Linear Systems", SIAM Review 18, 92–106.

## Глава 6

# Параллельные матричные вычисления

Параллельные вычисления с матрицами – это область интенсивных исследований. Поскольку данное научное направление зародилось совсем недавно, литературы здесь предстотачно и в ней преобладает «изучение отдельных случаев». Трудности машинной зависимости мы обходим, занимаясь проработкой алгоритмов на достаточно высоком уровне. Рассматриваются парадигмы распределенной и общей памяти. Технические детали сведены к минимуму, и мы лишь слегка касаемся нескольких важных языковых и системных проблем, чтобы сосредоточиться на главных вопросах. Как следствие, в этой главе мало конкретных рецептов – здесь вы не найдете четких рекомендаций о том, как решать матричную задачу  $X$  на параллельной машине  $Y$ .

В § 6.1 и § 6.2 ключевые идеи вводятся на примере операции *гахру*. В § 6.3 обсуждается параллельное умножение матриц для систем как с распределенной, так и с общей памятью. Затем в § 6.4, 6.5 и 6.6 рассматривается параллельное вычисление различных матричных разложений. Мы делаем акцент на разложении Холецкого, но всюду даются указания насчет LU- и QR-разложений.

## 6.1. Операция гахру на распределенной памяти

В этом разделе разрабатываются средства записи алгоритмов для распределенной памяти. Чтобы проиллюстрировать их и выявить ключевые положения, рассматривается операция *гахру*:

$$z = y + Ax, \quad A \in \mathbb{R}^{n \times n}, \quad x, y, z \in \mathbb{R}^n. \quad (6.1.1)$$

Более сложные вычислительные процессы будут рассмотрены в § 6.3–6.5.

### 6.1.1. Системы с распределенной памятью

В мультипроцессоре с распределенной памятью имеется сеть процессоров, и в каждом из них есть локальная память и своя программа. Процессоры общаются один с другим, посылая и принимая сообщения. К числу важных факторов при разработке эффективных алгоритмов в случае распределенной памяти относятся: а) число процессоров и размер локальной памяти; б) организация связи между процессорами; в) соотношение скорости вычислений и скорости межпроцессорного взаимодействия. Наша цель – показать совершенно неформально, как эти факторы взаимосвязаны и как они влияют на разработку алгоритма.

В рамках концепции распределенной памяти рассматриваются две модели вычислений: полностью синхронная «систолическая» модель и асинхронная модель

передачи сообщений. В систематической<sup>1)</sup> модели процессоры отмечают порции вычислительной и коммуникационной работы в соответствии с отсчетом времени по глобальным часам. Абсолютно синхронная, пошаговая природа систематических вычислений в какой-то степени облегчает показ потока вычислений. По этой причине они являются превосходной моделью для того, чтобы начать знакомство с параллельными матричными вычислениями.

После того как мы расскажем о систематическом решении задачи (6.1.1), она же будет рассмотрена в контексте модели с обменом сообщениями. Этот стиль работы с распределенной памятью предполагает, что процессоры координируют свою деятельность на основе принимаемых сообщений. Глобальных часов здесь нет.

### 6.1.2. Сети процессоров

В мультипроцессоре с распределенной памятью каждый процессор имеет локальную память и выполняет свою собственную локальную программу, изменяющую значения локальных переменных. В этой главе мы будем постоянно обозначать через  $p$  число процессоров, а о типе связей между ними будем говорить как о сетевой топологии. К числу популярных топологий относятся кольцо, сетка и тор, изображенные на рис. 6.1.1–6.1.3. Каждый из алгоритмов для распределенной памяти в этой главе будет приспособливаться к одной из этих трех топологий. Однако имеется несколько других очень важных коммуникационных схем, в том числе гиперкуб (интересен общностью, оптимальностью и коммерческой доступностью) и дерево (для реализации процедур, построенных по принципу «разделяй и властвуй»). См. Ortega, Voigt (1985) – там обсуждаются различные возможности.

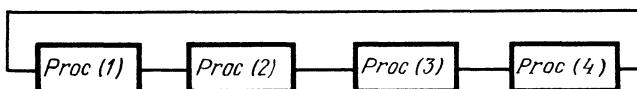


Рис. 6.1.1. 4-процессорное кольцо.

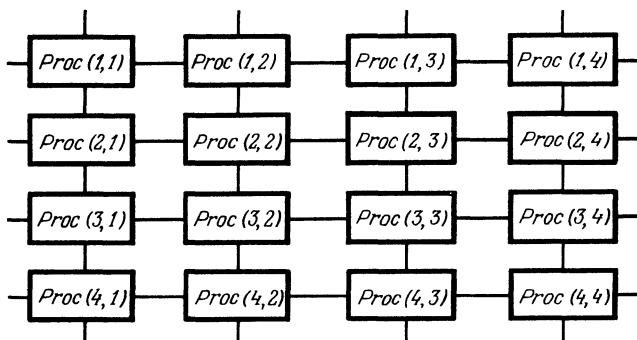
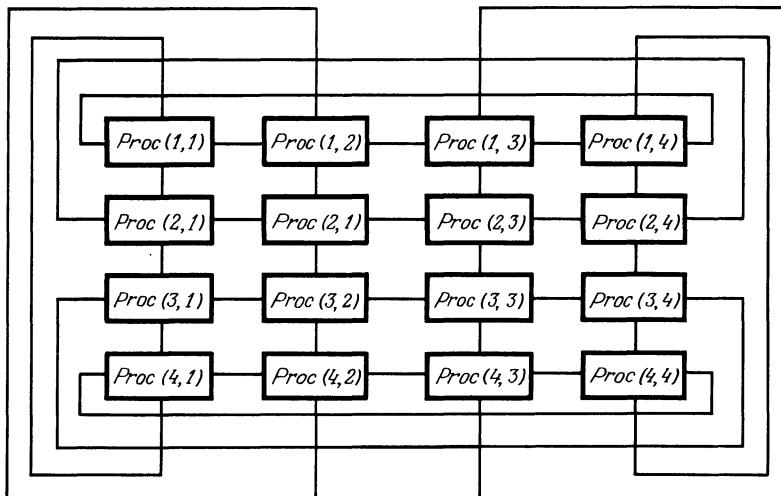


Рис. 6.1.2. Сетка  $4 \times 4$ .

<sup>1)</sup> Мы употребляем слово «систематический», говоря об использующем распределенную память вычислительном процессе, в высокой степени сегментированном и синхронном. Из сформулированных Кунгом [Kung, 1982] свойств вычислений на систематических массивах мы оставляем здесь некоторые, но не все.

Рис. 6.1.3. Тор  $4 \times 4$ .

Заметим, что в торе каждый столбец и каждая строка процессоров образуют кольцо.

### 6.1.3. Имена для соседей

В мультипроцессоре каждый процессор имеет числовую метку  $id$ . Чтобы указать на  $\mu$ -й процессор в кольце, мы используем обозначение  $Proc(\mu)$ . В двумерных сетях используем двойную индексацию:  $Proc(i,j)$ .

Будем говорить, что  $Proc(\lambda)$  является *соседом* для  $Proc(\mu)$ , если между ними существует прямое физическое соединение. Так, в 4-процессорном кольце  $Proc(2)$  и  $Proc(4)$  – соседи для  $Proc(3)$ . Для тора размером  $4 \times 4$  соседями для  $Proc(1, 3)$  являются  $Proc(1, 2)$ ,  $Proc(1, 4)$ ,  $Proc(2, 3)$  и  $Proc(4, 3)$ .

Соседи процессора обычно играют довольно заметную роль в его локальной программе, и поэтому для ссылки на них мы договоримся об удобном нечисловом обозначении. Говоря об индексах левого и правого соседей в кольце, будем писать *left* и *right*. Так, в 4-процессорном кольце  $Proc(4)$  имеет локальные переменные *left* и *right*, значения которых соответственно равны 3 и 1.

В двумерных сетях каждый отдельный процессор  $Proc(i,j)$  имеет локальные переменные *north*, *east*, *south* и *west*, содержащие метки четырех его соседей, т. е.  $(i-1, j)$ ,  $(i, j+1)$ ,  $(i+1, j)$  и  $(i, j-1)$  соответственно. Для сетки может произойти ссылка на несуществующие процессоры, если  $Proc(i,j)$  находится на границе, но мы не будем останавливаться на этой детали. Для определения соседей для тора должна использоваться «круговая» арифметика.

### 6.1.4. Инициализация и окончание

Подобно программам, написанным для единственного процессора, локальные программы начинаются с инициализаций. Например, перед началом собственно вычислений должны быть заданы матрица, ее размеры, множество блочных параметров. Однако в случае мультипроцессора с распределенной памятью также должна быть доступной информация о процессорной сети. В самом деле интересующая нас часть локальной программы может ссылаться на:

- Общее число процессоров  $p$  в сети.
- Метку  $id$ , т. е. индекс или индексы, определяющие исполняющий процессор.
- Подходящие значения для  $left$  и  $right$ , если процессор входит в кольцо, и подходящие значения для  $north$ ,  $east$ ,  $south$  и  $west$ , если процессор является частью тора или сетки.

В интересах нашего изложения мы соберем все эти инициализации в одну команду **loc.init** (*local initialization* – локальная инициализация). Пусть для иллюстрации  $\mu$ -й процессор в  $p$ -процессорном кольце должен разместить подматрицу  $A(1:m, \mu:p:n)$  в локальном массиве  $A_{loc}$ . Тогда локальная программа будет начинаться с предложения вида

**loc.init** [ $p, \mu = my.id, left, right, m, n, A.loc = A(\mu:p:n)$ ].

В такой манере будут начинаться все наши локальные программы. Выражение “ $\mu = my.id$ ” – это просто способ сказать о том, что локальная переменная  $\mu$  содержит числовую метку процессора. Как мы видели в случае сетки и тора,  $id$  может быть двумерной меткой:  $(\alpha, \beta) = my.id$ .

Оператор **loc.init** – это удобное средство сгруппировать в одном месте все отвлекающие аспекты локальной программы, и тем самым оно позволяет нам сосредоточиться на алгоритмических проблемах более высокого уровня.

Все локальные программы заканчиваются оператором **quit**. Это обозначение подчеркивает тот факт, что процессор становится бездействующим после того, как завершено выполнение его собственной локальной программы. Другие процессоры могут оставаться работающими.

### 6.1.5. Связь между процессорами

В условиях распределенной памяти процессоры должны общаться между собой. Нам нужен язык для описания этого действия, и для этой цели мы примем простые отображения «отправить» (**send**) и «принять» (**recv**):

**send**(*{матрица}*, *{куда}*)  
**recv**(*{матрица}*, *{откуда}*).

Таким образом, если  $\text{Proc}(i,j)$  выполняет инструкцию **send**( $A, south$ ), то процессору  $\text{Proc}(i+1,j)$  посыпается копия локально хранимой матрицы  $A$ . Конечно, размер матрицы может быть  $1 \times 1$  или  $n \times 1$ , – в этих случаях посыпается скаляр или вектор. Будем считать, что всякий раз, когда в операторах **send** или **recv** дается имя какой-либо матрицы, ее элементы располагаются в локальной памяти слитно. На практике, возможно, в этом не будет необходимости, но это требование заставляет нас думать в правильном направлении о структуре данных. Итак, запись **send**( $A, (i:), left$ ) незаконна, если  $A$  – локальная матрица, хранимая по столбцам.

Локальные программы, совокупность которых определяет распределенное вычисление, выглядят так же, как обычные последовательные программы, за исключением того, что они включают коммуникационные примитивы типа **send** и **recv**, семантику которых мы сейчас обсуждаем. Если  $\text{Proc}(\mu)$  выполняет инструкцию **send**( $V, \lambda$ ), то процессору  $\text{Proc}(\lambda)$  отсылается копия локальной матрицы  $V$ . Выполнение локальной программы в  $\text{Proc}(\mu)$  возобновляется немедленно. Мы полагаем (как отмечалось выше), что элементы матрицы  $V$  расположены слитно в локальной памяти в  $\text{Proc}(\mu)$ . Мы также полагаем, что процессор имеет право посыпать сообщение самому себе.

Если  $\text{Proc}(\mu)$  выполняет инструкцию **recv**( $V, \lambda$ ), то выполнение локальной программы приостанавливается до тех пор, пока от  $\text{Proc}(\lambda)$  не поступит сообщение.

Как только сообщение получено, оно помещается в локальную матрицу  $V$ , а процессор Proc( $\mu$ ) возобновляет выполнение своей локальной программы. Предполагается, что полученное  $V$  имеет ту же структуру данных, что и в Proc( $\lambda$ ).

При разработке программы для распределенной памяти мы должны позаботиться о том, чтобы гарантировать, что требуемое сообщение будет действительно отправлено. В противном случае локальная программа может «зависнуть», ожидая несуществующего сообщения.

Хотя наши обозначения «отправить» и «принять» совершенно достаточны для записи алгоритмов, они в действительности скрывают некоторые важные детали, связанные с реализацией:

- Длина сообщения в байтах или каких-то других единицах. Обработка более длинных сообщений требует больше времени. Наше обозначение не отражает этой важной стороны вычислений с распределенной памятью.
- Путь сообщения по процессорной сети. Отправленное сообщение может следовать от одного процессора к другому по окружному пути, если системное обеспечение выбора маршрута сочтет это подходящим. Тем самым затрудняется предсказание коммуникационных издержек.
- Буферизация принимаемых сообщений. Поскольку процессоры имеют ограниченную память, локального пространства может быть недостаточно для того, чтобы разместить в нем лавину принимаемых сообщений. Эту возможность мы игнорируем.

### 6.1.6. Некоторые распределенные структуры данных

Предположим, что  $x \in \mathbf{R}^n$  и требуется распределить  $x$  по локальным запоминающим устройствам  $p$ -процессорной сети. Имеются два «канонических» подхода к этой проблеме – строчная и столбцовая схемы хранения. Будем считать на данный момент, что  $n = rp$ . В столбцовой схеме хранения мы рассматриваем  $x$  как  $r \times p$ -матрицу

$$x_{r \times p} = [x(1:r) \ x(r+1:2r) \dots x((p-1)r+1:n)]$$

и храним каждый столбец в одном процессоре, т. е.  $x((\mu-1)r+1:\mu r) \in \text{Proc}(\mu)$ . (Здесь « $\in$ » означает «хранится в».) Заметим, что каждый процессор берет на хранение порцию соседних компонент вектора  $x$ .

В строчной схеме хранения мы рассматриваем  $x$  как  $p \times r$ -матрицу

$$x_{p \times r} = [x(1:p) \ x(p+1:2p) \dots x((r-1)p+1:n)]$$

и храним каждую строку в одном процессоре, т. е.  $x(\mu:p:n) \in \text{Proc}(\mu)$ . О схеме хранения по строкам иногда говорят как о круговом методе распределения вектора  $x$ , потому что его компоненты напоминают карты в колоде, которые «сдаются» процессорам по кругу.

Если  $n$  не является точным кратным для  $p$ , то эти идеи работают после незначительной модификации. Рассмотрим столбцовую схему хранения при  $n = 14$  и  $p = 4$ :

$$x^T = [x_1 x_2 x_3 x_4 | x_5 x_6 x_7 x_8 | x_9 x_{10} x_{11} | x_{12} x_{13} x_{14}].$$

Proc(1)   Proc(2)   Proc(3)   Proc(4)

В общем случае, если  $n = pr + q$  с  $0 \leq q < p$ , то процессоры Proc(1), ..., Proc( $q$ ) могут взять себе по  $r + 1$  компонент каждый, а процессоры Proc( $q + 1$ ), ..., Proc( $p$ ) могут взять на хранение по  $r$  компонент. В строчной схеме хранения мы просто полагаем, что в Proc( $\mu$ ) размещается  $x(\mu:p:n)$ .

Далее рассмотрим распределение  $n \times n$ -матрицы  $A$  на кольцевой сети процессоров. Снова предположим для ясности, что  $n = rp$ . Есть четыре очевидных возможности:

Способ задания матрицы	Схемы хранения	Что помещается в Proc ( $\mu$ )
По столбцам	Подряд	$A(:, (\mu - 1)r + 1 : \mu)$
По столбцам	По кругу	$A(:, \mu : p : n)$
По строкам	Подряд	$A((\mu - 1)r + 1 : \mu r, :)$
По строкам	По кругу	$A(\mu : p : n, :)$

Распределение матрицы на сетке или торе может быть осуществлено несколькими способами. В случае  $p_1 \times p_2$ -сети простейшую распределенную структуру данных получаем, рассматривая  $A = A(i, j)$  как блочную  $p_1 \times p_2$ -матрицу и помещая  $A_{ij}$  в Proc( $i, j$ ). Возможны также двумерные круговые отображения.

### 6.1.7. Систолическая модель

Теперь мы рассмотрим систолические вычисления. В систолической сети процессоры работают абсолютно синхронно. В течение одного «отсчета времени» по глобальным часам каждый процессор обменивается информацией со своими соседями и затем выполняет некоторое локальное вычисление.

Предполагается, что в этой простой высокоструктурированной модели вычислений с распределенной памятью каждый процессор выполняет локальную программу следующего вида:

```

for t = 1 : tfinal
    Послать данные соседям (если необходимо).
    Получить данные от соседей (если необходимо).          (6.1.2)
    Выполнить вычисления над локальными данными (если необходимо).
end
quit

```

Здесь  $t$  играет роль глобального счетчика, задающего темп вычислений в этой сети. Сначала все процессоры выполняют свои предписания для  $t = 1$ . Затем все процессоры выполняют свои предписания для  $t = 2$ , и т. д. Все организовано таким образом, что сначала посылаются все сообщения, затем все они принимаются, и после этого выполняются все локальные вычисления. Такого рода систематизация упрощает строгое доказательство корректности распределенных вычислений, но в действительности она может затруднить разработку алгоритма. Это будет показано в § 6.5.

Чтобы проиллюстрировать наше обозначение, рассмотрим операцию гахру (6.1.1) на  $p$ -процессорном систолическом кольце. Для ясности будем считать, что  $n = rp$ , и разобьем задачу на блоки:

$$\begin{bmatrix} z_1 \\ \vdots \\ z_p \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} + \begin{bmatrix} A_{11} & \dots & A_{1p} \\ \vdots & & \vdots \\ A_{p1} & \dots & A_{pp} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}. \quad (6.1.3)$$

Здесь  $A_{ij} \in \mathbf{R}^{r \times r}$  и  $x_i, y_i, z_i \in \mathbf{R}^r$ . Предположим, что в начале вычислений Proc( $\mu$ ) берет на хранение  $\mu$ -ю блочную строку из  $A$ ,  $x_\mu$  и  $y_\mu$ , а возложенная на него задача состоит в получении  $z_\mu$  на месте  $y_\mu$ . Заметим, что в вычислении

$$z_\mu = y_\mu + \sum_{\tau=1}^p A_{\mu\tau} x_\tau$$

участвуют локальные данные ( $A_{\mu\tau}, y_\mu, x_\mu$ ) и нелокальные данные (другие  $x_\tau$ ). Чтобы сделать  $x$  доступным для всего кольца процессоров, нужно заставить  $x_\tau$  перемещаться по кругу наподобие карусели. Для иллюстрации возьмем  $p = 3$  и покажем, «где» находится  $x_\tau$  на каждом шаге этой процедуры:

Шаг	Proc (1)	Proc (2)	Proc (3)
1	$x_3$	$x_1$	$x_2$
2	$x_2$	$x_3$	$x_1$
3	$x_1$	$x_2$	$x_3$

Для кольца этот метод, в котором  $x_\tau$  ходит по кругу, особенно привлекателен, так как обмен сообщениями всегда происходит между соседями. Ниже – точная формулировка процедуры.

**Алгоритм 6.1.1.** Предположим, что заданы  $A \in \mathbf{R}^{n \times n}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{R}^n$ , и пусть  $z = y + Ax$ . Если каждый процессор в  $p$ -процессорном систолическом кольце исполняет следующую локальную программу и  $n = rp$ , то после окончания работы Proc( $\mu$ ) содержит  $z(1 + (\mu - 1)r : \mu r)$  в  $y_{loc}$ .

```

loc. init [ $p, \mu = my.id, left, right, n, r = n/p, row = 1 + (\mu - 1)r : \mu r,$ 
 $A_{loc} = A(row, :), x_{loc} = x(row), y_{loc} = y(row)]$ 
for  $t = 1 : p$ 
    send ( $x_{loc}, right$ ); recv ( $x_{loc}, left$ )
     $\tau = \mu - t$ 
    if  $\tau \leq 0$ 
         $\tau = \tau + p$ 
    end
     $\{x_{loc} = x(1 + (\tau - 1)r : \tau r)\}$ 
     $y_{loc} = y_{loc} + A_{loc}(:, 1 + (\tau - 1)r : \tau r) x_{loc}$ 
end
quit

```

Целочисленная арифметика в этой процедуре типична для матричных вычислений на распределенной памяти. Индекс  $\tau$  помечает текущую доступную часть вектора  $x$ . Если  $\tau$  известно, то можно вычислить поправку к локально хранимой порции вектора  $y$ . Заметим, что  $A_{loc}(i, j)$  не содержит  $A(i, j)$ .

### 6.1.8. Атрибуты параллельного алгоритма

Каким образом мы можем добиваться нужной нам производительности для параллельной процедуры типа алгоритма 6.1.1? Сейчас мы находимся в положении, сравнимом с тем, когда мы обсуждали векторные вычисления в § 1.4. В тот раз мы ограничились довольно неформальным обсуждением – «о чём нужно думать» при разработке процедуры для векторного компьютера.

Здесь наша философия будет сходной. Мы не собираемся строить детализированную модель параллельных вычислений, которая позволяла бы предсказывать время выполнения алгоритма. Вместо этого мы отметим несколько ключевых

положений, которые должны учитываться при конструировании параллельных матричных вычислений. Наша задача не так уж непохожа на то, чем занимается администратор, руководящий работой приемной канцелярии университета. Распределена ли работа равномерно среди сотрудников? Есть ли узкие места, ведущие к бездеятельности? Не тратится ли больше времени на перекладывание бумаг со стола на стол, вместо того чтобы прочесть их? Насколько эффективно работает канцелярия? Что дало бы увеличение штата?

### 6.1.9. Равномерная загруженность

Мы начнем со свойства равномерной загруженности. Будем говорить, что параллельное вычисление дает равномерную загруженность, если каждому процессору приходится выполнять примерно одинаковое количество полезной вычислительной и коммуникационной работы на каждом временном шаге. Алгоритм 6.1.1 обеспечивает идеальную равномерную загруженность в том смысле, что на каждом временном шаге каждый процессор имеет в точности один и тот же объем вычислений и обменов.

На примере алгоритма 6.1.1 можно проиллюстрировать и неравномерную загруженность; для этого предположим, что матрица  $A$  – блочная нижняя треугольная с  $p \times p$ -блоками и  $r = p$ . Тогда разумно перевычислять  $y_{loc}$ , лишь когда блок  $A_{loc}(:, 1 + (\tau - 1)p : \tau p)$  не равен нулю:

```
if  $\tau \leq \mu$ 
     $y_{loc} = y_{loc} + A_{loc}(:, 1 + (\tau - 1)p : \tau p) x_{loc}$ 
end
```

При таком использовании структуры матрицы в итоге получается процедура, не дающая равномерной загруженности, потому что (например) при  $t = p - 1$  только Proc( $p$ ) имеет что перевычислять. К выводу о неравномерной загруженности можно прийти и с другой стороны, – заметив, что для получения  $z_\mu$  требуется  $2\mu p^2$  флопов, и, таким образом, Proc( $p$ ) должен выполнить в  $p$  раз больше флопов, чем Proc(1).

Однако чтобы получить процедуру с равномерной загруженностью, мы просто прибегнем к круговому отображению данных.

**Алгоритм 6.1.2.** Предположим, что заданы  $A \in \mathbf{R}^{n \times n}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{R}^n$  и что  $z = y + Ax$ . Пусть  $n = rp$  и  $A$  – блочная нижняя треугольная матрица с  $r \times r$ -блоками. Если каждый процессор в  $p$ -процессорном систолическом кольце выполняет следующую локальную программу, то в конце процесса в  $y_{loc}$  для Proc( $\mu$ ) будет содержаться  $z(\mu : p : n)$ .

```
loc.init [ $p, \mu = my.id, left, right, n, r = n/p$ ,
 $A_{loc} = A(\mu:p:n, :), x_{loc} = x(\mu:p:n), y_{loc} = y(\mu:p:n)$ ]
for  $t = 1:p$ 
    send ( $x_{loc}, right$ ); recv ( $x_{loc}, left$ )
     $\tau = \mu - t$ 
    if  $\tau \leq 0$ 
         $\tau = \tau + p$ 
    end
     $\{x_{loc} = x(\tau:p:n)\}$ 
    for  $j = 1:r$ 
         $y_{loc}(j:r) = y_{loc}(j:r) + A_{loc}(j:r, \tau + (j - 1)p) x_{loc}(j)$ 
    end
end
quit
```

Каждый процессор должен выполнить на каждом шаге некоторое множество операций одинаковых размеров, и, таким образом, эта процедура дает равномерную загруженность.

В алгоритме 6.1.2 труднее всего понять индексирование. Мы работаем мысленно с  $A(i,j)$ ,  $x(j)$  и  $y(i)$ , но они не соответствуют  $A_{loc}(i,j)$ ,  $x_{loc}(j)$  и  $y_{loc}(i)$ . Таким образом, с этим алгоритмом связан большой труд по «обдумыванию индексов», и это, к несчастью, типично для матричных вычислений на распределенной памяти – до тех пор, пока у нас нет хорошего компилятора.

### 6.1.10. Стоимость обмена информацией

Накладные расходы на обмен информацией можно оценить количественно, если мы сформируем модель оценки стоимости передачи сообщения. Для того чтобы это сделать, будем полагать, что передача  $m$  чисел с плавающей точкой от одного процессора к другому требует

$$\text{comm}(m) = \alpha_d + \beta_d m \quad (6.1.4)$$

секунд. Здесь  $\alpha_d$  – время, необходимое для инициализации сообщения, и  $\beta_d$  – скорость передачи сообщения. В алгоритме 6.1.1 на каждом временном шаге посыпается и принимается  $r$ -вектор и выполняется  $2r^2$  флопов. Если вычисления производятся со скоростью  $R$  флопов в секунду и нет никаких задержек, связанных с `recv`, то каждый временной шаг требует  $(2r^2/R) + 2(\alpha_d + \beta_d r)$  секунд.

Более информативной величиной является отношение вычислительных затрат к коммуникационным. Для алгоритма 6.1.1 оно имеет вид

$$\frac{\text{Время, занимаемое вычислениями}}{\text{Время, занимаемое обменами}} \approx \frac{2r^2/R}{2(\alpha_d + \beta_d r)}.$$

Эта дробь дает количественное выражение коммуникационным издержкам по отношению к объему вычислений. Очевидно, если  $r = n/p$  возрастает, то увеличивается и часть времени, занимаемого вычислениями.

### 6.1.11. Эффективность и ускорение

Эффективность  $p$ -процессорного параллельного алгоритма определяется

$$E = \frac{T(1)}{p T(p)},$$

где  $T(k)$  – время, требуемое для выполнения программ на  $k$  процессорах. Если вычисления происходят со скоростью  $R$  флопов в секунду, а коммуникации моделируются формулой (6.1.4), то в алгоритме 6.1.1 разумная оценка для  $T(k)$  дается формулой

$$T(k) = \sum_{i=1}^k 2(n/k)^2/R + 2(\alpha_d + \beta_d(n/k)) = \frac{2n^2}{Rk} + 2\alpha_d k + 2\beta_d n.$$

Предполагается, что процессоры не простаивают. Тогда мы находим

$$E = \frac{1 + \frac{\alpha_d R}{n^2} + \frac{\beta_d R}{n}}{1 + \frac{\alpha_d R p^2}{n^2} + \frac{\beta_d R_p}{n}}.$$

Заметим, что эффективность улучшается с ростом  $n$  при фиксированном  $p$  и ухудшается с ростом  $p$  при фиксированном  $n$ . Полезное наблюдение такого рода – вот практически и все, что удается извлечь из нашей грубой модели вычислений и обменов. На практике единственным заслуживающим доверия способом приобретения интуиции насчет матричных вычислений на какой-либо конкретной системе с распределенной памятью являются тестовые замеры производительности.

С эффективностью связано понятие *ускорения*. Мы говорим, что параллельный алгоритм для конкретной задачи достигает ускорения  $S$ , если

$$S = T_{seq}/T_{par},$$

где  $T_{par}$  – время выполнения параллельного алгоритма и  $T_{seq}$  – время реализации на одном процессоре – при условии, что используется наилучшая однопроцессорная процедура. Для многих задач самый быстрый последовательный алгоритм не распараллеливается и, следовательно, для оценки ускорения привлекаются два различных алгоритма.

### 6.1.12. Дробление вычислений

Еще одной важной характеристикой параллельного алгоритма является принятая в нем дробление вычислений. Это – качественная характеристика, относящаяся к объему вычислительной работы, выполняемой между точками синхронизации. В систолическом алгоритме процессоры синхронизируются на каждом временном шаге, и, значит, в этой модели показателем дробления вычислений служит просто их количество на каждом шаге. Например, каждый временной шаг в алгоритме 6.1.1 содержит  $r \times r$ -операцию гахру. Более мелкое дробление получается, когда объем сообщений ограничен одним числом с плавающей точкой. Вот процедура, возникающая при этом

```

loc. init [p, μ = my.id, left, right, n, r = n/p, row = 1 + (μ - 1)r : μr,
           Aloc = A(row, :), xloc = x(row), yloc = y(row)]
for t = 1 : n
    send(xloc(ceil(t/p), right)); recv(xloc(ceil(t/p), left))
    ρ = ceil(t/p); t1 = t - (ρ - 1)p
    τ = μ - t1
    if τ ≤ 0
        τ = τ + p
    end
    yloc = yloc + Aloc(:, (τ - 1)r + ρ) xloc(ρ)
end
quit

```

(6.1.5)

Вместо одного оборота карусели, как в алгоритме 6.1.1, здесь делается  $r$  оборотов. При  $ρ$ -м обороте  $x(\rho:p:n)$  обходит круг. В момент перевычисления  $y_{loc}$  в  $x_{loc}(\rho)$  помещается  $x(\rho + (\tau - 1)r)$ .

Заметим, что (6.1.5) имеет временных шагов в  $r = n/p$  раз больше, чем алгоритм 6.1.1. Однако на каждом шаге выполняется меньшая работа, так как вместо гахру выполняется схару. Итак, дробление вычислений в (6.1.5) более мелкое, чем в алгоритме 6.1.1.

Важно понимать, что вместе с дроблением вычислений изменяются и коммуникационные издержки. В алгоритме 6.1.1 процессор тратит на обмены  $2(p\alpha_d + \beta_d n)$  секунд, а в (6.1.5) –  $2(n\alpha_d + \beta_d n)$  секунд. Таким образом, для системы, поддерживающей мелко-дробный параллелизм, важнейшее значение имеет малость стартового времени  $\alpha_d$  сообщения.

Параллелизм мелкого дробления часто облегчает получение равномерной загруженности. Однако если в системе с высокопроизводительными процессорами вычисления дробятся слишком мелко, то локальные программы не могут работать со скоростью 2-го или 3-го уровня просто из-за того, что именно им и недостаточно локальных линейно-алгебраических операций.

На дробление вычислений может влиять также выбор структуры распределенных данных. Ниже – реорганизация алгоритма 6.1.1 в предположении, что Proc( $\mu$ ) содержит  $\mu$ -й блочный столбец (вместо  $\mu$ -й блочной строки) матрицы  $A$  из (6.1.1).

**Алгоритм 6.1.3.** Предположим, что заданы  $A \in \mathbb{R}^{n \times n}$ ,  $x \in \mathbb{R}^n$  и что  $z = y + Ax$ . Если в  $p$ -процессорном кольце каждый процессор исполняет следующую локальную программу и  $n = rp$ , то после окончания процесса Proc( $\mu$ ) содержит  $z(1 + (\mu - 1)r : \mu r)$ .

```

loc. init [ $p, \mu = my.id, left, right, n, r = n/p, col = 1 + (\mu - 1)r : \mu r,$ 
 $A_{loc} = A(:, col), x_{loc} = x(col), y_{loc} = y(col)]$ 
for  $t = 0 : p$ 
    if  $t = 0$ 
         $w = A_{loc} x_{loc}$ 
    else
        send ( $w, right$ ); recv ( $w, left$ )
         $\tau = \mu - t$ 
        if  $\tau \leq 0$ 
             $\tau = \tau + p$ 
        end
         $\{w = A_\tau x_\tau\}$ 
         $y_{loc} = y_{loc} + w(1 + (\tau - 1)r : \tau r)$ 
    end
end
quit
```

Заметим, что в этой процедуре циркулируют векторы  $w$ , а не векторы  $x_{loc}$ , как в алгоритме 6.1.1. Поскольку векторы  $w$  имеют длину  $n$ , а  $x_{loc}$  – длину  $r = n/p$ , мы видим, что алгоритму 6.1.3 сопутствуют большие коммуникационные издержки.

По отношению к дроблению вычислений вместо  $2r^3$  флопов между точками коммуникации, как в алгоритме 6.1.1, теперь мы имеем только  $r$  флопов. Изменение в степени дробления вызвано массивным вычислением, имеющим место на шаге  $t = 0$ .

### 6.1.13. Модель передачи сообщений

Теперь обсудим вычисления на распределенной памяти в отсутствие глобальных часов. Координация процессоров здесь достигается с помощью посылки и приема сообщения. В отличие от систолической модели процессор может отправить сообщение в любое время и не обязан ожидать следующего отсчета глобальных часов.

Наш язык передачи сообщений «не заботится» о том, как связаны процессоры. Любой процессор может отправить сообщение любому процессору. Однако на практике распределенные вычисления стоит организовывать таким образом, чтобы сообщения посыпались «соседним» процессорам в сети. Так и будет в большинстве наших примеров.

Мы начнем с основанной на передаче сообщений версии алгоритма 6.1.3.

**Алгоритм 6.1.4.** Предположим, что заданы  $A \in \mathbf{R}^{n \times n}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{R}^n$  и что  $z = y + Ax$ . Если в  $p$ -процессорном кольце каждый процессор исполняет следующую локальную программу и  $n = rp$ , то по окончании процесса  $\text{Proc}(\mu)$  содержит  $z(1 + (\mu - 1)r : \mu r)$  в  $y_{loc}$ .

```

loc.init [ $p, \mu = my.id., left, right, n, r = n/p, col = 1 + (\mu - 1)r : \mu r$ 
 $A_{loc} = A(:, col), x_{loc} = x(col), y_{loc} = y(col)]$ 
 $w = A_{loc}x_{loc}$ 
 $k = 1$ 
while  $k \leq p$ 
    send ( $w, right$ ); recv ( $w, left$ )
     $\tau = \tau - k$ 
    if  $\tau \leq 0$ 
         $\tau = \tau + p$ 
    end
     $\{w = A_\tau x_\tau\}$ 
     $y_{loc} = y_{loc} + w(1 + (\tau - 1)r : \tau r)$ 
     $k = k + 1$ 
end
quit
```

Хотя выглядит это очень похожим на свой систолический аналог – алгоритм 6.1.3, лежащая в основе логика здесь совершенно другая. Именно,  $\text{Proc}(\mu)$  узнает, что делать по индексу  $k$ , который отслеживает число принятых сообщений.

### 6.1.14. Дальнейшие примеры передачи сообщений

Поучительно рассмотреть некоторые вариации алгоритма 6.1.4, для того чтобы продолжить иллюстрацию парадигмы передачи сообщений.

В качестве первого примера предположим, что  $x$  и  $y$  целиком размещены в  $\text{Proc}(1)$  и что  $\text{Proc}(1)$  должен на месте  $y$  получить  $z = y + Ax$ . (Мы продолжаем считать, что  $A$  распределена по блочным столбцам.) Один из подходов к этой задаче – заставить  $\text{Proc}(1)$  передать копию  $x_\tau$  для  $\text{Proc}(\tau)$ . После этого  $\text{Proc}(\tau)$  вычисляет  $A_\tau x_\tau$  и возвращает результат в  $\text{Proc}(1)$ , где он используется для перевычисления  $y$ . Вот один из способов сделать это:

```

{Распределить  $x$  по процессорной сети.}
if  $\mu = 1$ 
    for  $\tau = 1:p$ 
        send ( $x(1 + (\tau - 1)r : \tau r), \tau$ )
    end
end
{Принять  $x_\mu$ , вычислить  $w = A_\mu x_\mu$ , послать  $w$  в  $\text{Proc}(1)$ .}
recv ( $x_{loc}, 1$ );  $w = A_{loc}x_{loc}$ ; send ( $w, 1$ )
{ $\text{Proc}(1)$  собирает векторы  $w$  и перевычисляет  $y$ .}
if  $\mu = 1$ 
    for  $\tau = 1:p$ 
        recv ( $w, \tau$ )
         $y = y + w$ 
    end
quit
```

Заметим, что  $\text{Proc}(1)$  посылает сообщение сам себе – вещь совершенно легальная в нашей модели. Скажем также о том, что предсказать коммуникационные издерж-

ки здесь труднее, чем для алгоритма 6.1.1, – из-за того, что обмениваются не соседи. При оценке производительности весьма важную роль играло бы системное обеспечение маршрутизации сообщений.

Теперь предположим, что  $\text{Proc}(\mu)$  содержит  $x(\text{col})$  и  $A(:, \text{col})$ , где  $\text{col} = 1 + (\mu - 1)r : \mu r$ , и что мы хотим на месте  $y$  получить  $z = y + Ax$ . Будем считать, что  $y$  помещается в  $\text{Proc}(1)$  в начале и в конце вычисления. Кажется разумным действовать следующим образом:

```

if  $\mu = 1$ 
     $y = y + A_{loc} x_{loc}$ 
    send( $y$ , right); recv( $y$ , left)
else
    recv( $y$ , left);  $y = y + A_{loc} x_{loc}$ ; send( $y$ , right)
end
quit

```

(6.1.6)

Логика этого подхода состоит в том, что  $y$  движется по кругу, вбирая в себя  $A_{loc} x_{loc}$  на каждой остановке. Проблема заключается в том, что здесь нет одновременно выполняемых вычислений. В любой момент только один процессор занят арифметической работой.

Чтобы это исправить, мы просто сделаем так, чтобы  $\text{Proc}(\mu)$  выполнял вычисления, прежде чем он будет ждать прибытия циркулирующего  $y$ :

```

if  $\mu = 1$ 
     $y = y + A_{loc} x_{loc}$ 
    send( $y$ , right); recv( $y$ , left)
else
     $w = A_{loc} x_{loc}$ 
    recv( $y$ , left);  $y = y + w$ ; send( $y$ , right)
end
quit

```

Теперь уровень проставивания намного сокращается в сравнении с (6.1.6).

## Задачи

- 6.1.1. Модифицировать алгоритм 6.1.1 так, чтобы  $n$  не обязательно было целым кратным  $r$ .
- 6.1.2. Модифицировать алгоритм 6.1.1 так, чтобы он эффективно работал в случае трехдиагональной  $A$ .
- 6.1.3. Модифицировать алгоритм 6.1.2 так, чтобы он мог работать в случае, когда матрица  $A$  является: а) нижней треугольной; б) блочной верхней треугольной.
- 6.1.4. Модифицировать алгоритмы 6.1.3 и 6.1.4 так, чтобы они на месте  $y$  получали  $z = y + A^m x$  для заданного положительного целого  $m$ , доступного каждому процессору.
- 6.1.5. Модифицировать алгоритмы 6.1.3 и 6.1.4 так, чтобы  $y$  заменялось вектором  $z = y + A^T A x$ .
- 6.1.6. Модифицировать алгоритм 6.1.3 так, чтобы по окончании локальный массив  $A_{loc}$  в  $\text{Proc}(\mu)$  содержал  $\mu$ -й блочный столбец для  $A + xy^T$ .

## Замечания и литература к § 6.1

К числу общих ссылок на параллельные матричные вычисления относятся

Dongarra J. J. and Sorensen D. C. (1986). “Linear Algebra on High Performance Computers”, Appl. Math. and Comp., 20, 57–88.

- Heller D. (1978). "A Survey of Parallel Algorithms in Numerical Linear Algebra", SIAM Review 20, 740–777.
- Фадеева В. Н., Фадеев Д. К. Параллельные вычисления в линейной алгебре.– Кибернетика, 1977, № 6, 28–40.
- Хокни Р., Джесхуп К. Параллельные ЭВМ.– М.: Радио и связь, 1986.
- Hockney R. W. and Jesshope C. R. (1988). Parallel Computers 2, Adam Hilger, Bristol and Philadelphia.
- Modi J. J. (1988). Parallel Algorithms and Matrix Computations, Oxford University Press, Oxford.
- Ortega J. M. and Voigt R. G. (1985). "Solution of Partial Differential Equations on Vector and parallel Computers", SIAM Review 27, 149–240.

Важной и трудной проблемой является оценка производительности на системах с распределенной памятью. См.

- Adams L. and Crockett T. (1984). "Modeling Algorithm Execution Time on Processor Arrays", Computer 17, 38–43.
- Gannon D. and Van Rosendale J. (1984). "On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithms", IEEE Trans. Comp. C-33, 1180–1194.
- Johnsson S. L. (1987). "Communication Efficient Basic Linear Algebra Computations on Hypercube Multiprocessors", J. Parallel and Distributed Computing, No. 4, 133–172.
- Saad Y. and Schultz M. H. (1985). "Data Communication in Hypercubes", Report YALEU/DCS/RR-428, Dept. of Computer Science, Yale University, New Haven, CT.
- Saad Y. and Schultz M. H. (1985). "Topological Properties of Hypercubes", Report YALEU/DCS/RR-389, Dept. of Computer Science, Yale University, New Haven, CT.

Вопросы реализации на параллельных системах обсуждаются в

- Connolly K., Dongarra J. J., Sorensen D., and Paterson J. (1988). "Programming Methodology and Performance Issues for Advanced Computer Architectures", Parallel Computing 5, 41–58.
- Dongarra J. J. and Sorensen D. C. (1987). "A Portable Environment for Developing Parallel Programs", Parallel Computing 5, 175–186.
- O'Leary D. P. and Stewart G. W. (1986). "Assignment and Scheduling in Parallel Matrix Factorization", Lin. Alg. and Its Applic. 77, 275–300.

Матричные вычисления на 2-мерных массивах обсуждаются в

- Kung H. T. (1982). "Why Systolic Architecture?", Computer 15, 37–46.
- O'Leary D. P. and Stewart G. W. (1985). "Data Flow Algorithms for Parallel Matrix Computations", Comm ACM 28, 841–853.

По поводу обсуждения базовых линейно-алгебраических вычислений на распределенной памяти см.

- Johnsson S. L. and Ho C. T. (1987). "Matrix Transposition on Boolean n-cube Configured Ensemble Architectures", Report YALEU/DCS/RR-574, Dept. of Computer Science, Yale University, New Haven, CT, to appear in SIAM J. Alg. and Discrete Methods.
- McBryan O. and van de Velde E. F. (1987). "Hypercube Algorithms and Implementations", SIAM J. Sci. and Stat. Comp. 8, 227–287.
- Moler C. B. (1986). "Matrix Computations on Distributed Memory Multiprocessors", in Proceedings of First SIAM Conference on Hypercube Multiprocessors, ed. M. T. Heath, SIAM Press, Philadelphia, Pa., 1986.

Хороший материал по матричным вычислениям на распределенной памяти содержится в трудах конференции, посвященной гиперкубу. См.

- Heath M. T. (ed.) (1986). Proceedings of First SIAM Conference on Hypercube Multiprocessors, SIAM Publications, Philadelphia, Pa.
- Heath M. T. (ed.) (1987). Hypercube Multiprocessors, SIAM Publications, Philadelphia, Pa.
- Fox G. (ed.) (1988). The Third Conference on Hypercube Concurrent Computers and Applications, Vol. II – Applications, ACM Press, New York.

### Добавления при переводе

Свежим обзором по параллельным вычислениям с плотными матрицами является

Gallivan K. A., Plemmons R. T., Sameh A. H. (1990). "Parallel algorithms for dense linear algebra computations", SIAM Review, 32, No. 1, 54–135.

Общий взгляд на параллельные вычислительные системы и процессы с точки зрения математика представлен в

Воеводин В. В. Математические модели и методы в параллельных процессах. М.: Наука, 1977.

## 6.2. Операция гахру на общей памяти

Теперь мы обсудим операцию гахру на многопроцессорных системах с общей памятью. В такой системе каждый процессор имеет доступ к общей, разделенной с другими памяти, как показано на следующем рисунке:

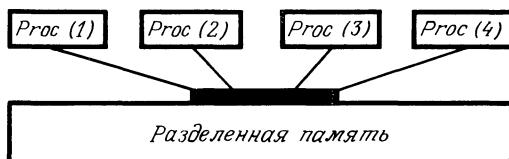


Рис. 6.2.1. 4-процессорная система с общей памятью.

Связь между процессорами достигается с помощью считывания и записи общих переменных, размещаемых в общей памяти. Каждый процессор выполняет свою локальную программу и имеет свою локальную память. В ходе вычислений данные текут туда и обратно из общей памяти. Например, для операции гахру ( $z = y + Ax$ )  $x$ ,  $y$  и  $A$  могут располагаться в общей памяти. Процессор, участвующий в обработке, может: а) скопировать  $x$ ,  $i$ -ю блочную строку  $A_i$  и  $i$ -й подвектор  $y_i$  из общей памяти в свою память; б) вычислить  $z_i = y_i + A_i x$ ; с) скопировать  $z_i$  из локальной памяти в общую память.

В модифицированной форме здесь присутствует все то, чем мы интересовались в случае вычислений на распределенной памяти. Как обсуждалось в предыдущем параграфе, вся процедура должна обеспечивать равномерную загруженность. Вычисления следует организовывать таким образом, чтобы отдельным процессорам приходилось как можно меньше простоять в ожидании какой-нибудь полезной работы. Потоки данных между общей памятью и локальными должны управляться с особой аккуратностью, так как они, по нашему мнению, являются значительным накладным расходом. (Это соответствует межпроцессорным коммуникациям в случае распределенной памяти.) Природа физического соединения между процессорами и общей памятью очень важна и может влиять на разработку алгоритмов. Например, если в любой заданный момент времени доступ к общей памяти может получить только один процессор, то следует позаботиться о том, чтобы свести к минимуму возможность конфликтов, связанных с одновременным обращением к общей памяти. Большой частью мы стараемся не опускаться до столь подробной детализации, предпочитая рассматривать этот аспект системы как черный ящик, изображенный на рис. 6.2.1.

Все локальные вычисления подчинены традиционным требованиям эффективности. Если отдельные процессоры являются векторными/конвейерными процессорами, то, как обсуждалось в § 1.4, могут быть важными такие вещи, как шаг и длина вектора, число обращений к локальной памяти повторного использования данных.

Чтобы познакомиться с матричными вычислениями на общей памяти, полезно еще раз рассмотреть параллельную реализацию для гахру. Сначала мы изложим такой алгоритм для гахру, в котором задача каждому процессору ставится заранее. Для менее регулярных вычислений необходимо назначать задачи процессорам по ходу вычислений. Динамическое назначение задач требует специальных средств синхронизации, и для этого мы вводим понятие монитора. Используя мониторы, мы можем найти элегантные решения для широкого класса проблем планирования вычислений.

### 6.2.1. Статическое планирование операции гахру

Всюду далее будем считать, что наш модельный компьютер с общей памятью имеет  $p$  процессоров и  $\mu$ -й процессор обозначается  $\text{Proc}(\mu)$ . Как и в предыдущем параграфе, рассмотрим следующее блочное разбиение  $n \times n$ -задачи (гахру)  $z = y + Ax$ :

$$\begin{bmatrix} z_1 \\ \vdots \\ z_p \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} + \begin{bmatrix} A_1 \\ \vdots \\ A_p \end{bmatrix} x. \quad (6.2.1)$$

Здесь мы полагаем, что  $n = rp$  и  $A_\mu \in \mathbf{R}^{r \times n}$ ,  $y_\mu \in \mathbf{R}^r$ ,  $z_\mu \in \mathbf{R}^r$ . Наши обозначения, связанные с общей памятью, мы введем, формируя следующий алгоритм, в котором в  $\text{Proc}(\mu)$  на месте  $y_\mu$  получается  $z_\mu = y_\mu + A_\mu x$ .

**Алгоритм 6.2.1.** Предположим, что  $A \in \mathbf{R}^{n \times n}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{R}^n$  располагаются в общей памяти, к которой имеют доступ  $p$  процессоров. Если  $n = rp$  и каждый процессор выполняет следующий алгоритм, то по окончании процесса на месте  $y$  получается  $z = y + Ax$ .

```

shared variables [A(1:n, 1:n), x(1:n), y(1:n)]
glob. init [A, x, y]
loc. init [p, μ = my.id, n]
r = n/p; row = 1 + (μ - 1)r:μr
get(x, xloc); get(y(row), yloc)
for j = 1:n
    . get(A(row, j), aloc); yloc = yloc + aloc · x(j)
end
put(yloc, y(row))
end
quit
```

Мы предполагаем, что в каждом процессоре находится копия этой программы. Все переменные являются локальными, за исключением тех, которые описаны оператором **shared variables**. Чтобы подчеркнуть локальность переменной, мы часто используем нижний индекс *loc*. Оператор **glob. init** указывает, какие общие переменные должны инициализироваться. В операторе **loc. init** указываются локальные переменные, которые должны инициализироваться, в их числе и обозначение процессора  $\mu$ . Мы не будем думать ни о том, каким образом осуществляются эти инициализации, ни о том, как загружаются локальные программы или как организуется их параллельное выполнение. Это – важные детали, связанные с реализацией, но они выходят за рамки нашего уровня изложения. Оператор **quit** напоминает нам, что

окончание программы на одном процессоре не означает, что все вычисления закончены.

Исполняемая часть локальной программы следует за определением общих переменных и локальными и глобальными инициализациями. В общую память и обратно данные пересылаются с помощью операторов **get** и **put**. Например, **get**( $A(row, j)$ ,  $a_{loc}$ ) копирует часть столбца  $A(row, j)$  из общей памяти в локальное рабочее пространство  $a_{loc}$ . Аналогично, после того как **Proc**( $\mu$ ) завершает вычисление вектора  $y_\mu + A_\mu x$ , он возвращает его в общую память с помощью предложения **put**( $y_{loc}$ ,  $y(row)$ ).

Заметим, что в алгоритме 6.2.1 только один процессор делает записи в заданном месте общей памяти, отведенной для  $y$ . Таким образом, в синхронизации здесь нет необходимости. Каждый процессор имеет совершенно независимую часть всех вычислений, связанных с  $gaxpy$ , и ему не нужно следить за тем, что делают другие процессоры. Мы говорим, что алгоритм допускает статическое планирование работы, потому что ее распределение осуществляется перед выполнением.

### 6.2.2. Обмен данными с общей памятью

Важно учитывать накладные расходы, связанные с операторами **get** и **put**. Они соответствуют издержкам, связанным с **send** и **recv** для вычислений на распределенной памяти. Будем считать, что в общем случае **put** и **get** имеют вид

```
get({матрица в общей памяти}, {матрица в локальной памяти})  
put({матрица в локальной памяти}, {матрица в общей памяти})
```

Полагаем, что размеры общих и локальных матриц одинаковы, а их элементы в соответствующей памяти располагаются без пропусков. В алгоритме 6.2.1 предполагается, что  $A$  хранится в общей памяти по столбцам. Таким образом,  $A_\mu$  не занимает в памяти участка без пропусков и, значит, должна копироваться в локальную память столбец за столбцом.

Если **get** или **put** относятся к  $m$  числам с плавающей точкой, то мы определяем время передачи как

$$\text{trans}(m) = \alpha_s + \beta_s m. \quad (6.2.2)$$

Здесь  $\alpha_s$  представляет собой стартовое время, а  $\beta_s$  – соответствующую скорость передачи. Отметим сходство этих затрат с нашей моделью в § 1.4.3 для времени выполнения векторной операции и с нашей моделью (6.1.4) для обменов в системе с распределенной памятью.

Учитывая все операторы **get** и **put** в алгоритме 6.2.1, мы видим, что время, которое каждый процессор тратит на обмен с общей памятью, оценивается как

$$T_{6,2,1} = (n + 3)\alpha_s + (n + 2r + nr)\beta_s. \quad (6.2.3)$$

Однако могут быть и скрытые задержки, связанные с каждым **get** и **put**, потому что процессоры могут вступить в конфликт при одновременном обращении к общей памяти. Таким образом, важно помнить, что для алгоритма 6.2.1 (6.2.3) является *оценкой* накладных расходов на обмен с общей памятью.

### 6.2.3. Равномерная загруженность

Алгоритм 6.2.1 дает прекрасную равномерную загруженность. Однако если  $A$  нижняя треугольная матрица, то необходимо перераспределить загрузку процессоров таким образом, чтобы обеспечить каждому из них примерно одинаковое

количество полезной работы. Один из способов добиться этого – сделать так, чтобы Proc( $\mu$ ) вычислял  $z(\mu:p:n) = y(\mu:p:n) + A(\mu:p:n)x$ . Детали мы оставляем читателю, поскольку искомая процедура очень близка к алгоритму 6.1.2.

#### 6.2.4. Синхронизация с помощью барьеров

Предположим, что мы хотим на месте  $y$  получить единичный вектор, направленный так же, как вектор  $z = y + Ax$ , который мы считаем ненулевым. Наш подход к этой задаче состоит в том, чтобы вычислить  $z$ , используя параллелизм алгоритма 6.2.1, а затем заставить какой-либо один процессор, скажем Proc(1), выполнить нормировку  $z/\|z\|_2$ . На первый взгляд, для этой задачи *гахру* с нормировкой вполне разумным кажется следующий подход:

```
shared variables [A(1:n, 1:n), x(1:n), y(1:n), s]
glob.init [A, x, y]
loc.init [p, mu = my.id, n]
r = n/p; row = 1 + (mu - 1)r:mu r
get(x, xloc); get(y(row), yloc)
for j = 1:n
    get(A(row, j), aloc); yloc = yloc + alocx(j)
end
put(yloc, y(row))
if mu = 1
    get(y, uloc); sloc = \|uloc\|2; put(sloc, s)
end
get(s, sloc); yloc = yloc/s; put(yloc, y(row))
quit
```

Однако в этом подходе есть два серьезных изъяна. У нас нет никакой гарантии, что  $y$  содержит  $z$ , когда Proc(1) начинает вычисление 2-нормы, и нет гарантии, что  $s = \|z\|_2$ , когда Proc(2), ..., Proc( $p$ ) начинают нормирование.

Чтобы исправить эти недостатки, нам нужно какое-то средство для синхронизации, которое может задержать вычисление 2-нормы до тех пор, пока все процессоры не вычислят и не запомнят свою порцию вектора  $z$ , и которое может задержать нормирование до тех пор, пока не будет получено  $s$ . Для этой цели мы вводим конструкцию барьера:

**Алгоритм 6.2.2.** Предположим, что  $A \in \mathbb{R}^{n \times n}$ ,  $x \in \mathbb{R}^n$  и  $y \in \mathbb{R}^n$  размещаются в общей памяти, доступной для  $p$  процессоров. Если  $n = rp$  и каждый процессор исполняет следующий алгоритм, то по окончании процесса на месте  $y$  получается  $z/\|z\|_2$ , где вектор  $z = y + Ax$  предполагается ненулевым.

```
shared variables [A(1:n, 1:n), x(1:n), y(1:n), s]
glob.init [A, x, y]
loc.init [p, mu = my.id, n]
r = n/p; row = 1 + (mu - 1)r:mu r
get(x, xloc); get(y(row), yloc)
for j = 1:n
    get(A(row, j), aloc); yloc = yloc + alocx(j)
end
put(yloc, y(row))
barrier
if mu = 1
    get(y, uloc); sloc = \|uloc\|2; put(sloc, s)
```

```

end
barrier
get( $s, s_{loc}$ );  $y_{loc} = y_{loc}/s$ ; put( $y_{loc}, y(row)$ )
quit

```

Чтобы понять, что делает **barrier**, удобно считать, что процессор либо блокирован, либо свободен. Процессор блокирован и откладывает выполнение программ, если он выполняет **barrier**. После блокировки  $p$ -го процессора все остальные процессоры находятся в «свободном состоянии» и продолжают работу. Проще говоря, **barrier** – это препятствие. Все  $p$  процессоров собираются возле препятствия, и после этого они все вместе перепрыгивают через него.

В алгоритме 6.2.2 Proc( $\mu$ ) вычисляет  $z_\mu$  и немедленно блокируется. Нельзя предсказать порядок, в котором блокируются процессоры. Мы знаем лишь, что работа возобновляется в каждом процессоре, как только последний процессор достигнет барьера. Тогда Proc(1) продолжает работу, вычисляя  $s = \|z\|_2$ , но остальные процессоры немедленно блокируются снова. Наконец, когда получено  $s$ , каждый процессор освобождается и может продолжить работу, выполняя свою порцию нормирования.

### 6.2.5. Парадигма динамического резерва задач

Оператор **barrier** является полезной конструкцией для крупноблочных вычислений с равномерной загруженностью. Однако злоупотреблять им не следует, так как это увеличивает всеобщее простаивание. В тех случаях, когда равномерное распределение работы трудно организовать заранее, нужен какой-то более гибкий подход к синхронизации.

Чтобы мотивировать ключевые идеи, мы изложим параллельный алгоритм для  $n \times n$ -операции  $gaxpy(z = y + Ax)$ , основанный на общем блочном построчном разбиении, т. е. на разбиении вида

$$\begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} + \begin{bmatrix} A_1 \\ \vdots \\ A_N \end{bmatrix} x, \quad (6.2.4)$$

где  $y_i, z_i \in \mathbf{R}^{n_i}$  и  $A_i \in \mathbf{R}^{n_i \times n}$ ,  $i = 1:N$ . Полагаем, что элементы матрицы  $A_i$  располагаются в общей памяти без пропусков. Определим целочисленные векторы *first*(1:N) и *last*(1:N) как

$$\begin{aligned} \textit{first}(i) &= 1 + n_1 + \dots + n_{i-1}, \\ \textit{last}(i) &= n_1 + \dots + n_i. \end{aligned}$$

Вот последовательный алгоритм, основанный на блочном разбиении (6.2.4) и получающий  $z = y + Ax$  на месте  $y$ :

```

for  $i = 1:n$ 
     $row = \textit{first}(i):\textit{last}(i)$ 
     $y(row) = y(row) + A(row, :)x$ 
end

```

Заметим, что цикл определяет  $N$  независимых вычислений, и поэтому при разработке параллельного алгоритма нет каких-либо проблем. Вычисление  $z_i$ ,  $i = \mu:p:N$ , можно было бы назначить процессору Proc( $\mu$ ). Однако итоговая процедура может не давать равномерной загруженности, если строчные размеры блоков  $A_i$  разные.

Это подталкивает нас к разработке какого-либо динамического метода распределения работы. До сих пор мы рассматривали лишь «хорошие» блочные разбиения для вычисления гахру, которые позволяли нам заранее предложить удовлетворительное распределение работы. Такое распределение называется статическим планированием. Наш подход к динамическому планированию основываетя на идеи общего резерва<sup>1)</sup> задач. Чтобы проиллюстрировать этот подход, будем рассматривать вычисления, относящиеся к  $i$ -му проходу цикла в (6.2.5), как  $i$ -ю задачу. В подходе, связанном с резервом задач, поддерживается список оставшихся задач. Когда процессор завершает задачу, он сверяется со списком. Если список пуст, то нет никакой работы, которую можно было бы назначить этому процессору. Если список не пуст, то процессор выбирает задачу из «общего котла» оставшихся задач. После этого выбранная задача убирается из резерва.

Чтобы применить эту идею к (6.2.5), разумно завести глобальную переменную  $count$ , сохраняющую номер последней выбранной задачи. Это может быть сделано (или кажется, что это так), если инициализировать  $count$  нулем и затем увеличивать ее значение каждый раз, когда процессор берется за новую задачу:

```
shared variables [ $A(1:n, 1:n)$ ,  $x(1:n)$ ,  $y(1:n)$ ,  $count$ ]
glob. init [ $A$ ,  $x$ ,  $y$ ,  $count = 1$ ]
loc. init [ $p$ ,  $\mu = my.id$ ,  $n$ ,  $N$ ,  $first(1:N)$ ,  $last(1:N)$ ]
get ( $x$ ,  $x_{loc}$ )
get ( $count$ ,  $i$ );  $i = i + 1$ ; put ( $i$ ,  $count$ )
while  $i \leq N$ 
     $row = first(i) : last(i)$ 
    get ( $A(row, :)$ ,  $A_{loc}$ ); get ( $y(row)$ ,  $y_{loc}$ )
     $y_{loc} = y_{loc} + A_{loc}x_{loc}$ 
    put ( $y_{loc}$ ,  $y(row)$ )
    get ( $count$ ,  $i$ );  $i = i + 1$ ; put ( $i$ ,  $count$ )
end
```

Неприятность, связанная с этим подходом к динамическому планированию, заключается в том, что увеличение переменной  $count$  не контролируется. Два различных процессора могут «захватить» одно и то же значение переменной  $count$ , если соответствующие им операции

$$\text{get}(count, i); \quad i = i + 1; \quad \text{put}(i, count) \quad (6.2.6)$$

перекрываются во времени. Таким образом, нам нужен какой-либо способ, гарантирующий, что увеличением счетчика согласно (6.2.6) в любой момент занимается не более чем один процессор. Будем говорить, что (6.2.6) является *критическим разделом алгоритма*.

## 6.2.6. Мониторы

В решении проблемы критических разделов могут использоваться *мониторы*. Проиллюстрируем эту концепцию на примере идеи резерва задач. Ключом к решению является разработка специальной процедуры

**nexti.index** ( $p$ ,  $N$ , *more*,  $i$ ),

которая имеет на входе  $p$  (число процессоров, назначенных на параллельное выполнение цикла) и  $N$  (число задач). На выходе процедуры – логическое значение

---

<sup>1)</sup> В отечественной литературе по программированию такой динамический резерв задач иногда называют пулом; в оригинале – pool. – Прим. перев.

в *more* и индекс в *i*, обладающие следующим свойством:

*more* = *true*  $\Rightarrow$  назначается *i*-я задача;  
*more* = *false*  $\Rightarrow$  нет оставшихся задач.

Если мы можем гарантировать, что в любой момент только один процессор выполняет **nexti.index**, то программа

```
call nexti.index(p, N, more, i)
while more = true
    На месте  $y_i$  получить  $z_i = y_i + A_i x$ 
    call nexti.index(p, N, more, i)
end
```

(6.2.7)

выполняет вычисления корректно. Чтобы определить вычисление этого типа, мы расположим **nexti.index** внутри монитора:

```
monitor : nexti
    protected variables: flag, count, idle.proc
    condition variables: waiti
    initializations: flag = 0
    procedure nexti.index(p, N, more, i)
        if flag = 0
            count = 0; flag = 1; idle.proc = 0
        end
        count = count + 1
        if count  $\leq N$ 
            more = true; i = count
        else
            more = false; idle.proc = idle.proc + 1
            if idle.proc  $\leq p - 1$ 
                delay(waiti)
            else
                flag = 0
            end
            continue(waiti)
        end
    end nexti.index
end nexti
```

Из этого примера видно, что монитор имеет имя и в его состав входят операторы *protected variables* (защищенные переменные), *condition variables* (переменные условия), *initializations* (инициализации) и одна или несколько процедур для мониторинга; в нашем случае монитор имеет одну процедуру: **nexti.index**. Главная задача, возложенная на монитор, состоит в том, чтобы аккуратно поддерживать значение счетчика *count*, изменяющееся от 1 до *N*, т. е. число задач, которые должны обрабатываться на *p* процессорах. Когда уже не остается задач, монитор инициализирует процесс «перематывания назад». Переменная *idle.proc* отслеживает число процессоров, которым не удалось заполучить новую задачу. Когда этот счетчик достигает *p*, работа монитора заканчивается и он возвращается к своему начальному состоянию. Механику этого процесса лучше всего описать после того, как мы увидим монитор в действии.

**Алгоритм 6.2.3.** Предположим, что  $A \in \mathbb{R}^{n \times n}$ ,  $x \in \mathbb{R}^n$  и  $y \in \mathbb{R}^n$  располагаются в общей памяти, доступной для *p* процессоров. Предположим, что *A* хранится по блочным

строкам и  $i$ -я блочная строка записывается как  $A(first(i):last(i), :)$ . Если каждый процессор исполняет следующий алгоритм, то по окончании процесса на месте  $y$  получается  $z = y + Ax$ .

```

shared variables [ $A(1:n, 1:n)$ ,  $x(1:n)$ ,  $y(1:n)$ ]
glob. init [ $A$ ,  $x$ ,  $y$ ]
loc. init [ $p$ ,  $\mu = my.id$ ,  $N$ ,  $first(1:N)$ ,  $last(1:N)$ ]
get( $x$ ,  $x_{loc}$ )
call nexti.index( $p$ ,  $N$ ,  $more$ ,  $i$ )
while  $more = true$ 
     $row = first(i):last(i)$ 
    get( $A(row, :)$ ,  $A_{loc}$ ); get( $y(row)$ ,  $y_{loc}$ )
     $y_{loc} = y_{loc} + A_{loc}x_{loc}$ 
    put( $y_{loc}$ ,  $y(row)$ )
    call nexti.index( $p$ ,  $N$ ,  $more$ ,  $i$ )
end
quit
```

Этот пример мы используем для того, чтобы объяснить основные особенности использования мониторов.

- Процессор пытается «войти» в монитор с помощью вызова своих процедур. Таким образом, оператор **«call nexti.index**( $p$ ,  $N$ ,  $more$ ,  $i$ )» представляет собой попытку исполняющего процессора войти в **nexti**.
- Внутри монитора процессор является либо активным, либо ожидающим. В любой момент времени в мониторе может быть не более одного активного процессора. Значит, если Proc(1) находится в **nexti** и Proc(2) вызывает **nexti.index**, то в момент вызова в своей локальной программе Proc(2) становится ожидающим.
- Процессор становится ожидающим внутри монитора, когда он встречает оператор вида **delay**( $\{\text{переменная условия}\}$ ). Смысл переменной условия в том, что она указывает очередь, и когда процессор становится ожидающим, его *id* помещается в очередь. Если процессор является ожидающим, то монитор становится свободным, и если какие-либо процессоры стремятся войти в монитор, то в точности один получит разрешение на вход. Пусть Proc( $\mu$ ) входит в монитор **nexti** и *count* имеет значение  $N$ , т. е. нет задач для выбора. Тогда Proc( $\mu$ ) становится ожидающим. Заметим, что прежде чем стать ожидающим, Proc( $\mu$ ) полагает *more* = *false* и увеличивает значение *idle proc*. Таким образом, когда управление передается назад в локальную программу, которая вызывала **nexti.index**, будет «известно», что задач больше нет. Именно это дает возможность работать циклу **while** в алгоритме 6.2.3.
- Когда текущий активный процессор выполняет оператор вида **continue**( $\{\text{переменная условия}\}$ ), он немедленно покидает монитор с несколькими вариантами продолжения. Если очередь, указанная в **continue**, непуста, то в точности один из ожидающих процессоров становится следующим активным процессором. Если бы указанная очередь была пустой, то монитор был бы открыт для входа. Если при этом процессоры ждут допуска в монитор, то ровно одному из них будет разрешено войти. Если в **nexti.index** защищенная переменная *idle proc* имеет значение  $p$ , то все вычислительные задачи завершены. Тот процессор, который первым обнаружит, что задач больше нет, устанавливает *flag* на нуль и дает старт процессу активизации  $p - 1$  ожидающих процессоров посредством выполнения оператора **continue(wait)**. После «пробуждения» процессор возобновляет работу. В **nexti.index** «разбуженный» процессор не-

медленно выполняет `continue` и тем самым активизирует еще один ожидающий процессор. Этот процесс активизации продолжается до тех пор, пока все  $p$  процессоров не уйдут из монитора.

- Значение защищенной переменной в мониторе может быть изменено только процедурой в мониторе. Заметим, что вызывающие программы не могут испортить `flag`, `count` или `idle.proc`.
- Защищенные переменные инициализируются только один раз – до каких-либо обращений к монитору. В `nexti` переменная `flag` инициализируется нулем. В то время когда `nexti` следит за распределением задач, `flag` имеет значение 1. Он устанавливается на нуль первым процессором, который обнаружит, что задач больше нет.

Имеет смысл затратить немного времени и разобраться шаг за шагом с этой довольно сложной логикой, для того чтобы понять соображения, лежащие в основе динамически планируемых матричных вычислений.

### 6.2.7. Столбцовая реализация операции `gaxpy`

В качестве второй иллюстрации использования монитора и идеи резерва задач мы изложим параллельную реализацию для `gaxpy`, основанную на следующем блочном столбцовом разбиении для  $z = y + Ax$ :

$$z = y + [A_1, \dots, A_N] \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = y + A_1 x_1 + \dots + A_N x_N. \quad (6.2.8)$$

Мы не делаем никаких предположений относительно ширины блочных столбцов. В параллельном алгоритме, основанном на этом блочном разбиении, разумно поручить `Proc( $\mu$ )` вычисление  $w_k = A_k x_k$ , где  $k = \mu:p:N$ . Чтобы завершить `gaxpy`, нужно просуммировать частичные векторные суммы. На первый взгляд корректное решение вроде бы дается следующей процедурой:

```
shared variables [A(1:n, 1:n), x(1:n), y(1:n)]
glob. init [A, x, y]
loc. init [p, mu = my.id, N]
for k = mu:p:N
    get(x_k, x_loc); get(A_k, A_loc)
    w_loc = A_loc * x_loc
    get(y, y_loc); y_loc = y_loc + w_loc; put(y_loc, y)
end
quit
```

Однако возникают проблемы, если выполнение модификации

$$\text{get}(y, y_{loc}); \quad y_{loc} = y_{loc} + w_{loc}; \quad \text{put}(y_{loc}, y) \quad (6.2.9)$$

не управляется должным образом. Например, может быть потеряно то, что вычислит `Proc(1)`, если `Proc(2)` читает `y` в то время, когда `Proc(1)` находится на полупути при выполнении (6.2.9). Все будет работать, если процессоры не смешивают свое обращение к `y`. Ситуация требует монитора, управляющего вычислением суммы векторов, получаемых несколькими процессорами:

```

monitor : gax
  procedure gax.add( $u, v$ )
    get( $v, v_{loc}$ );  $v_{loc} = v_{loc} + u$ ; put( $v_{loc}, v$ )
  end gax.add
end gax

```

Этот простой монитор не имеет защищенных переменных или переменных условия. В процедуре **gax.add** два аргумента. Первый – это локальный вектор, и вызывающий процессор хочет включить его в общую накапливаемую сумму, обозначенную вторым аргументом. Используя этот монитор, мы можем вести безопасное накапливание суммы в течение вычисления гахру.

**Алгоритм 6.2.4.** Предположим, что  $A \in \mathbf{R}^{n \times n}$ ,  $x \in \mathbf{R}^n$  и  $y \in \mathbf{R}^n$  размещаются в общей памяти, доступной для  $p$  процессоров, и пусть мы имеем блочное разбиение (6.2.8). Если каждый процессор исполняет следующий алгоритм, то по окончании процесса на месте  $u$  будет  $z = y + Ax$ .

```

shared variables [ $A(1:n, 1:n)$ ,  $x(1:n)$ ,  $y(1:n)$ ]
glob. init [ $A$ ,  $x$ ,  $y$ ]
loc. init [ $p$ ,  $\mu = my.id$ ,  $N$ ]
call nexti.index( $p, N, more, k$ )
while more = true
  get( $A_k, A_{loc}$ ); get( $x_k, x_{loc}$ );  $w_{loc} = A_{loc}x_{loc}$ 
  call gax.add( $w_{loc}, y$ )
  call nexti.index( $p, n, more, k$ )
end
quit

```

Итак, когда Proc( $\mu$ ) вызывает **gax.add**, к  $y$  прибавляется еще один вектор. Свойства монитора гарантируют, что в любой заданный момент только один процессор модифицирует  $y$ . Заметим, что динамический резерв задач в алгоритме 6.2.4 управляется процедурой мониторинга **nexti.index**, разработанной в § 6.2.6.

Алгоритм 6.2.4 выявляет некоторые взаимосвязи между степенью дробления вычислений, равномерной загруженностью и уровнем производительности. При возрастании  $N$  объем вычислений, связанных с одной задачей, уменьшается. Это влечет за собой более равномерную загруженность, поскольку уменьшается простояние процессоров в конце вычислений, когда они ожидают, пока обнаружится, что задач больше нет. С другой стороны, с ростом  $N$  локальное вычисление  $w_{loc} = A_{loc}x_{loc}$  все меньше и меньше напоминает операцию 2-го уровня, так как уменьшается ширина блочных столбцов в  $A_{loc}$ . Таким образом, в этой задаче есть некоторое несоответствие между равномерной загруженностью и крупноблочной линейной алгеброй. Разрешение этого противоречия, возможно, потребует хитрого блочного разбиения (6.2.8) с переменной шириной, например, чтобы блочные столбцы сужались по направлению к правому краю матрицы.

### Задачи

- 6.2.1. Модифицировать алгоритм 6.2.2 и 6.2.3 так, чтобы на месте  $A$  получалась ее малоранговая модификация  $A + xy^T$ .
- 6.2.2. Модифицировать алгоритм 6.2.2 так, чтобы на месте  $u$  получалось  $A^2 x$ .
- 6.2.3. Модифицировать алгоритм 6.2.2 так, чтобы на месте  $u$  получался вектор с единичной 2-нормой, направленный как  $y + A^k x$ .
- 6.2.4. Модифицировать алгоритм 6.2.1 так, чтобы он эффективно работал в случае нижней треугольной  $A$ . См. § 6.2.3.

**6.2.5.** В этой задаче изучается что будет, если изменить определение «следующей задачи» для алгоритма 6.2.4. Рассмотрим  $n \times n$ -задачу гахру ( $y \leftarrow y + A x$ ), и пусть задано  $m \leq n$ . Предположим, что какому-то процессору уже назначено вычисление  $A(:, 1: last) x(1:last)$ , где  $1 \leq last \leq n$ . В качестве «следующей задачи» возьмем присоединение  $A(:, last + 1: last + k) x(last + 1:last + k)$  к накапливаемой векторной сумме  $y$ , где  $k = \min(m, \max(1, \text{ceil}((n - last)/p)))$ . Модифицируйте монитор `nexti` и алгоритм 6.2.4 так, чтобы реализовывалось это новое определение следующей задачи. Подсказка: Вам нужна защищенная переменная `last` и процедура мониторинга вида `nexti, indices(p, n, more, i, j)`, возвращающая индексы  $i$  и  $j$ , определяющие  $y \leftarrow y + A(:, i:j) x(i:j)$  как новую назначаемую задачу.

### Замечания и литература к § 6.2

Хорошее обсуждение мониторов и смежных понятий можно найти в любой из следующих работ:

Andrews G. and Schneider F. B. (1983). “Concepts and Notations for Concurrent Programming”, Computing Surveys 15, 1–43.

Boyle J., Butler R., Disz T., Glickfield B., Lusk E., Overbeek R., Patterson J., and Stevens R. (1987). Portable Programs for Parallel Processors, Holt, Rinehart and Winston.

## 6.3. Параллельное умножение матриц

В этом параграфе рассматривается задача умножения матрицы на матрицу  $D = C + AB$  для многопроцессорных систем с распределенной и общей памятью. Рассматриваются процедуры для блочной операции гахру и для блочного скалярного произведения.

### 6.3.1. Процедуры для блочной операции

Предположим, что  $A$ ,  $B$  и  $C$  – это  $n \times n$ -матрицы, и мы хотим вычислить

$$D = C + AB.$$

Для ясности будем считать, что  $n = rp$ , где  $p$  – число процессоров, и пусть имеет место блочное разбиение

$$[D_1, \dots, D_p] = [C_1, \dots, C_p] + [A_1, \dots, A_p][B_1, \dots, B_p], \quad (6.3.1)$$

где каждый блочный столбец имеет ширину  $r$ . Если

$$B_j = \begin{bmatrix} B_{1j} \\ \vdots \\ B_{pj} \end{bmatrix}, \quad B_{ij} \in \mathbb{R}^{r \times r},$$

то один из способов распределения работы – поручить  $\text{Прос}(\mu)$  вычисление

$$D_\mu = C_\mu + AB_\mu = C_\mu + \sum_{\tau=1}^p A_\tau B_{\tau\mu}.$$

Сначала мы опишем реализацию на распределенной памяти в предположении, что  $\text{Прос}(\mu)$  размещает  $A_\mu$ ,  $B_\mu$  и  $C_\mu$  и по окончании процесса на месте  $C_\mu$  получается  $D_\mu$ . Заметим, что вычисление  $D_\mu$  включает локальные данные ( $A_\mu$ ,  $B_\mu$  и  $C_\mu$ ) и недлокальные данные (остальные  $A_\tau$ ). Таким образом, по ходу вычислений каждый процессор должен иметь доступ к каждому столбцу матрицы  $A$ .

Простой способ сделать это состоит в том, чтобы заставить  $A_\tau$  двигаться по кругу наподобие «карусели» так же, как мы врашали подвекторы для  $x$  в нашей процедуре гахру на распределенной памяти. Вот что получается:

**Алгоритм 6.3.1.** Предположим, что  $A$ ,  $B$  и  $C$  суть  $n \times n$ -матрицы и  $D = C + AB$ . Если в  $p$ -процессорном кольце каждый процессор исполняет следующую программу и  $n = rp$ , то по окончании процесса  $\text{Proc}(\mu)$  содержит  $D_\mu$  в локальном массиве  $C_{loc}$ .

```

loc.init [ $p$ ,  $\mu = my.id$ ,  $left$ ,  $right$ ,  $n$ ,  $r = n/p$ ,  $col = 1 + (\mu - 1)r:\mu r$ ,
 $A_{loc} = A(:, col)$ ,  $B_{loc} = B(:, col)$ ,  $C_{col} = C(:, col)$ ]
for  $k = 1:p$ 
    send ( $A_{loc}$ ,  $right$ ); recv ( $A_{loc}$ ,  $left$ )
     $\tau = \mu - k$ 
    if  $\tau \leq 0$ 
         $\tau = \tau + p$ 
    end
     $\{A_{loc} = A_\tau = A(:, 1 + (\tau - 1)r:\tau r)\}$ 
     $C_{loc} = C_{loc} + A_{loc} B_{loc} (1 + (\tau - 1)r:\tau r, :)$ 
end
quit
```

Эта процедура дает равномерную загруженность. Если считать, что нет простояивания и нет перекрытия во времени вычислений и обменов, то каждый процессор тратит на обмены время

$$T_{6.3.1} = 2p(a_d + \beta_d n^2/p) = 2p a_d + 2\beta_d n^2, \quad (6.3.2)$$

поскольку здесь  $p$  операторов **send**,  $p$  операторов **recv**, а каждый из них имеет дело с сообщением длины  $nr = n^2/p$ .

Теперь построим процедуру для общей памяти на базе того же самого блочного разбиения 6.3.1. Следуя алгоритму 6.3.1, мы устроим все таким образом, что  $\text{Proc}(\mu)$  будет вычислять  $D_\mu$  с помощью суммирования

$$D_\mu = C_\mu + A_{\mu-1} B_{\mu-1} + \dots + A_1 B_{1\mu} + A_p B_{p\mu} + \dots + A_\mu B_{\mu\mu}. \quad (6.3.3)$$

«Неровный» доступ к  $A$  имеет целью сокращение возможности одновременного обращения двух процессоров к одному и тому же блоку  $A_j$ .

Вот детали:

**Алгоритм 6.3.2.** Предположим, что  $A$ ,  $B$  и  $C$  суть  $n \times n$ -матрицы, располагающиеся в общей памяти, доступной для  $p$  процессоров. Пусть  $n = pr$ . Если каждый процессор исполняет следующий алгоритм, то по окончании работы на месте  $C$  получается  $D = C + AB$ .

```

shared variables [ $A(1:n, 1:n)$ ,  $B(1:n, 1:n)$ ,  $C(1:n, 1:n)$ ]
glob.init [ $A$ ,  $B$ ,  $C$ ]
loc.init [ $p$ ,  $\mu = my.id$ ,  $n$ ,  $r = n/p$ ]
get ( $B(:, 1 + (\mu - 1)r:\mu r)$ ,  $B_{loc}$ )
get ( $C(:, 1 + (\mu - 1)r:\mu r)$ ,  $C_{loc}$ )
for  $k = 1:p$ 
     $i = \mu - k$ 
    if  $i \leq 0$ 
         $\tau = \tau + p$ 
    end
    get ( $A(:, 1 + (\tau - 1)r:\tau r)$ ,  $A_{loc}$ )
     $\{A_{loc} = A_\tau = A(:, 1 + (\tau - 1)r:\tau r)\}$ 
     $C_{loc} = C_{loc} + A_{loc} B_{loc} (1 + (\tau - 1)r:\tau r, :)$ 
end
put ( $C_{loc}$ ,  $C(:, 1 + (\mu - 1)r:\mu r)$ )
quit
```

Заметим, что в этой дающей равномерную загруженность процедуре каждое  $C_\mu$  связано с  $p + 2$  предложениями `get` и одним предложением `put`. Следовательно, время, которое каждый процессор тратит на обмены с общей памятью, приблизительно равно

$$T_{6.3.2} = (p + 3)(\alpha_s + \beta_s n^2/p). \quad (6.3.4)$$

Ширина  $r$  блочных столбцов в (6.3.1) полностью определяется размером задачи и числом процессоров:  $r = n/p$ . Важно понимать, что это строгое соотношение не является ни необходимым, ни желательным. Например,  $nr$  может превышать допустимую длину сообщения в системе с распределенной памятью. В системе с общей памятью может так случиться, что три  $n \times r$ -массива, требуемых в алгоритме 6.3.2, не помещаются в локальной памяти. Чтобы обойти эти возможные ограничения, мы покажем в общих чертах, как приспособить наши два алгоритма к произвольному блочному разбиению

$$[D_1, \dots, D_N] = [C_1, \dots, C_N] + [A_1, \dots, A_N][B_1, \dots, B_N], \quad (6.3.5)$$

где каждый блок имеет ширину  $r = n/N$ . Параметр  $N$  можно взять достаточно большим для того, чтобы сообщения или локальные буферы были достаточно малыми.

Модифицируя алгоритм 6.3.1 при упрощающем допущении  $N = qp$ , мы предположим, что Proc( $\mu$ ) хранит у себя блочные столбцы  $A_j$ ,  $B_j$  и  $C_j$ ,  $j = \mu:p:N$ . Снова  $A_i$  ходят по кругу по принципу карусели, только требуется  $q$  оборотов. На  $k$ -м обороте по кругу перемещаются блочные столбцы  $A_{i+(k-1)p}$ ,  $i = 1:p$ . Коммуникационные издержки, связанные с этой процедурой, имеют вид

$$\tilde{T}_{6.3.1} = 2qp(\alpha_d + \beta_d nr). \quad (6.3.6)$$

Для модификации алгоритма 6.3.2, чтобы требовались только три локальных буфера размера  $n \times r$ , мы поручим Proc( $\mu$ ) задачу вычисления  $D_j$  для  $j = \mu:p:N$ . Каждое вычисление  $D_j$  процессор начинает с загрузки  $B_j$  и  $C_j$ . При этом все  $A_j$  должны поступить по ходу модификации  $C_j$ . После полного перевычисления блока  $C_j$  он возвращается в общую память. Издержки на обмен с общей памятью даются формулой

$$\tilde{T}_{6.3.2} = q(3 + N)(\alpha_s + \beta_s nr). \quad (6.3.7)$$

### 6.3.2. Проблема равномерной загруженности

Предположим, что матрица  $B$  в (6.3.1) нижняя треугольная. Поскольку  $D_1$  требует немного большей работы, чем  $D_p$ , очевидна необходимость перераспределения работы для того, чтобы алгоритмы 6.3.1 и 6.3.2 давали равномерную загруженность.

Эта проблема решается круговым отображением данных. Именно, Proc( $\mu$ ) назначается на вычисление

$$D(:, \mu:p:n) = C(:, \mu:p:n) + AB(:, \mu:p:n) = \sum_{\tau=1}^p A(:, \tau:p:n)B(\tau:p:n, \mu:p:n). \quad (6.3.8)$$

### 6.3.3. Некоторые вопросы, связанные с дроблением вычислений

Алгоритмы 6.3.1 и 6.3.2, очевидно, показывают, что дробление вычислений может задаваться подходящим блочным разбиением. Чтобы проиллюстрировать некоторые дополнительные аспекты, связанные с дроблением вычислений, рассмотрим новую версию алгоритма 6.3.1, в которой эффективно реализуется вычисление

$C + AB$  вида

$$[C_1 \ C_2 \ C_3 \ C_4] + [A_1 \ A_2 \ A_3 \ A_4] \begin{bmatrix} B_{11} & 0 & 0 & B_{14} \\ B_{21} & B_{22} & 0 & 0 \\ 0 & B_{32} & B_{33} & 0 \\ 0 & 0 & B_{43} & B_{44} \end{bmatrix}.$$

Заметим, что Proc( $\mu$ ) нуждается в единственном нелокальном  $A_t$ , которое хранится у его правого соседа. Таким образом, это специальное вычисление  $C + AB$  может выполняться следующим образом:

$$\begin{aligned} &\text{send}(A_{loc}, \text{left}) \\ &\text{recv}(W, \text{right}) \\ C_{loc} = & C_{loc} + A_{loc} B_{\mu\mu} + WB_{\mu+1,\mu}. \end{aligned} \quad (6.3.9)$$

Здесь принято соглашение о том, что  $B_{p+1,p} = B_{1p}$ . Как видим, для  $W$  требуется 2-мерное рабочее пространство.

Размер рабочего пространства можно уменьшить, если вычислять  $WB_{\mu+1,\mu} = A_{\mu+1,\mu} B_{\mu+1,\mu}$  как сумму внешних произведений. В этом способе Proc( $\mu$ ) довольствуется тем, что запрашивает  $A_{\mu+1}$  от своего правого соседа по одному столбцу в каждый момент времени:

```

 $C_{loc} = C_{loc} + A_{loc} B_{loc} (1 + (\mu - 1)r : \mu r)$ 
for  $t = 1:r$ 
    send( $A_{loc}(:, t)$ , left)
    recv( $w$ , right)
    if  $\mu < p$ 
         $i = t + \mu r$ 
    else
         $i = t$ 
    end
     $\{w = A(:, i)\}$ 
     $C_{loc} = C_{loc} + w B_{loc}(i, :)$ 
end

```

Эта процедура отличается более мелким дроблением вычислений по сравнению с (6.3.9); тем самым это показывает, что дробление вычислений может быть связано с ограничениями на буфер.

#### 6.3.4. Систолическое умножение $3 \times 3$ -матриц

Теперь мы покажем, как реализуется умножение  $N \times N$ -матриц  $D = C + AB$  на систолическом  $N \times N$ -торе. (См. рис. 6.1.3.) Могут использоваться и другие топологии процессорной сети, но мы выбрали тор из педагогических соображений.

Начнем с  $N = 3$ . Предположим, что у нас имеется систолический  $3 \times 3$ -тор, который изображается по ячейкам следующим образом:

Proc(1,1)	Proc(1,2)	Proc(1,3)
Proc(2,1)	Proc(2,2)	Proc(2,3)
Proc(3,1)	Proc(3,2)	Proc(3,3)

Мы покажем, как на этой систолической сети организовать умножение  $3 \times 3$ -матриц  $D = C + AB$ . На Proc( $i, j$ ) возлагается задача получения  $d_{ij}$  на месте  $c_{ij}$ . На каждом временном шаге Proc( $i, j$ ) добавляет одно из произведений  $a_{ik}b_{kj}$  к локальной накапливаемой сумме  $c$ , которая после временных шагов будет содержать  $d_{ij}$ .

Сначала мы заострим внимание на Proc(1, 1) и вычислении величины

$$d_{11} = c_{11} + a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}.$$

Предположим, что шесть входов, определяющих это скалярное произведение, размещаются внутри тора следующим образом:

$a_{11}$	$b_{11}$	$a_{12}$	.	$a_{13}$	.
.	$b_{21}$	.	.	.	.
.	$b_{31}$	.	.	.	.

(Не обращайте внимание на точки. Позже они будут замещены различными  $a_{ij}$  и  $b_{ij}$ .)

В нашей систолической модели сейчас каждый процессор исполняет программу вида (6.1.1). Значит, мы можем «прокрутить» через Proc(1, 1) первую строку для  $A$  и первый столбец для  $B$  таким образом, что на каждом шаге по времени к текущей сумме  $c$ , аккумулирующей  $c_{11}$ , будет прибавляться новая пара  $\{a_{1k}, b_{k1}\}$ . Для этой отдельной порции всех вычислений поток данных представляется следующим образом:

$a_{12}$	$b_{21}$	$a_{13}$	.	$a_{11}$	.
.	$b_{31}$	.	.	.	.
.	$b_{11}$	.	.	.	.

$$\begin{aligned} t &= 1 \\ c &= c + a_{12}b_{21} \end{aligned}$$

$a_{13}$	$b_{31}$	$a_{11}$	.	$a_{12}$	.
.	$b_{11}$	.	.	.	.
.	$b_{21}$	.	.	.	.

$$\begin{aligned} t &= 2 \\ c &= c + a_{13}b_{31} \end{aligned}$$

$a_{11}$	$b_{11}$	$a_{12}$	.	$a_{13}$	.
.	$b_{21}$	.	.	.	.
.	$b_{31}$	.	.	.	.

$$\begin{aligned} t &= 3 \\ c &= c + a_{11}b_{11} \end{aligned}$$

Итак, после трех шагов локальная переменная  $c$  в Proc(1, 1) содержит  $d_{11}$ .

Мы организовали поток данных так, что по тору  $a_{1j}$  перемещаются в западном, а  $b_{i1}$  — в северном направлении. Таким образом, ясно, что Proc(1, 1) должен

выполнять локальную программу следующего вида:

```
for t = 1:3
    send(a, west); send(b, north)
    recv(a, east); recv(b, south)
    c = c + ab
end
```

Далее, посмотрим, что делается в Proc(1, 2), Proc(1, 3), Proc(2, 1) и в Proc(3, 1). На данном этапе изложения эти процессоры просто помогают двигать по кругу элементы  $A(1, 1:3)$  и  $B(1:3, 1)$ . Они способны на большее. Возьмем Proc(1, 2), в моменты  $t = 1:3$  он хранит у себя соответственно  $a_{13}$ ,  $a_{11}$  и  $a_{12}$ . Если бы в эти моменты времени через Proc(1, 2) проходили  $b_{32}$ ,  $b_{12}$  и  $b_{22}$ , то можно было бы сформировать  $d_{12} = c_{12} + a_{13}b_{32} + a_{11}b_{12} + a_{12}b_{22}$ . Аналогично Proc(1, 3) мог бы вычислить  $d_{13} = c_{13} + a_{11}b_{13} + a_{12}b_{23} + a_{11}b_{33}$ , если бы в моменты  $t = 1:3$  были доступны  $b_{13}$ ,  $b_{23}$  и  $b_{33}$ . Для этого мы инициализируем тор следующим образом:

$a_{11}$	$b_{11}$	$a_{12}$	$b_{22}$	$a_{13}$	$b_{33}$
.	$b_{21}$	.	$b_{32}$	.	$b_{13}$
.	$b_{31}$	.	$b_{12}$	.	$b_{23}$

Учитывая поток  $b_{ij}$  в северном направлении, получаем:

$a_{12}$	$b_{21}$	$a_{13}$	$b_{32}$	$a_{11}$	$b_{13}$
.	$b_{31}$	.	$b_{12}$	.	$b_{23}$
.	$b_{11}$	.	$b_{22}$	.	$b_{33}$

 $t = 1$ 

$a_{13}$	$b_{31}$	$a_{11}$	$b_{12}$	$a_{12}$	$b_{23}$
.	$b_{11}$	.	$b_{22}$	.	$b_{33}$
.	$b_{21}$	.	$b_{32}$	.	$b_{13}$

 $t = 2$ 

$a_{11}$	$b_{11}$	$a_{12}$	$b_{22}$	$a_{13}$	$b_{33}$
.	$b_{21}$	.	$b_{32}$	.	$b_{13}$
.	$b_{31}$	.	$b_{12}$	.	$b_{23}$

 $t = 3$ 

Итак, если  $B$  отображается на тор с «косой линией старта», мы можем сделать так, чтобы первая строка процессоров вычисляла первую строку для  $C$ .

Если мы скосили вторую и третью строки для  $A$  в такой же манере, то сможем заставить все девять процессоров выполнять умножение-сложение на каждом шаге. Именно, при расположении

$a_{11}$	$b_{11}$	$a_{12}$	$b_{22}$	$a_{13}$	$b_{33}$
$a_{22}$	$b_{21}$	$a_{23}$	$b_{32}$	$a_{21}$	$b_{13}$
$a_{33}$	$b_{31}$	$a_{31}$	$b_{12}$	$a_{32}$	$b_{23}$

поток  $a_{ij}$  на запад и поток  $b_{ij}$  на север дают нам следующее:

$a_{12}$	$b_{21}$	$a_{13}$	$b_{32}$	$a_{11}$	$b_{13}$	$t = 1$
$a_{23}$	$b_{31}$	$a_{21}$	$b_{12}$	$a_{22}$	$b_{23}$	
$a_{31}$	$b_{11}$	$a_{32}$	$b_{22}$	$a_{33}$	$b_{33}$	
$a_{13}$	$b_{31}$	$a_{11}$	$b_{12}$	$a_{12}$	$b_{23}$	$t = 2$
$a_{21}$	$b_{11}$	$a_{22}$	$b_{22}$	$a_{23}$	$b_{33}$	
$a_{32}$	$b_{21}$	$a_{33}$	$b_{32}$	$a_{31}$	$b_{13}$	
$a_{11}$	$b_{11}$	$a_{12}$	$b_{22}$	$a_{13}$	$b_{33}$	$t = 3$
$a_{22}$	$b_{21}$	$a_{23}$	$b_{32}$	$a_{21}$	$b_{13}$	
$a_{33}$	$b_{31}$	$a_{31}$	$b_{12}$	$a_{32}$	$b_{23}$	

После этого примера мы готовы к тому, чтобы сформулировать алгоритм для произвольного  $N$ .

### 6.3.5. Общий случай

Чтобы описать распределение  $N \times N$ -матриц  $A$  и  $B$  в общем случае, удобно определить целочисленную функцию  $\langle \cdot \rangle_N$  по следующему правилу:

$$\left. \begin{array}{l} a = aN + \beta \\ 0 \leq \beta < N \end{array} \right\} \Rightarrow \langle a \rangle_N = \begin{cases} \beta, & \beta \neq 0, \\ N, & \beta = 0. \end{cases}$$

**Алгоритм 6.3.3.** Предположим, что заданы  $A \in \mathbb{R}^{N \times N}$ ,  $B \in \mathbb{R}^{N \times N}$ ,  $C \in \mathbb{R}^{N \times N}$  и  $D = C + AB$ . Если в систолическом  $N \times N$ -торе каждый процессор исполняет следующий алгоритм, то по окончании процесса  $\text{Proc}(i, j)$  в локальной переменной  $c$  содержит  $d_{ij}$ .

```

loc. init [ $N, (i, j) = my.id, a = a_{i, \langle i+j-1 \rangle_N}, b = b_{\langle i+j-1 \rangle_{N,j}}$ ]
for  $t = 1 : N$ 
    send ( $a$ , west); send ( $b$ , north)
    recv ( $a$ , east); recv ( $b$ , south)
     $c = c + ab$ 
end
quit

```

Нетрудно доказать, что этот алгоритм будет работать. Сначала заметим, что в момент  $t$   $\text{Proc}(i, j)$  хранит у себя  $a_{i, \langle i+j-1+t \rangle_N}$  и  $b_{\langle i+j-1+t \rangle_{N,j}}$ . По окончании

$$c = c_{ij} + \sum_{t=1}^N a_{i, \langle i+j-1+t \rangle_N} b_{\langle i+j-1+t \rangle_{N,j}} = c_{ij} + \sum_{k=1}^N a_{ik} b_{kj} = d_{ij}.$$

Одно из приятных свойств систолической модели заключается в том, что строгая регламентация делает проверку локальной программы относительно простым делом.

### 6.3.6. Блочный аналог

Предположим, что  $A$ ,  $B$  и  $C$  суть блочные  $N \times N$ -матрицы с  $r \times r$ -блоками. Алгоритм 6.3.3 легко модифицируется так, чтобы вычислять

$$D = \begin{bmatrix} C_{11} & \dots & C_{1N} \\ \vdots & & \vdots \\ C_{N1} & \dots & C_{NN} \end{bmatrix} + \begin{bmatrix} A_{11} & \dots & A_{1N} \\ \vdots & & \vdots \\ A_{N1} & \dots & A_{NN} \end{bmatrix} \begin{bmatrix} B_{11} & \dots & B_{1N} \\ \vdots & & \vdots \\ B_{N1} & \dots & B_{NN} \end{bmatrix}.$$

Все, что нам нужно сделать,—это заменить обращения к  $a_{ij}$ ,  $b_{ij}$  и  $c_{ij}$  на обращения к  $A_{ij}$ ,  $B_{ij}$  и  $C_{ij}$ .

### 6.3.7. Асинхронная тороидальная процедура

Мы построим ниже асинхронный аналог алгоритма 6.3.3 для вычисления  $D = C + AB$ . Пусть  $A \in \mathbf{R}^{N \times N}$  и  $B \in \mathbf{R}^{N \times N}$ , и будем считать, что  $\text{Proc}(i, j)$  имеет своей задачей вычислить  $d_{ij}$ . Предположим ради любопытства, что  $\text{Proc}(i, j)$  вначале хранит у себя  $a_{ij}$ ,  $b_{ij}$  и  $c_{ij}$ . Чтобы получить сконченность данных, которая оказалась существенной для алгоритма 6.3.3, строки матрицы  $A$  будут циркулировать вдоль строк тора в западном направлении, в то время как столбцы матрицы  $B$  циркулируют вверх по столбцам тора в северном направлении.

**Алгоритм 6.3.4.** Предположим, что заданы  $A \in \mathbf{R}^{N \times N}$ ,  $B \in \mathbf{R}^{N \times N}$  и  $C \in \mathbf{R}^{N \times N}$  и что  $D = C + AB$ . Если каждый процессор в  $N \times N$ -торе исполняет следующий алгоритм, то по окончании процесса  $\text{Proc}(i, j)$  хранит у себя  $d_{ij}$  в локальной переменной  $c$ .

```

loc. init [ $N, (\mu, \lambda) = my.id, a = a_{\mu,\lambda}, b = b_{\mu,\lambda}, c = c_{\mu,\lambda}$ ]
for  $k = 1 : \mu - 1$ 
    send ( $a$ , west); recv ( $a$ , east)
end
for  $k = 1 : \lambda - 1$ 
    send ( $b$ , north); recv ( $b$ , south)
end

```

```

for  $k = 1 : n$ 
   $c = c + ab$ 
  send( $a$ , west); recv( $a$ , east)
  send( $b$ , north); recv( $b$ , south)
end
for  $k = 1 : \mu - 1$ 
  send( $a$ , east); recv( $a$ , west)
end
for  $k = 1 : \lambda - 1$ 
  send( $b$ , south); recv( $b$ , north)
end
quit

```

### 6.3.8. Использование блочного скалярного произведения в режиме резерва задач

Теперь разберемся с вычислением  $n \times n$ -матрицы  $D = C + AB$  на общей памяти, когда за основу берутся блочные скалярные произведения. Будем считать, что  $n = Nr$  и  $A = (A_{ij})$ ,  $B = (B_{ij})$ ,  $C = (C_{ij})$  и  $D = (D_{ij})$  суть блочные  $N \times N$ -матрицы с  $r \times r$ -блоками. Итак,

$$D_{ij} = C_{ij} + \sum_{k=1}^N A_{ik} B_{kj}.$$

Вычисление  $D_{ij}$  мы рассматриваем как задачу с меткой  $(i, j)$ . Будем строить параллельный алгоритм, в основе которого режим резерва задач. Чтобы задача не казалась слишком легкой, будем считать, что  $D$  нужно получить на месте  $B$ .

Мы организуем вычисления таким образом, чтобы  $D$  вычислялась блок за блоком в порядке возрастания столбцов. После вычисления  $j$ -го блочного столбца матрицы  $D$  можно без опасений изменить содержимое  $j$ -го блочного столбца в  $B$ . Таким образом, мы хотим иметь процедуру следующего вида:

Повторять до тех пор, пока не останется задач:

Получить следующую задачу, назвать ее задачей  $(i, j)$ .

Вычислить  $D_{ij}$ .

Если это безопасно, на месте  $B_{ij}$  записать  $D_{ij}$ .

**end**

Как и в наших гахру-процедурах с резервом задач, нам нужен монитор, на который возлагается назначение задач. Теперь ситуация двумерная – в том смысле, что задачи с наибольшим удобством помечаются парой индексов, указывающих на подлежащее вычислению  $D_{ij}$ :

$$(1, 1), \dots, (N, 1), (1, 2), \dots, (N, 2), \dots, (1, N), \dots, (N, N).$$

Поэтому по аналогии с **nexti** мы получаем

```

monitor: nextij
  protected variables: flag, row, col, idle.proc
  condition variables: waiti
  initializations: flag = 0
  procedure nextij.index(N, p, more, i, j)
    if flag = 0
      row = 0; col = 1; idle.proc = 0; flag = 1
    end
    if row ≤ N – 1
      row = row + 1
    end
  end

```

```

else
    row = 1; col = col + 1
end
if col ≤ N
    i = row; j = col; more = true
else
    more = false; idle.proc = idle.proc + 1
    if idle.proc ≤ p - 1
        delay(waiti)
    else
        flag = 0
    end
    continue(waiti)
end
end nextij

```

Безопасное замещение данных может управляться другим монитором с процедурой **prod.canwrite**. Именно, если **prod.canwrite**( $W, B, i, j, r, N$ ) безопасно замещает блоки  $B_{ij}$   $r \times r$ -матрицей  $W$ , то мы получаем

**Алгоритм 6.3.5.** Предположим, что  $n = Nr$  и  $A = (A_{ij})$ ,  $B = (B_{ij})$  и  $C = (C_{ij})$  суть блочные  $N \times N$ -матрицы с  $r \times r$ -блоками. Если эти матрицы хранятся в общей памяти по блокам и каждый процессор исполняет следующий алгоритм, то по окончании процесса на месте  $B_{ij}$  получается  $C_{ij} + \sum_{k=1}^N A_{ik} B_{kj}$ ,  $i = 1:N$ ,  $j = 1:N$ . Предполагается, что  $p \geq N$ .

```

call nextij.index(N, p, more, i, j)
while more = true
    get(Cij, Cloc)
    for k = 1:N
        get(Aik, Aloc); get(Bkj, Bloc); Cloc = Cloc + Aloc Bloc
    end
    call prod.canwrite(Cloc, i, j, N)
    call nextij.index(N, p, more, i, j)
end

```

Монитор **prod**, обеспечивающий безопасное замещение, устроен следующим образом:

```

monitor : prod
    protected variables : flag, col.done(·)
    condition variables : waitq(·)
    initializations : flag = 0
    procedure prod.canwrite(W, i, j, N)
        if flag = 0
            flag = 1; col.done(1:N) = 0
        end
        col.done(j) = col.done(j) + 1
        if col.done(j) < N
            delay(waitq(j))
        end
        put(W, Bij)
        continue(waitq(j))
    end conwrite
end prod

```

Защищенный вектор *col.done* используется для подсчета числа блоков, уже извлеченных из каждого блочного столбца матрицы  $D$ . Когда процессор вызывает *prod.canwrite* с помещенным в  $C_{loc}$  блоком  $D_{ij}$ , переменная *col.done(j)* увеличивается на единицу. Если после этого ее значение меньше  $N$ , то замещать  $B_{ij}$  небезопасно, так как этот блок может еще потребоваться другой задаче. В этом случае вызывающий процессор ставится в очередь, управляемую переменной условия *waitq(j)*. После того как в  $j$ -м блочном столбце получен  $N$ -й блок, он хранится в соответствующем  $B_{ij}$ , а процессоры, в соответствии с *waitq(j)*, освобождаются для того, чтобы возобновить запись своих  $D_{ij}$ .

Предположение о том, что  $A$ ,  $B$  и  $C$  хранятся по столбцам (см. § 1.3), для  $A_{ij}$  и  $B_{ij}$  влечет за собой возможность считывания их с помощью одного оператора *get* и записи с помощью одного оператора *put*.

Если в алгоритме 6.3.5 опустить предположение  $p \geq N$ , то процессоры могут все оказаться ожидающими из-за *waitq(1)*, поскольку возможно, что  $N$ -й блок в первом блочном столбце никогда не будет вычислен.

В предположении, что нет простаивания и каждый процессор обрабатывает примерно  $N^2/p$  задач, на обмены с общей памятью каждый процессор тратит

$$T_{6.3.5} = \frac{N^2}{p} (2N + 2)(\alpha_s + \beta_s r^2) \quad (6.3.9)$$

секунд.

Этот подход к задаче  $D = C + AB$  связан с меньшим числом обменов с памятью, чем блочный saxpy-алгоритм на базе (6.3.5). Обоснование имеет много общего с нашим обсуждением переиспользования данных в § 1.4.9. Это обсуждение в нашем случае относится к локальной памяти и общей памяти, которые соответствуют кеш-памяти и основной памяти. Конечно, локальные запоминающие устройства в системе с общей памятью могут иметь свою собственную иерархию.

### Задачи

- 6.3.1. Расширить алгоритмы 6.3.1 и 6.3.2 таким образом, чтобы  $C$  заменялось на  $C + ABA^T$ .
- 6.3.2. Разработать детальные версии алгоритмов 6.3.1 и 6.3.2, основанные на (6.3.5).
- 6.3.3. Дать детальный, использующий структурные особенности алгоритм для вычисления  $C_{loc} + A_{loc}B_{loc}(\tau:p:n,:)$  в алгоритме 6.3.3.
- 6.3.4. В режиме резерва задач разработать алгоритм, в котором на месте  $A$  получается  $D = C + AB$ , где  $B$  – нижняя треугольная. В качестве  $j$ -й задачи взять вычисление  $D(:, j)$ .
- 6.3.5. Модифицировать алгоритм 6.3.4 так, чтобы на месте  $C$  получалось  $D = C + A^k B$ .
- 6.3.6. Разработать версию алгоритма 6.3.1, дающую равномерную загруженность при обработке нижней треугольной  $B$ .
- 6.3.7. Приспособить алгоритм 6.3.1 для обработки трехдиагональной  $B$ . Пусть  $e(1:n)$ ,  $d(1:n)$  и  $f(1:n)$  обозначают поддиагональную, диагональную и наддиагональную части, причем  $e(n) = f(1) = 0$ . Считайте, что Proc( $\mu$ ) хранит у себя  $e(col)$ ,  $d(col)$  и  $f(col)$ , где  $col = 1 + (\mu - 1)r : \mu r$ .
- 6.3.8. На месте верхней треугольной матрицы можно получить ее квадрат без какого-либо дополнительного рабочего пространства. Напишите для этого процедуру, использующую динамическое планирование.

### Замечания и литература к § 6.3

Различные аспекты параллельных матричных вычислений обсуждаются в

Cheng K. H. and Sahni S. (1987). “VLSI Systems for Band Matrix Multiplication”, Parallel Computing 4, 239–258.

- Elster A. and Reeves A. P. (1988). "Block Matrix Operations Using Orthogonal Trees", in G. Fox (ed.), The Third Conference on Hypercube Concurrent Computers and Applications, Vol. II – Applications, ACM Press, New York, pp. 1554–1561.
- Fox G., Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. (1988). On Concurrent Processors Vol I: General Techniques and Regular Problems. Prentice-Hall, Englewood Cliffs, NJ.
- Fox G., Otto S. W. and Hey A. J. (1987). "Matrix Algorithms on a Hypercube I: Matrix Multiplication", Parallel Computing 4, 17–31.
- Johnsson S. L. and Ho C. T. (1987). "Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on a Boolean Cube", Report YALEU/DCS/RR-569, Dept. of Computer Science, Yale University, New Haven, CT.

## 6.4. Кольцевые процедуры разложения

В этом параграфе мы обсудим, как можно организовать различные матричные разложения на кольце. Представив параллельную реализацию основанного на гахру Холецкого, мы покажем затем, как работает та же самая логика планирования для разложений  $PA = LU$  и  $A = QR$ . В конце обсуждается параллельное решение треугольных систем.

### 6.4.1. Кольцевой алгоритм Холецкого: случай $n = p$

Посмотрим, как процедура разложения Холецкого может распределяться по кольцу из  $p$  процессоров  $\text{Proc}(1), \dots, \text{Proc}(p)$ . Отправной точкой является уравнение

$$G(j,j)G(j:n,j) = A(j:n,j) - \sum_{k=1}^{j-1} G(j,k)G(j:n,k) \equiv v(j:n).$$

Это уравнение получается приравниванием  $j$ -х столбцов в  $n \times n$ -уравнении  $A = GG^T$ . После того как найден вектор  $v(j:n)$ ,  $G(j:n,j)$  получается с помощью простого нормирования:  $G(j:n,j) = v(j:n)/\sqrt{v(j)}$ . Для ясности предположим сначала, что  $n = p$  и  $\text{Proc}(\mu)$  в начальный момент хранит у себя  $A(\mu:n,\mu)$ . По окончании процесса каждый процессор замещает свой  $A$ -столбец на соответствующий  $G$ -столбец. Для  $\text{Proc}(\mu)$  этот процесс включает  $\mu - 1$  шаги-модификаций вида  $A(\mu:n,\mu) \leftarrow A(\mu:n,\mu) - G(\mu,j)G(\mu:n,j)$ , вслед за которыми выполняются извлечение квадратного корня и нормирование. Поэтому общая структура локальной программы для  $\text{Proc}(\mu)$  имеет следующий вид:

```

for j = 1 : μ - 1
    Получить G-столбец от соседа слева.
    Послать копию полученного G-столбца соседу справа.
    Модифицировать A(μ:n,μ).
end
Сформулировать G(μ:n,μ) и послать соседу справа.

```

Таким образом,  $\text{Proc}(1)$  сразу же вычисляет  $G(1:n, 1) = A(1:n, 1)/\sqrt{A(1,1)}$  и отсылает его  $\text{Proc}(2)$ . Как только  $\text{Proc}(2)$  получит этот столбец, он может сформировать  $G(2:n, 2)$  и передать его для  $\text{Proc}(2)$  и т. д. Устроив такой конвейер, мы можем утверждать, что процессор может быть свободным, как только он вычислит свой  $G$ -столбец. Отсюда также вытекает, что каждый процессор получает  $G$ -столбцы в порядке возрастания, т. е.  $G(:, 1)$ ,  $G(:, 2)$  и т. д. Основываясь на этих наблюдениях, мы получаем

**Алгоритм 6.4.1.** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и положительно определенная и  $A = GG^T$  – ее разложение Холецкого. Если в  $n$ -процессорном

кольце каждый процессор исполняет следующую программу, то по окончании процесса  $\text{Proc}(\mu)$  содержит  $G(\mu:n, \mu)$  в локальном массиве  $A_{loc}(\mu:n)$ .

```

loc.init [ $n, \mu = my.id, left, right, A_{loc}(\mu:n) = A(\mu:n, \mu)$ ]
 $j = 1$ 
while  $j < \mu$ 
    recv ( $g(j:n), left$ )
    if  $\mu < n$ 
        send ( $g(j:n), right$ )
    end
     $A_{loc}(\mu:n) = A_{loc}(\mu:n) - g(\mu)g(\mu:n)$ 
     $j = j + 1$ 
end
 $A_{loc}(\mu:n) = A_{loc}(\mu:n)/\sqrt{A_{loc}(\mu)}$ 
if  $\mu < n$ 
    send ( $A_{loc}(\mu:n), right$ )
end
quit

```

Заметим, что число полученных  $G$ -столбцов должно быть  $j - 1$ . Если  $j = \mu$ , то наступает время, когда  $\text{Proc}(\mu)$  формирует и отсылает  $G(\mu:n, \mu)$ .

#### 6.4.2. Кольцевой метод Холецкого: общий случай

Теперь обобщим алгоритм 6.4.1 на случай произвольного  $n$ . Для распределения вычислений есть два очевидных способа. Можно потребовать, чтобы каждый процессор вычислял множество идущих подряд  $G$ -столбцов. Например, если  $n = 11$ ,  $p = 3$  и  $A = [a_1, \dots, a_{11}]$ , то мы могли бы распределить  $A$  следующим образом:

$$\left[ \underbrace{a_1 a_2 a_3 a_4}_{\text{Proc (1)}} \mid \underbrace{a_5 a_6 a_7 a_8}_{\text{Proc (2)}} \mid \underbrace{a_9 a_{10} a_{11}}_{\text{Proc (3)}} \right].$$

Тогда каждый процессор мог бы заниматься отысканием соответствующих столбцов матрицы  $G$ . Неприятность, связанная с этим подходом, состоит, например, в том, что  $\text{Proc}(1)$  приступает после того, как будет найден четвертый столбец для  $G$ , хотя еще остается много работы.

Более равномерная загруженность получается, если вычислительные задачи распределить с использованием кругового отображения, т.е.

$$\left[ \underbrace{a_1 a_4 a_7 a_{10}}_{\text{Proc (1)}} \mid \underbrace{a_2 a_5 a_8 a_{11}}_{\text{Proc (2)}} \mid \underbrace{a_3 a_6 a_9}_{\text{Proc (3)}} \right].$$

В этой схеме  $\text{Proc}(\mu)$  получает  $G(:, \mu:p:n)$ . Когда любой заданный процессор заканчивает вычисление своих  $G$ -столбцов, у каждого из остальных процессоров будет по крайней мере один столбец из  $G$ , который нужно вычислить. Итак, если  $n/p \gg 1$ , то все процессоры загружены большую часть времени.

Давайте разберемся с деталями процедуры Холецкого с круговым распределением. Каждый процессор поддерживает пару счетчиков. Счетчик  $j$  – это индекс следующего  $G$ -столбца, который поступит в  $\text{Proc}(\mu)$ . Процессору также нужно знать индекс следующего  $G$ -столбца, который он должен найти. Заметим, что если  $col = \mu:p:n$ , то  $\text{Proc}(\mu)$  отвечает за  $G(:, col)$ , а  $L = \text{length}(col)$  – это число  $G$ -столбцов,

которые он должен вычислить. Будем использовать  $q$  для того, чтобы обозначить состояние процесса получения  $G$ -столбцов. В любой момент  $\text{col}(q)$  – это индекс следующего подлежащего вычислению  $G$ -столбца.

**Алгоритм 6.4.2.** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и положительно определенная и  $A = GG^T$  – ее разложение Холецкого. Если в  $p$ -процессорном кольце каждый процессор исполняет следующую программу, то по окончании процесса  $\text{Proc}(\mu)$  хранит у себя  $G(k:n, k)$  для  $k = \mu:p:n$  в локальном массиве  $A_{loc}(1:n, L)$ , где  $L = \text{length}(\text{col})$  и  $\text{col} = \mu:p:n$ . В частности,  $G(\text{col}(q):n, \text{col}(q))$  содержится в  $A_{loc}(\text{col}(q):n, q)$  при  $q = 1:L$ .

```

loc.init [p, μ = pid(·), left, right, n, Aloc = A(1:n, μ:p:n)]
j = 1; q = 1; col = μ:p:n; L = length(col)
while q ≤ L
    if j = col(q)
        {Сформировать G(j:n, j).}
        Aloc(j:n, q) = Aloc(j:n, q)/√Aloc(j, q)
        if j < n
            send(Aloc(j:n, q), right)
        end
        j = j + 1
        {Модифицировать локальные столбцы.}
        for k = q + 1:L
            r = col(k)
            Aloc(r:n, k) = Aloc(r:n, k) - Aloc(r, q) Aloc(r:n, q)
        end
        q = q + 1
    else
        recv(g(j:n), left)
        Вычислить α, для которого Proc(α) – поставщик полученного G-столбца.
        Вычислить β – индекс последнего столбца для Proc(right).
        if right ≠ α ∧ j < β
            send(g(j:n), right)
        end
        {Модифицировать локальные столбцы.}
        for k = q:L
            r = col(k); Aloc(r:n, k) = Aloc(r:n, k) - g(r) g(r:n)
        end
        j = j + 1
    end
end
quit

```

Чтобы проиллюстрировать логику системы указателей, мы рассмотрим выборочную 3-процессорную ситуацию с  $n = 10$ . Допустим, что три локальных значения  $q$  равны 3, 2 и 2 и три соответствующих значения  $\text{col}(q)$  суть 7, 5 и 6:

$$\left[ \underbrace{a_1 a_4}_{\text{Proc } (1)} \overset{\downarrow}{a_7} a_{10} \quad | \quad \underbrace{a_2 a_5}_{\text{Proc } (2)} \overset{\downarrow}{a_8} a_{11} \quad | \quad \underbrace{a_3 a_6}_{\text{Proc } (3)} \overset{\downarrow}{a_9} \right]$$

Сейчас  $\text{Proc}(2)$  вырабатывает пятый  $G$ -столбец и увеличивает свое  $q$  до 3.

Нуждается в объяснении логика, связанная с решением передать полученный

$G$ -столбец соседу справа. Должны выполняться два условия:

- Сосед справа не должен быть процессором, породившим  $G$ -столбец. В этом случае циркуляция полученного  $G$ -столбца должным образом заканчивается.
- У соседа справа должно быть еще больше  $G$ -столбцов для вычисления. Иначе  $G$ -столбец будет отправлен какому-нибудь неактивному процессору.

Соображения такого рода весьма типичны для матричных вычислений на распределенной памяти.

Проследим за поведением алгоритма 6.4.2 в предположении, что  $n \gg p$ . Нетрудно показать, что  $\text{Prcs}(\mu)$  выполняет

$$f(\mu) = \sum_{k=1}^r 2(n - (\mu + (k-1)p))(\mu + (k-1)p) \approx n^3/p$$

флопов. Заметим, что  $f(1)/f(p) = 1 + O(p^2/n^2)$ , и, значит, с точки зрения флопов мы имеем процедуру с приблизительно равномерной загруженностью.

Каждый процессор получает и отсылает почти что каждый  $G$ -столбец. Используя нашу модель (6.1.4) для расходов на коммуникацию, мы видим, что каждый процессор тратит на обмены время

$$\sum_{j=1}^n 2(a_d + \beta_d(n-j)) \approx 2a_d n + \beta_d n^2.$$

Если предположить, что вычисления идут со скоростью  $R$  флопов в секунду, то отношение вычисления/обмены для алгоритма 6.4.2 приблизительно равно  $(n/p)(2/3 R \beta_d)$ . Таким образом, значения коммуникационных издержек снижаются с ростом  $n/p$ .

#### 6.4.3. Кольцевые процедуры для других разложений

Алгоритм 6.4.2 легко модифицируется для вычисления других разложений, таких, как  $PA = LU$  и  $A = QR$ . Это вызвано тем, что в случае  $n \times n$ -матриц алгоритмы для этих разложений можно разбить на следующие этапы:

```
for j = 1:n-1
    Вычислить преобразование  $T_j$  по текущему  $A(:, j)$ .
    Модифицировать  $A(j:n, j:n)$ .
end
```

Типичные  $T_j$  включают преобразования Гаусса и Хаусхолдера, а типичные модификации включают операцию захвата.

Чтобы подчеркнуть то общее, что есть у различных процедур разложения, мы представим «шаблон» процедуры, основанный на алгоритме 6.4.2:

```
loc.init [p, mu = my.id, left, right, n, A_loc = A(1:n, mu:p:n)]
j = 1; q = 1; col = mu:p:n; L = length(col)
while q <= L
    if j = col(q)
        Вычислить преобразование  $T_j$ .
        if j < n
            Послать  $T_j$  ближайшему соседу.
        end
        Применить  $T_j$  к локальным столбцам.
        Сохранить  $T_j$ .
    end
end
```

(6.4.1)

```

 $j = j + 1$ 
 $q = q + 1$ 
else
    Получить  $T_j$  от соседа слева.
    Послать  $T_j$  соседу справа (если необходимо).
    Применить  $T_j$  к локальным столбцам.
 $j = j + 1$ 
end
end
quit

```

Важно подчеркнуть, что на практике все время работы идет с подходящим представлением для  $T_j$ . Например, в разложении Холецкого  $T_j$  закодировано в векторе  $G(j:n, j)$ . В LU оно задается  $j$ -м вектором Гаусса. В QR оно задается  $j$ -м вектором Холецкого и т. д. По кольцу передаются именно эти векторные представления преобразований, а не сами они в явном виде.

#### 6.4.4. Параллельное решение треугольных систем

Обычно после получения разложения матрицы оно используется для того, чтобы решать линейные системы. При решении треугольных систем возникают некоторые интересные моменты, о которых было рассказано в § 3.1. Но они никогда не были «злой дня», потому что стоимость разложений –  $O(n^3)$ , а стоимость обратной подстановки или прямого исключения –  $O(n^2)$ . Однако для многопроцессорной системы с распределенной памятью не очевидно, что в стоимости параллельного решения системы  $Ax = b$  будет доминировать параллельная факторизация. По сравнению, скажем, с разложением Холецкого при распараллеливании прямой или обратной подстановки «простора» намного меньше. В этом разделе мы должны разобраться с особенностями одного специального подхода к решению треугольных систем на кольце, отличающегося высокой эффективностью. Используя его вместе с рассмотренными выше параллельными схемами факторизации, мы можем сохранить старую точку зрения на решение треугольных систем, как на дешевый «постскрипту» к вычислению разложения.

Для простоты мы сосредоточимся на решении нижней треугольной  $n \times n$ -задачи  $Lx = b$ . Будем считать, что у нас есть  $p$ -процессорное кольцо и  $n$  – целое кратное  $p$ ,  $n = pr$ . Предположим также, что для  $L$  и  $b$  используется круговое отображение, т. е. Proc( $\mu$ ) хранит у себя  $L(:, \mu:p:n)$  и  $b(\mu:p:n)$ . Как действовать без этих предположений – эта тема развивается в упражнениях и в работе Li, Coleman (1988), на которой основано наше обсуждение.

Исходным пунктом будет запись основанной на saxpy прямой подстановки в следующем виде:

```

 $j = 0; z(1:n) = 0$ 
while  $j < n$ 
     $j = j + 1; b(j) = b(j) - z(j)$ 
     $z(j+1:n) = z(j+1:n) + L(j+1:n, j)b(j)$ 
end

```

(6.4.2)

В искомой кольцевой процедуре на Proc( $\mu$ ) возлагается получение  $x(\mu:p:n)$  на месте  $b(\mu:p:n)$ . Чтобы это сделать, мы прокрутим вектор  $z$  из (6.4.2) по кругу  $r$  раз, начиная с Proc(1). Этим вращениям вектора  $z$  присвоим номера от 0 до  $r - 1$ . Когда вектор  $z$  «посещает» Proc( $\mu$ ) во время  $k$ -го оборота, он используется, чтобы

вычислить  $x(\mu + kp)$ . Формально получаем следующее:

```

loc.init [p, μ = my.id, left, right, Lloc = L(1:n, μ:p:n),
          bloc = b(μ:p:n)]
r = n/p; col = μ:p:n; k = 0; z(1:n) = 0
while k < r - 1
    {Proc(1) не получает ничего в начале оборота 0.}
    if k ≠ 0 ∨ μ > 1
        recv(z, left)
    end
    k = k + 1; j = col(k)
    {Вычислить  $x_j$  и, если необходимо, модифицировать  $z$  и отослать направо.}
    bloc(k) = bloc(k) - z(j)
    if k ≠ r - 1 ∨ μ < p
        z(j + 1:n) = z(j + 1:n) + Lloc(j + 1:n, k) bloc(k)
        send(z, right)
    end
end
quit

```

Несмотря на распределенность вычислений в этом алгоритме, он обладает тем свойством, что в любой заданный момент работает только один процессор. Чтобы подступиться к распараллеливанию, заметим, что вычисление следующей переменной процессором справа может начинаться до того, как будет полностью закончено первоначальное вычисление вектора  $z$ . Например, Proc(2) может приступить к работе над  $x(2)$  сразу же по получении  $x(1)$  от Proc(1). Таким образом, Proc(1) должен был бы передать  $x(1)$  и после этого модифицировать  $z(2:n)$ . Однако для Proc(3), который будет вычислять  $x(3) = b(3) - L(3, 1)x(1) - L(3, 2)x(2)$ , нужно, чтобы Proc(1) выполнил также модификацию компоненты  $z(3)$ . В общем случае, после того как Proc( $\mu$ ) вычислит  $x(j)$ , для процессоров  $\mu + 1, \dots, p, 1, \dots, \mu - 1$  должны быть доступны величины  $L(j + 1:j + p - 1)x(j)$ .

Это подводит нас к мысли, что по кольцу должно бы прокручиваться нечто меньшее, чем весь вектор  $z$ . Давайте обозначим это «нечто» через  $w$  и попытаемся выяснить, что же это должно быть. Предположим, что когда  $w$  попадает в Proc( $\mu$ ) на  $k$ -м обороте, вычисляется  $x(\mu + kp)$ . Предположим также, что после  $k$ -го оборота Proc( $\lambda$ ) хранит у себя вектор  $z_{\lambda}^{(k)} = L(:, \lambda:p:\lambda + kp)x(\lambda:p:\lambda + kp)$ . Теперь посмотрим более внимательно, в чем нуждается Proc( $\mu$ ) для того, чтобы вычислить  $x(\mu + kp)$  по  $z_{\lambda}^{(k)}$ . Для  $j = \mu + kp$  получаем

$$x(j) = b(j) - \sum_{i=1}^{j-1} L(j, i)x(i) = b(j) - \sum_{i=1}^{\min(p, j-1)} L(j, i:p:j-1)x(i:p:j-1).$$

В тот момент, когда в Proc( $\mu$ ) должно выполняться это вычисление,

$$L(j, i:p:j-1)x(i:p:j-1) = \begin{cases} z_i^{(k)}(j), & i = 1:\mu-1, \\ z_i^{(\mu-1)}(j), & i = \mu:p, \end{cases}$$

и поэтому

$$x(j) = b(j) - \sum_{i=1}^{\mu-1} z_i^{(k)}(j) - z_{\mu}^{(\mu-1)}(j) - \sum_{i=\mu+1}^p z_i^{(\mu-1)}(j).$$

Это подсказывает нам, что в качестве циркулирующего вектора  $w$  нужно взять  $p$ -вектор, обладающий тем свойством, что по прибытии в Proc( $\mu$ ) на  $k$ -м обороте

$$w(\mu) = \sum_{i=1}^{\mu-1} z_i^{(k)}(j) + \sum_{i=\mu+1}^p z_i^{(k-1)}(j).$$

Заметим, что значения, формирующие эту сумму, могут быть «подобранны» на тех остановках (их  $p - 1$ ), которые циркулирующий вектор  $w$  делает перед прибытием в  $\text{Proc}(\mu)$ . Вот как можно реализовать эти идеи:

**Алгоритм 6.4.3.** Предположим, что матрица  $L \in \mathbb{R}^{n \times n}$  нижняя унитреугольная и  $x \in \mathbb{R}^n$  есть решение системы  $Lx = b$ , где  $b \in \mathbb{R}^n$ . Если  $n = pr$  и в  $p$ -процессорном кольце каждый процессор исполняет следующую программу, то по окончании процесса  $\text{Proc}(\mu)$  хранит у себя  $x(\mu:p:n)$  в  $b_{loc}(1:r)$ .

```

loc.init [p, μ = my.id, left, right, Lloc = L(1:n, μ:p:n),
          bloc = b(μ:p:n)]
r = n/p; col = μ:p:n; k = 0; z(1:n) = 0; w(1:p) = 0
while k < r - 1
    if k ≠ 0 ∨ μ > 1
        recv(w, left)
    end
    k = k + 1; j = col(k)
    bloc(k) = bloc(k) - z(j) - w(μ)
    if k ≠ r - 1 ∨ μ < p
        {Модифицировать ту часть z, которая нужна прямо сейчас.}
        q = min(n, j + p - 1)
        z(j + 1:q) = z(j + 1:q) + Lloc(j + 1:q, k) bloc(k)
        {Модифицировать циркулирующий вектор w.}
        if k < r - 1
            w(1:μ - 1) = w(1:μ - 1) + z(j + p - μ + 1:j + p - 1)
            w(μ) = 0
        end
        w(μ + 1:p) = w(μ + 1:p) + z(j + 1:j + p - μ)
        send(w, right)
        {Модифицировать остальную часть z.}
        if q < n
            z(q + 1:n) = z(q + 1:n) + Lloc(q + 1:n, k) bloc(k)
        end
    end
end
quit

```

За доказательством того, что в данном алгоритме достигается ускорение, близкое к линейному, мы отсылаем читателя к Li, Coleman (1988).

### Задачи

**6.4.1.** Заполнить шаблон (6.4.1) таким образом, чтобы он вычислял QR-разложение. По окончании процесса  $\text{Proc}(\mu)$  должен хранить у себя как векторы Хаусхолдера, так и полученные им  $R$ -столбцы.

**6.4.2.** Заполнить шаблон (6.4.1) таким образом, чтобы он вычислял разложение  $PA = LU$ . По окончании процесса  $\text{Proc}(\mu)$  должен хранить у себя в удобной форме информацию о  $P$  и полученные им части матриц  $L$  и  $U$ .

**6.4.3.** В алгоритме 6.4.2 сообщения ( $G$ -векторы) можно сделать более короткими. Объясните, почему?

**6.4.4.** Модифицировать алгоритм 6.4.3 так, чтобы он мог обрабатывать линейные треугольные системы произвольного порядка.

**6.4.5.** Модифицировать алгоритм 6.4.3 так, чтобы он решал верхнюю унитреугольную систему  $Ux = b$  порядка  $n = pr$ .

**6.4.6.** Модифицировать алгоритм 6.4.3 так, чтобы он решал систему  $L^T x = b$ .

#### Замечания и литература к § 6.4

Общие соображения относительно процедур факторизации на распределенной памяти можно найти в

Geist G. A. and Heath M. T. (1986). "Matrix Factorization on a Hypercube", in M. T. Heath (ed.) (1986). Proceedings of First SIAM Conference on Hypercube Multiprocessors, SIAM Publications, Philadelphia, Pa.

Ipsen I. C. F., Saad Y. and Schultz M. (1986). "Dense Linear Systems on a Ring of Processors", Lin. Alg. and Its Applic. 77, 205–239.

O'Leary D. P. and Stewart G. W. (1986). "Assignment and Scheduling in Parallel Matrix Factorization", Lin. Alg. and Its Applic. 77, 275–300.

Работы, посвященные специальному LU, Холецкому и QR, включают

Bischof C. H. (1988). "QR Factorization Algorithms for Coarse Grain Distributed Systems", PhD Thesis, Dept. of Computer Science, Cornell University, Ithaca, NY.

Davis G. J. (1986). "Column LU Pivoting on a Hypercube Multiprocessor", SIAM J. Alg. and Disc. Methods 7, 538–550.

Delosme J. M. and Ipsen I. C. F. (1986). "Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations", Lin. Alg. and Its Applic. 77, 75–112.

Geist G. A. and Heath M. T. (1985). "Parallel Cholesky Factorization on a Hypercube Multiprocessor", Report ORNL 6190, Oak Ridge Laboratory, Oak Ridge, TN.

Geist G. A. and Romine C. H. (1988). "LU Factorization Algorithms on Distributed Memory Multiprocessor Architectures", SIAM J. Sci. and Stat. Comp. 9, 639–649.

Kapur R. N. and Browne J. C. (1984). "Techniques for Solving Block Tridiagonal Systems on Reconfigurable Array Computers", SIAM J. Sci. and Stat. Comp. 5, 701–719.

Ortega J. M. and Romine C. H. (1988). "The  $ijk$  Forms of factorization Methods II: Parallel Systems", Parallel Computing 7, 149–162.

Pothen A., Jha S. and Vemapulati U. (1987). "Orthogonal Factorization on a Distributed Memory Multiprocessor", In Hypercube Multiprocessors, ed. M. T. Heath, SIAM Press, 1987.

Walker D. W., Aldcroft T., Cisneros A., Fox G., and Furmanski W. (1988). "LU Decomposition of Banded Matrices and the Solution of Linear Systems on Hypercubes", in G. Fox (ed.) (1988). The Third Conference on Hypercube Concurrent Computers and Applications, Vol. II – Applications, ACM Press, New York, pp. 1635–1655.

Параллельные решатели треугольных систем обсуждаются в

Romine C. H. and Ortega J. M. (1988). "Parallel Solution on Triangular Systems of Equations", Parallel Computing 6, 109–114.

Heath M. T. and Romine C. H. (1988). "Parallel Solution of Tridiagonal Systems on Distributed Memory Multiprocessors", SIAM J. Sci. and Stat. Comp. 9, 558–588.

Li G. and Coleman T. (1988). "A Parallel Triangular Solver for a Distributed-Memory Multiprocessor", SIAM J. Sci. and Stat. Comp. 9, 485–502.

Eisenstat S. C., Heath M. T., Henkel C. S., and Romine C. H. (1988). "Modified Cyclic Algorithms for Solving Triangular Systems on Distributed Memory Multiprocessors", SIAM J. Sci. and Stat. Comp. 9, 589–600.

Evans D. J. and Dunbar R. (1983). "The Parallel Solution of Triangular Systems of Equations", IEEE Trans. Comp. C-32, 201–204.

Montoye R. and Laurie D. (1982). "A Practical Algorithm for the Solution of Triangular Systems on a Parallel Processing System", IEEE Trans. Comp. C-31, 1076–1082.

Johnsson S. L. (1984). "Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution of Tridiagonal Systems of Equations", Report YALEU/CSD/RR-339, Dept. of Comp. Sci., Yale University, New Haven, CT.

## 6.5. Сеточные процедуры разложения

В этом параграфе обсуждаются различные алгоритмы для разложений Холецкого и QR. Для разложения Холецкого приводятся синхронные и асинхронные процедуры, позволяющие нам сравнить особенности рассуждений, связанных с этими двумя способами вычислений на распределенных системах. Параграф заканчивается кратким обсуждением систолической сеточной процедуры для QR-разложения.

### 6.5.1. Сеточная систолическая процедура Холецкого

Напомним, что версия разложения Холецкого с использованием внешних произведений связана с повторяющимся применением следующей редукции:

$$A = \begin{bmatrix} a & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \beta & 0 \\ w & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C \end{bmatrix} \begin{bmatrix} \beta & w^T \\ 0 & I_{n-1} \end{bmatrix}.$$

Здесь  $\beta = \sqrt{a}$ ,  $w = v/\beta$  и  $C = B - ww^T$ . Для записи  $k_{ji}$  алгоритма<sup>1)</sup> Холецкого с внешними произведениями воспользуемся верхними индексами, чтобы проследить этапы преобразований, которым подвергается матрица  $A$ :

```

 $A^{(0)} = A$ 
for  $k = 1:n$ 
     $a_{ij}^{(k)} = a_{ij}^{(k-1)}$  for all  $(i,j)$  with  $i < k$  or  $j > k$ 
     $a_{kk}^{(k)} = \sqrt{a_{kk}^{(k-1)}}$ 
    for  $i = k+1:n$ 
         $a_{ik}^{(k)} = a_{ik}^{(k-1)} / a_{kk}^{(k)}$ ;  $a_{ki}^{(k)} = a_{ki}^{(k-1)} / a_{kk}^{(k)}$ 
    end
    for  $j = k+1:n$ 
        for  $i = k+1:n$ 
             $a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k)} a_{kj}^{(k)}$ 
        end
    end
end

```

Для того чтобы вывести систолическую версию этой процедуры, предположим для ясности, что у нас есть систолическая  $n \times n$ -сетка и  $A \in \mathbb{R}^{n \times n}$ . Рес( $i, j$ ) будет отвечать за формирование элементов  $a_{ij}^{(k)}$  при  $k = 1:n$ . Чтобы упростить изложение, симметрия игнорируется.

Предположим, что для хранения  $a_{ij}^{(k)}$  в Рес( $i, j$ ) используется переменная  $a_{ij}^{(k)}$ . Заметим, что с переходом от  $A^{(k-1)}$  к  $A^{(k)}$  связаны четыре типа вычислений:

вычислений нет	: $a \leftarrow a$
квадратный корень	: $a \leftarrow \sqrt{a}$
деление	: $a \leftarrow a/d$
модификация	: $a \leftarrow a - bc$

Вспоминая нашу модель систолических вычислений (см. § 6.1.5), мы попытаемся

<sup>1)</sup> Имеется в виду запись с помощью циклов с параметрами  $k, j, i$ . — Прим. перев.

разработать локальную программу вида

**for**  $t = 1 : t_{final}$

Послать локальные данные соседям.

Получить данные от соседей.

Выполнить извлечение квадратного корня, деление, модификацию или бездействовать.

**end**

Происходящее в момент  $t$  в  $\text{Proc}(i, j)$  есть функция от  $t, i$  и  $j$ . Заметим, что при  $t = 1$  только  $\text{Proc}(1, 1)$  может сделать что-нибудь полезное, и это есть вычисление  $\sqrt{A(1, 1)}$ . Потом эта величина потребуется первой строке и первому столбцу процессоров, вычисляющих соответственно  $A(2:n, 1) \leftarrow A(2:n, 1) / A(1, 1)$  и  $A(1, 2:n) \leftarrow A(1, 2:n) / A(1, 1)$ . Теперь полагаем, что в нашей систолической модели на каждом шаге данные могут передвигаться только к соседнему процессору. Значит, эти шаги с делением могут иметь место в следующие моменты времени:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & x & x & x & x \\ 3 & x & x & x & x \\ 4 & x & x & x & x \\ 5 & x & x & x & x \end{bmatrix}.$$

Приступив к формированию множителей в строке и столбце, мы можем начать выполнение модификации  $A(2:n, 2:n) \leftarrow A(2:n, 2:n) - A(2:n, 1)A(1, 2:n)$ . Заметим, что  $\text{Proc}(i, j)$  нуждается в множителях, размещенных в  $\text{Proc}(i, 1)$  и  $\text{Proc}(1, j)$ . Можно так устроить, что эти две величины доберутся до  $\text{Proc}(i, j)$  в одно и то же время, если сразу же по мере готовности  $\text{Proc}(i, 1)$  отправит свой множитель вдоль строки  $i$ , а  $\text{Proc}(1, j)$  пошлет свой множитель вниз по столбцу  $j$ . Если мы определим массив

$$T^{(1)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix},$$

то  $T^{(1)}(i, j)$  – это временной шаг, на котором находится  $a_{ij}^{(1)}$ . Заметим, что на временном шаге  $t = 4$  мы можем вычислить  $a_{22}^{(2)}$ , и по аналогии с предыдущим рассуждением положим

$$T^{(2)} = \begin{bmatrix} . & . & . & . & . \\ . & 4 & 5 & 6 & 7 \\ . & 5 & 6 & 7 & 8 \\ . & 6 & 7 & 8 & 9 \\ . & 7 & 8 & 9 & 10 \end{bmatrix}.$$

Здесь  $T^{(2)}(i, j)$  при  $i \geq 2$  и  $j \geq 2$  обозначает временной шаг, на котором вычисляется  $a_{ij}^{(2)}$ . Позиции с «.» не связаны с какой-либо работой или обменами, и поэтому они не играют роли, когда обсуждается планирование вычислений.

Время вычисления «интересных» элементов для  $A^{(3)}, A^{(4)}$  и  $A^{(5)}$  в сходной манере

задается с помощью

$$T^{(3)} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 7 & 8 & 9 \\ \cdot & \cdot & 8 & 9 & 10 \\ \cdot & \cdot & 9 & 10 & 11 \end{bmatrix}, \quad T^{(4)} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 10 & 11 \\ \cdot & \cdot & \cdot & 11 & 12 \end{bmatrix},$$

$$T^{(5)} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 13 \end{bmatrix}.$$

Из этого примера для  $n = 5$  мы извлечем три вывода:

- Процессор  $\text{Proc}(i,j)$  не имеет для себя вычислений до момента  $t = i + j - 1$ .
- Если  $a_{kk}^{(k)}$  определено при  $t = \tau$ , то  $a_{ij}^{(k)}$  может быть вычислено при  $t = \tau + (i - k) + (j - k)$  для всех  $i \geq k$  и  $j \geq k$ .
- Если  $a_{kk}^{(k)}$  определено при  $t = \tau$ , то  $a_{k+1,k+1}^{(k+1)}$  может быть определено при  $t = \tau + 3$ .

Эти наблюдения позволяют нам вывести несколько важных фактов относительно планирования. Например, диагональному процессору  $\text{Proc}(i,i)$  нечего делать при  $t = 1 : 2i - 2$ . При  $t = 2i - 2 : 3i - 3$  ему приходится выполнять модификации, а вслед за ними, в момент  $t = 3i - 2$ , извлечение квадратного корня. При  $t \geq 3i - 1$  ему снова нечего делать. К подобным заключениям можно прийти и в отношении внедиагональных процессоров, и в целом для  $\text{Proc}(i,j)$  мы получаем следующее расписание:

```

for t = 1 : 3n - 2
    if i = j
        if t ≥ 2i - 1 ∧ t < 3i - 2
            модификация
        elseif t = 3i - 2
            квадратный корень
        end
    elseif i > j
        if t ≥ i + j - 1 ∧ t < i + 2j - 2
            модификация
        elseif t = i + 2j - 2
            деление
        end
    elseif i < j
        if t ≥ i + j - 1 ∧ t < j + 2i - 2
            модификация
        elseif t = j + 2i - 2
            деление
        end
    end
end

```

Всего требуется  $3n - 2$  шагов, потому что  $\text{Proc}(n,n)$  вычисляет свой квадратный корень, т. е.  $g_{nn}$ , при  $t = 3n - 2$ .

Нам остается решать задачу организации обменов, отвечающих этим вычислительным действиям. Нужно сделать два наблюдения:

- На временном шаге после того, как  $\text{Proc}(i,j)$  вычислит  $g_{ij}$ , т. е. после  $t = i + j + \min\{i,j\} - 1$ , все, что он должен сделать,—это послать  $g_{ij}$  на восток, если  $i > j$  и  $j < n$ , на юг, если  $j > i$  и  $i < n$ , и на восток и юг, если  $i = j < n$ .
- На временных шагах  $t = i + j - 1 : i + j + \min\{i,j\} - 3$   $\text{Proc}(i,j)$  выполняет модификацию, т. е. вычисление вида  $a \leftarrow a - a_{\text{north}} a_{\text{west}}$ , где  $a_{\text{north}}$  поступает с севера, а  $a_{\text{west}}$ —с запада. Чтобы эти данные продолжили свое собственное передвижение,  $\text{Proc}(i,j)$  должен отправить текущее  $a_{\text{north}}$  на юг и текущее  $a_{\text{west}}$  на восток.

В целом мы получаем следующую процедуру.

**Алгоритм 6.5.1.** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и положительно определенная и  $A = GG^T$ —её разложение Холецкого. Если в системическом  $n \times n$ -массиве с соединениями в виде сетки каждый процессор исполняет следующую программу, то по окончании процесса в  $\text{Proc}(i,j)$  локальная переменная  $a$  содержит  $g_{ij}$  при  $i \geq j$  и  $g_{ji}$  при  $j \geq i$ .

```

loc. init [(i,j) = my. id, north, east, south, west, n, a = A(i,j)]
for t = 1:3n - 2
    if i = j
        {Вычисления диагональных процессоров.}
        if t = 2i - 1 ∧ i = 1
            recv(anorth, north)
            recv(awest, west)
            a = a - anorth awest
        elseif 2i - 1 < t < 3i - 2
            if i ≠ n
                send(anorth, south)
                send(awest, east)
            end
            recv(anorth, north)
            recv(awest, west)
            a = a - anorth awest
        elseif t = 3i - 2
            a = √a
        elseif t = 3i - 1 ∧ i ≠ n
            send(a, south)
            send(a, east)
        end
    elseif i > j
        {Вычисления поддиагональных процессоров.}
        if t = i + j - 1 ∧ j ≠ 1
            recv(anorth, north); recv(awest, west)
            a = a - anorth awest
        elseif i + j - 1 < t < i + 2j - 2
            if i ≠ n
                send(anorth, south)
            end
            send(awest, east)
            recv(anorth, north); recv(awest, west)
            a = a - anorth awest
        end
    end
end

```

```

elseif  $t = i + 2j - 2$ 
    recv( $a_{north}$ , north);  $a = a / a_{north}$ 
elseif  $t = i + 2j - 1$ 
    if  $i \neq n$ 
        send( $a_{north}$ , south)
    end
    send( $a$ , east)
end
elseif  $i < j$ 
    {Вычисления наддиагональных процессоров.}
    if  $t = i + j - 1 \wedge i \neq 1$ 
        recv( $a_{north}$ , north); recv( $a_{west}$ , west)
         $a = a - a_{north} a_{west}$ 
    elseif  $i + j - 1 < t < 2i + j - 2$ 
        if  $j \neq n$ 
            send( $a_{west}$ , east)
        end
        send( $a_{north}$ , south)
        recv( $a_{north}$ , north); recv( $a_{west}$ , west)
         $a = a - a_{north} a_{west}$ 
    elseif  $t = 2i + j - 2$ 
        recv( $a_{west}$ , west);  $a = a/a_{west}$ 
    elseif  $t = 2i + j - 1$ 
        if  $j \neq n$ 
            send( $a_{west}$ , east)
        end
        send( $a$ , east)
    end
end
quit

```

Можно было бы сжать этот алгоритм (его представил C. Paige), но его настоящая форма точнее отражает его вывод.

Отметим, что алгоритм 6.5.1 является процедурой с линейным временем. Конечно, в каждый момент времени занят работой не каждый процессор, и, следовательно, здесь есть некоторая неэффективность. А именно, среднее число активных процессоров на любом заданном временном шаге определяется как  $(n^3/3)/((3n-2)n^2) \approx 1/9$ .

Для алгоритма 6.5.1 легко формулируется блочная версия. Она представляет интерес, когда порядок матрицы больше, чем размер сетки. Если размеры сетки  $N \times N$ , то мы рассматриваем  $A$  как блочную  $N \times N$ -матрицу. Если связи то-роидальные, то для них подходящими могут быть другие отображения кругового типа. Можно сформулировать также версию алгоритма, в которой учитывается симметричность, а для реализации берется нижняя треугольная систолическая сеть процессоров.

## 6.5.2. Асинхронная сеточная процедура Холецкого

При выводе асинхронной, основанной на передаче сообщений процедуры Холецкого для сетки процессоров, в отличие от систолического случая мы не спрашиваем, «когда» вычисляется  $a_{ij}^{(k)}$ . Вместо этого у нас Proc( $i, j$ ) «дает старт»

вычислениям в зависимости от локального счетчика, отмечающего число завершенных шагов модификации  $a \leftarrow a - a_{\text{west}} a_{\text{north}}$ . В следующем массиве указывается число модификаций, в которых участвует Proc( $i, j$ ) до того, как он будет выполнять деление (если  $i \neq j$ ) или извлечение квадратного корня (если  $i = j$ ):

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 3 & 3 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix}.$$

Экстраполируя этот пример, заключаем, что вход  $(i, j)$  определяется как  $\min(i, j) - 1$ , и мы получаем:

**Алгоритм 6.5.2.** Предположим, что матрица  $A \in \mathbf{R}^{n \times n}$  симметричная и положительно определенная и  $A = GG^T$  – ее разложение Холецкого. Если в  $n \times n$ -массиве соединенных в сетку процессоров каждый из них исполняет следующую программу; то по окончании процесса локальная переменная  $a$  в Proc( $i, j$ ) содержит  $g_{ij}$  для  $i \geq j$  и  $g_{ij}$  для  $j \geq i$ .

```
loc.init [( $i, j$ ) = my.id, north, east, south, west, n, a = A( $i, j$ )]
```

{Модифицировать фазу.}

$k = 1$

while,  $k < \min(i, j)$

recv( $a_{\text{north}}$ , north); recv( $a_{\text{west}}$ , west)

$a = a - a_{\text{west}} a_{\text{north}}$

if  $i < n$

send( $a_{\text{north}}$ , south)

end

if  $j < n$

send( $a_{\text{west}}$ , east)

end

$k = k + 1$

end

{Свернуть по кругу фазу.}

if  $i = j$

$a = \sqrt{a}$

if  $i < n$

send( $a$ , east); send( $a$ , south)

end

elseif  $i > j$

recv( $a_{\text{north}}$ , north)

if  $i < n$

send( $a_{\text{north}}$ , south)

end

$a = a / a_{\text{north}}$

send( $a$ , east)

elseif  $i < j$

recv( $a_{\text{west}}$ , west)

if  $j < n$

send( $a_{\text{west}}$ , east)

end

$a = a / a_{\text{west}}$

send( $a$ , a<sub>south</sub>)

end

quit

Заметим, что вывод алгоритма 6.5.2 не содержит детального анализа синхронизации, как в его систолическом аналоге – алгоритме 6.5.1. Для того чтобы установить корректность процедуры, еще требуется доказать несколько фактов, относящихся к планированию вычислений. Например, если  $k < j < i$ , то можно показать, что  $k$ -м сообщением, прибывающим в  $\text{Proc}(i, j)$  с запада, будет  $g_{ik}$ . Аналогично  $g_{kj}$  является  $k$ -м сообщением, прибывающим с севера. Результаты, подобные этим, можно затем использовать для строгого доказательства того, что по окончании процесса  $\text{Proc}(i, j)$  хранит у себя  $g_{ij}$ .

### 6.5.3. Сеточная систолическая процедура QR-разложения

Частичный выбор ведущего элемента не очень хорошо отображается на двумерные массивы процессоров, потому что перестановка строк может быть связана с обменом данными между удаленными процессорами. По этой причине QR-разложение оказалось довольно привлекательным методом решения линейных уравнений, особенно на систолических массивах. Напомним, что согласно § 5.2.3 для вычисления QR-разложения матрицы  $A \in \mathbb{R}^{m \times n}$  можно использовать вращения Гивенса, причем это можно сделать многими способами. Мы остановимся на алгоритме 5.2.2, в котором используются вращения Гивенса для соседних строк:

```

for j = 1:n
    for i = m:-1:j + 1
        [c, s] = givens(A(i - 1:i, j))
        A(i - 1:i, j:n) = row. rot(A(i - 1:i, j:n), c, s)
    end
end

```

Формирование  $Q$  не будем рассматривать, так как это только затруднит обсуждение распараллеливания.

Чтобы построить параллельную версию для (6.5.1), посмотрим на некоторые промежуточные этапы редукции. Предположим, что  $(m, n) = (6, 3)$  и уже аннулированы  $a_{61}$  и  $a_{51}$ :

$$\left[ \begin{array}{ccc} x & x & x \\ 0 & x & x \\ 0 & x & x \end{array} \right].$$

Следующий шаг алгоритма – аннулировать  $a_{41}$ . Но заметим, что мы готовы также к тому, чтобы аннулировать  $a_{62}$ . Поэтому давайте то и другое сделаем одновременно:

$$\left[ \begin{array}{ccc} x & x & x \\ 0 & x & x \\ 0 & x & x \end{array} \right] \xrightarrow{(3,4),(5,6)} \left[ \begin{array}{ccc} x & x & x \\ x & x & x \\ x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & 0 & x \end{array} \right].$$

На следующем «временном шаге» можно ввести еще одну пару нулей:

$$\left[ \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{array} \right] \xrightarrow{(2,3),(4,5)} \left[ \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{array} \right].$$

На этом этапе можно одновременно провести анулирование  $a_{21}$ ,  $a_{42}$  и  $a_{61}$ :

$$\left[ \begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{array} \right] \xrightarrow{(1,2),(3,4),(5,6)} \left[ \begin{array}{ccc} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{array} \right].$$

Как видим, на языке шагов по времени в случае  $6 \times 3$  нули могут вводиться в следующие моменты:

$$\left[ \begin{array}{ccc} \times & \times & \times \\ 5 & \times & \times \\ 4 & 6 & \times \\ 3 & 5 & 7 \\ 2 & 4 & 6 \\ 1 & 3 & 5 \end{array} \right].$$

Вообще, поддиагональный матричный элемент  $a_{ij}$  может быть обнулен в момент

$$t_{ij} = (2j - 1) + m - i = m + 2j - i - 1. \quad (6.5.2)$$

Значит, компонента  $i = m + 2j - t$  столбца  $j$  может быть анулирована в момент  $t$ , при условии что  $m \leq i \leq j + 1$ . Таким образом, мы получаем следующий пересмотренный вариант для (6.5.1):

```

for t = 1:m + n - 2
    for j = 1:n
        i = m - t + 2j - 1
        if i <= m & i >= j + 1
            {Сформировать и применить  $G_{ij}$ }
            [c, s] = givens(A(i - 1:i, j))
            A(i - 1:i, j:n) = row. rot(A(i - 1:i, j:n), c, s)
        end
    end
end

```

(6.5.3)

Общее число требуемых временных шагов равно  $m + n - 2$ , потому что последним обнуляется элемент  $a_{n+1, n}$  и это происходит при  $t = m + n - 2$ .

С точки зрения параллельных вычислений ключевое наблюдение относительно (6.5.3) состоит в том, что все вращения, связанные с каким-либо заданным «временным шагом»  $t$ , являются «свободными от конфликтов» и могут выполняться одновременно. В течение стабильного периода ( $2n - 1 \leq t \leq m - 1$ ) обнуляется по одному элементу на столбец.

Если у нас есть систематическая сетка такого же размера, как и  $A$ , то возможна реализация с линейным временем:

- Proc( $i, j$ ) хранит у себя текущее  $a_{ij}$ .
- Proc( $i, j$ ) генерирует гивенсовое вращение  $G_{ij}$ .
- После генерации вращения  $G_{ij}$  оно передвигается на восток вдоль процессорных строк  $i - 1$  и  $i$ .

В такой схеме  $r_{ij}$  появляется в Proc( $i, j$ ). Если проследить за деятельностью Proc( $i, j$ ), то обнаружится, что на начальном периоде он пристаивает. Затем он занимается получением вращений, связанных с аннулированием в строках  $i + 1$  и  $i$ . В течение этого периода он модифицирует хранимые у него  $a_{ij}$  и передает полученные вращения на запад. Наконец, он вырабатывает  $G_{ij}$  и после этого пристаивает. Мы видели такой образец занятости в алгоритме 6.5.1. Как и в той постановке, в любой момент времени активной является значительная часть процессоров, а поскольку всего их  $n^2$ , получается процедура с линейным временем. Детали см. в Heller, Ipsen (1983) и Gentleman, Kung (1982).

### Задачи

**6.5.1.** Модифицировать алгоритм 6.5.1 и 6.5.2 так, чтобы они вычисляли: а)  $LDL^T$ -разложение; в) LU-разложение. Позаботьтесь о том, чтобы указать распределение множителей на входе.

### Замечания и литература к § 6.5

Работы по параллельному вычислению разложений LU и Холецкого включают

- Brent R. P. and Luk F. T. (1982). "Computing the Cholesky Factorization Using a Systolic Architecture", Proc. 6th Australian Computer Science Conf. 295–302.  
 Costnard M., Marrakchi M. and Robert Y. (1988). "Parallel Gaussian Elimination on an MIMD Computer", Parallel Computing 6, 275–296.  
 Delosme J. M. and Ipsen I. C. F. (1986). "Parallel Solution of Symmetric Positive Definite Systems with Hyperbolic Rotations", Lin. Alg. and Its Appl. 77, 75–112.  
 Funderlic R. E. and Geist A. (1986). "Torus Data Flow for Parallel Computation of Misized Matrix Problems", Lin. Alg. and Its Appl. 77, 149–164.  
 O'Leary D. P. and Stewart G. W. (1985). "Data Flow Algorithms for Parallel Matrix Computations", Comm. of the ACM 28, 841–853.

Параллельные методы для ленточных систем включают

- Johnsson S. L. (1984). "Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution of Tridiagonal Systems of Equations", Report YALEU/CSD/RR-339, Dept. of Comp. Sci., Yale University, New Haven, CT.  
 Johnsson S. L. (1985). "Solving Narrow Banded Systems on Ensemble Architectures", ACM Trans. Math. Soft. 11, 271–288.  
 Johnsson S. L. (1986). "Band Matrix System Solvers on Ensemble Architectures", in Supercomputers: Algorithms, Architectures, and Scientific Computations, eds. F. A. Matsen and T. Tajima, University of Texas Press, Austin TX., 196–216.  
 Johnsson S. L. (1987). "Solving Tridiagonal Systems on Ensemble Architectures", SIAM J. Sci. and Stat. Comp. 8, 354–392.  
 Johnsson S. L. and Ho C. T. (1987). "Multiple Tridiagonal Systems, the Alternative Direction Methods, and Boolean Cube Configured Multiprocessors", Report YALEU/DCS/RR-532, Dept. of Computer Science, Yale University, New Haven, CT.

Параллельные процедуры для QR-разложения представляют интерес для обработки сигналов в реальном времени. Подробности можно найти в

- Costnard M., Muller J. M., and Robert Y. (1986). "Parallel QR Decomposition of a Rectangular Matrix", Numer. Math. 48, 239–250.  
 Eldén L. (1988). "A Parallel QR Decomposition Algorithm", Report LiTh Mat R 1988-02, Dept. of Math., Linköping University Sweden.  
 Eldén L. and Schreiber R. (1986). "An Application by Systolic Arrays to Linear Discrete Ill-Posed Problems", SIAM J. Sci. and Stat. Comp. 7, 892–903.

- Gentlman W. M. and Kung H. T. (1982). "Matrix Triangularization by Systolic Arrays", SPIE Proceedings, Vol. 298, 19–26.
- Heller D. E. and Ipsen I. C. F. (1983). "Systolic Networks for Orthogonal Decompositions", SIAM J. Sci. and Stat. Comp. 4, 261–269.
- Luk F. T. (1986). "A Rotation Method for Computing the QR Factorization", SIAM J. Sci. and Stat. Comp. 7, 452–459.
- Modi J. J. and Clarke M. R. B. (1986). "An Alternative Givens Ordering", Numer. Math. 43, 83–90.

## 6.6. Методы разложения на общей памяти

В этом параграфе мы посмотрим на различные реализации разложения Холецкого на общей памяти. Каждая реализация обладает своим собственным образцом синхронизации, а вместе они дают представление о том, как разложение типа Холецкого может организовываться на системе с общей памятью. Мы выбрали разложение Холецкого ради удобства изложения, но большей частью то, о чем мы говорим для него, переносится и на другие разложения из нашего репертуара.

### 6.6.1. Статическое планирование для Холецкого с внешними произведениями

Начнем с распараллеливания алгоритма Холецкого с внешними произведениями:

```
for k = 1:n
    A(k:n) = A(k:n) / sqrt(A(k, k))
    for j = k + 1:n
        A(j:n, j) = A(j:n, j) - A(j:n, k) * A(j, k)
    end
end
```

Цикл по  $j$  отвечает за модификацию с использованием внешнего произведения. Его тело образуют  $n - k$  операций саxpy, которые независимы и легко распределяются.

**Алгоритм 6.6.1.** Предположим, что симметричная положительно определенная матрица  $A \in \mathbb{R}^{n \times n}$  хранится в общей памяти, к которой имеют доступ  $p$  процессоров. Если каждый процессор исполняет следующий алгоритм, то по окончании процесса нижняя треугольная часть матрицы  $A$  заменяется ее множителем Холецкого.

```
shared variables [A(1:n, 1:n)]
glob. init [A]
loc. init [p, μ = my.id, n]
for k = 1:n
    if μ = 1
        get(A(k:n), v(k:n)), v(k:n) = v(k:n) / sqrt(v(k))
        put(v(k:n), A(k:n, k))
    end
    barrier
    get(A(k + 1:n, k), v(k + 1:n))
    for j = (k + μ):p:n
        get(A(j:n, j), w(j:n)); w(j:n) = w(j:n) - v(j) * v(j:n)
        put(w(j:n), A(j:n, j))
    end
    barrier
end
quit
```

Нормирование перед  $j$ -циклом представляет собой очень малый объем работы по сравнению с модификацией, использующей внешние произведения. Поэтому эту порцию вычислений разумно поручить какому-то одному процессору. Отметим, что требуются два предложения **barrier**. Первое гарантирует, что процессор не начнет работу по прибавлению  $k$ -го внешнего произведения до тех пор, пока у Proc(1) не будет доступа к  $k$ -му столбцу из  $G$ . Второе предложение **barrier** не дает начать выполнение  $k + 1$ -го шага до полного завершения  $k$ -го шага.

Можно сформулировать и блочную версию алгоритма 6.6.1. Пусть  $n = rN$ . При  $k = 1:N$  мы (a) поручаем Proc(1) сформировать  $G(:, 1 + (k - 1)r : kr)$  и (b) заставляем все процессоры участвовать в модификации ранга  $r$  для оставшейся подматрицы  $A(kr + 1:n, kr + 1:n)$ . См. § 4.2.6. Если по отдельности процессоры хорошо воспринимают операции 3-го уровня, то производительность можно улучшить за счет укрупнения вычислительных блоков.

### 6.6.2. Статическое планирование для других разложений

Логика, связанная с алгоритмом 6.6.1, распространяется и на другие схемы разложений, такие, как LU и QR. Точно так же, как было сделано в § 6.4.3 для кольцевых разложений, мы можем предъявить «шаблон», в котором отражается образец синхронизации для реализуемых на общей памяти разложений с внешними произведениями:

```

shared variables [ $A, \dots$ ]
glob. init [ $A, \dots$ ]
loc. init [ $p, \mu = my.id, \dots$ ]
for  $k = 1:n$ 
  if  $\mu = 1$ 
    get ( $A(k:n), v(k:n)$ )
    {Сформировать  $k$ -е преобразование  $T_k$ .}
    {Записать  $T_k$  в общую память.}
  end
  barrier
  {Скопировать  $T_k$  в локальную память.}
  for  $j = k + \mu:p:n$ 
    {Применить  $T_k$  к  $A(k:n, j)$ .}
  end
  barrier
end `
```

(6.6.1)

Конечно, ссылки на  $T_k$  являются ссылками на некоторое подходящее представление для него, например на  $k$ -й вектор Хаусхолдера.

### 6.6.3. Две параллельные реализации для Холецкого с гахру

Теперь мы сконструируем пару параллельных процедур для разложения Холецкого на базе гахру-версии, т. е. на базе алгоритма 4.2.1:

```

for  $j = 1:n$ 
  for  $k = 1:j - 1$ 
     $A(j:n, j) = A(j:n, j) - A(j, k)A(j:n, k)$ 
  end
     $A(j:n, j) = A(j:n, j) / \sqrt{A(j, j)}$ 
  end
```

Обсудим два подхода к распараллеливанию. В первом из них каждый процессор участвует в каждой операции гахру. Вклад каждого процессора можно накапливать с помощью мониторинговой процедуры **gax.sum** из § 6.2.7. Тогда мы получим

**Алгоритм 6.6.2.** Предположим, что симметричная положительно определенная  $n \times n$ -матрица  $A$  хранится в общей памяти, доступной для  $p$  процессоров. Если каждый процессор исполняет следующий алгоритм, то по окончании процесса нижняя треугольная часть матрицы  $A$  замещается ее множителем Холецкого.

```

shared variables [ $A(1:n, 1:n)$ ]
glob.init [ $A$ ]
loc.init [ $p, \mu = my.id, n$ ]
for  $j = 1:n$ 
     $u(j:n) = 0$ 
    for  $k = \mu : p:j - 1$ 
        get ( $A(j:n, k), a_{loc}(j:n)$ )
         $u(j:n) = u(j:n) - a_{loc}(j) a_{loc}(j:n)$ 
    end
    call gax.sum ( $u(j:n), A(j:n, j)$ )
    barrier
    if  $\mu = 1$ 
        get ( $A(j:n, j), a_{loc}(j:n)$ )
         $a_{loc}(j:n) = a_{loc}(j:n) / \sqrt{a_{loc}(j)}$ 
        put ( $a_{loc}(j:n), A(j:n, j)$ )
    end
    barrier
end
```

Первое предложение **barrier** гарантирует, что нормирование, приводящее к  $G(j:n, j)$ , не будет начато до окончания гахру. Второе предложение **barrier** дает уверенность в том, что ни один из процессоров не перейдет к  $j + 1$ -му шагу прежде, чем завершится  $j$ -й шаг.

Второй подход к распараллеливанию основанной на гахру процедуры Холецкого состоит в применении идеи резерва задач, где  $j$ -й задачей является вычисление  $G(j:n, j)$ . Поскольку этот подстолбец множителя Холецкого получается нормированием из

$$v(j:n) = A(j:n, j) - \sum_{k=1}^{j-1} G(j, k) G(j:n, k), \quad (6.6.2)$$

мы видим, что  $j$ -я задача не может завершиться прежде, чем будут окончены задачи от 1-й до  $j - 1$ -й. Итак, мы добиваемся процедуры следующего вида:

Повторять до тех пор, пока есть задачи.

Получить следующую задачу, присвоить ей номер  $j$ .

Построить вектор  $v(j:n)$  вида (6.6.2) наискорейшим образом.

Вычислить  $G(j:n, j)$  и сохранить в общей памяти.

Проинформировать все процессоры, ожидающие  $G(j:n, j)$ , что он доступен.

Конец

Для назначения задач мы можем использовать разработанный в § 6.2.6 монитор **nexti**. Чтобы осуществить построение  $G$ -столбцов и замещение элементов в  $A$ , нужен еще один монитор **chol** с двумя процедурами **canread** и **write**.

```

monitor: chol
  protected variables: cols.done
  condition variables: waitq (·)
  initializations: cols.done = 0
  procedure chol.canread (z, k, j, n)
    if cols.done < k
      delay (waitq(k))
    end
    get (A(j:n, k), z(j:n))
    continue (waitq(k))
  end chol.canread
  procedure chol.write (v, j, n)
    put (v(j:n), A(j:n,j))
    cols.done = cols.done + 1
    if cols.done = n
      cols.done = 0
    else
      continue (waitq(j))
    end
  end chol.write
end chol

```

Этот монитор обладает некоторым числом новых свойств по сравнению с **nexti** и **gах**. Для начала он содержит две процедуры. Обращение либо к **chol.canread**, либо к **chol.write** представляет собой попытку вызывающей программы войти в монитор. Однако в любой момент в мониторе по-прежнему может быть не более одного активного процесса. На эти две процедуры возлагаются следующие задачи:

- **chol.canread** ( $z, k, j, n$ ) возвращает  $G(j:n, k)$  в  $z(j:n)$ , если оно готово. Если  $G(j:n, k)$  не готово, то вызывающий процессор становится ожидающим.
- **chol.write** ( $v, j, n$ ) копирует  $v(j:n) = G(j:n, j)$  в  $A(j:n, j)$  и информирует все процессоры, ожидающие именно этот  $G$ -столбец, что теперь он доступен и находится в общей памяти.

Отметим, что используется вектор переменных условия. Если  $G(:, k)$  не доступен, то процессор, запрашивающий часть из  $G(:, k)$ , становится ожидающим – в зависимости от  $waitq(k)$ .

Защищенная переменная  $col.done$  отслеживает число готовых столбцов. Таким образом, проверка готовности для какого-либо конкретного  $G$ -столбца является простым делом. Заметим, что после вычисления последнего  $G$ -столбца  $col.done$  устанавливается на нуль.

Вот как используется монитор **chol** в параллельной **gахру**-процедуре Холецкого:

**Алгоритм 6.6.3.** Предположим, что симметричная положительно определенная  $n \times n$ -матрица  $A$  хранится в общей памяти, доступной для  $p$  процессоров. Если каждый процессор исполняет следующий алгоритм, то по окончании процесса нижняя треугольная часть матрицы  $A$  замещается нижней треугольной частью ее множителя Холецкого.

```

shared variables [ $A(1:n, 1:n)$ ]
glob. init [ $A$ ]
loc. init [ $p, \mu = my.id, n$ ]
call nexti. index ( $p, n, more, j$ )

```

```

while more-true
  get (A(:j:n,j), v(j:n))
  for k = 1:j-1
    call chol.canread(z, k, j)
    v(j:n) = v(j:n) - z(j)z(j:n)
  end
  v(j:n) = v(j:n)/sqrt(v(j))
  call chol.write(v, j, n)
  call nexti.index(p, n, more, j)
end
quit

```

Напомним, что процедура `nexti.index` сконструирована так, что ни один процессор не может выйти из цикла `while` до полного завершения всех задач.

Алгоритм 6.6.3 схож с нашими кольцевыми процедурами Холецкого из § 6.4. Именно, каждый процессор испытывает период простояния как в начале, так и в конце всего комплекса вычислений.

Можно сформулировать и блочную столбцовую версию алгоритма 6.6.3, основанную на разбиении

$$G = [G_1, \dots, G_N]. \quad (6.6.3)$$

Теперь  $i$ -й задачей является вычисление блочного столбца  $G_i$ . Вследствие более крупного дробления вычисления процессоры, возможно, в состоянии извлечь пользу из линейной алгебры более высокого уровня, навеянной блочным разбиением.

#### 6.6.4. Замечания об обменах с общей памятью

В алгоритмах 6.6.1–6.6.3 интересно оценить время, которое каждый процессор тратит на обмены. При больших  $n/p$  эти методы дают равномерную загруженность, и в этой ситуации разумно предположить, что на обмены с общей памятью каждый процессор тратит  $T/p$  секунд, где  $T$  есть сумма всех издержек, связанных с `get` и `put`. Используя нашу модель обменов (6.2.2), находим, что в алгоритме 6.6.1 каждый процессор тратит

$$T_{6.6.1} \approx \frac{n^2}{p} \alpha_s + \frac{2n^3}{3p} \beta_s$$

секунд, общаясь с общей памятью. Будучи основанными на гахре, алгоритмы 6.6.2 и 6.6.3 содержат меньше обменов:

$$T_{6.6.2} \approx T_{6.6.3} \approx \frac{n^2}{2p} \alpha_s + \frac{n^3}{6p} \beta_s.$$

Конечно, это лишь грубые оценки. Тем не менее они еще раз указывают на то, что лучше опираться на операции гахре, чем на внешние произведения.

#### 6.6.5. Блочный Холецкий на базе резерва задач

Предположим, что  $A$  – симметричная положительно определенная  $n \times n$ -матрица и  $n = Nr$ . Вспоминая наше обсуждение блочного Холецкого со скалярными произведениями (см. § 4.2.6), мы получаем центральное уравнение

$$G_{ij} G_{jj}^T = A_{ij} - \sum_{k=1}^{j-1} G_{ik} G_{jk}^T, \quad (6.6.4)$$

приравнивая  $(i, j)$ -блоки в  $A = GG^T$ . Здесь все  $G_{ij}$  и  $A_{ij}$  суть  $r \times r$ -блоки. Заметим, что если  $i > j$ , то вычисление  $G_{ij}$  не может закончиться, пока не будут готовы  $G_{i,1}, \dots, G_{i,j-1}$  и  $G_{j,1}, \dots, G_{j,j}$ . Если  $j = i$ , то вычисление  $GG_{jj}$  не может закончиться, пока не будут готовы  $G_{j,1}, \dots, G_{j,j-1}$ . В свете этих замечаний разумно вычислять  $G_{ij}$  в порядке столбец за столбцом. В случае  $4 \times 4$  это означает, что  $G$ -блоки отыскиваются в таком порядке:

1			
2	5		
3	6	8	
4	7	9	10

Пусть задача  $(i, j)$  – это вычисление  $G_{ij}$ . Следуя рассуждениям, которые стоят за нашей процедурой Холецкого с использованием резерва задач, нам нужна мониторинговая процедура для назначения следующей задачи. Для этого подходит простая модификация мониторинговой процедуры **nextij.index** (см. § 6.3.8):

```

monitor: cholij
  protected variables row, col, idle.proc
  condition variables waiti
  initializations: flag = 0
  procedure cholij.index (N, p, more, i, j)
    if flag = 0
      row = 0; col = 1
      idle. proc = 0; flag = 1
    end
    if row < N
      row = row + 1
    else
      col = col + 1; row = col
    end
    if col ≤ N
      i = row; j = col
      more = true
    else
      more = false
      idle. proc = idle. proc + 1
      if idle. proc ≤ p - 1
        delay (waiti)
      else
        flag = 0
      end
      continue (waiti)
    end
  end cholij.index
end cholij

```

Используя **cholij.index**, мы можем организовать нашу параллельную процедуру для блочного Холецкого следующим образом:

```

call cholij.index ( $N, p, more, i, j$ )
while  $more = true$ 
  get ( $A_{ij}, S$ )
  for  $k = 1: j - 1$ 
    {В случае готовности получить  $G_{ik}$  и  $G_{jk}$  и вычислить модификацию
      $S = S - G_{ik} G_{jk}^T$ }
    end
    if  $i = j$ 
      {Решить  $G_{jj} G_{jj}^T = S$ , применив алгоритм 4.2.1.}
    else
      {Решить  $G_{ij} G_{jj}^T = S$  относительно  $G_{ij}$ .}
    end
    {Записать  $G_{ij}$  в общую память и проинформировать все процессоры, которые
     могут ожидать  $G_{ij}$ , о его готовности.}
    call cholij.index ( $N, p, more, i, j$ )
  end

```

Анализируя (6.6.4), мы обнаруживаем необходимость в еще одном мониторе для надзора за обращением к  $A$ , когда на ее месте получается  $G$ . В частности, нам нужны блочные версии мониторинговых процедур **chol.canread** ( $z, k, j, n$ ) и **chol.write** ( $v, j, n$ ). Другими словами, нам нужен монитор с процедурами **bloc.chol.canread** и **bloc.chol.write**, где:

- **block.chol.canread** ( $w, i, k$ ) возвращает  $G_{ik}$  в  $W$ , если оно готово. Если  $G_{ik}$  не готово, то вызывающий процессор становится ожидающим.
- **block.chol.write** ( $w, i, j$ ) копирует  $W = G_{ij}$  в  $A_{ij}$  и информирует все процессоры о том, что оно готово и находится в общей памяти.

Вот детали:

```

monitor: block.chol
  protected variables: row.progress (·)
  condition variables: waitq (·, ·)
  initializations: row.progress(·) = 0
  procedure block.chol.canread ( $w, i, k$ )
    if row.progress( $i$ ) <  $k$ 
      delay (waitq( $i, k$ ))
    end
    get ( $A_{ik}, W$ )
    continue (waitq( $i, k$ ))
  end block.chol.canread
  procedure block.chol.write ( $S, i, j, N$ )
    put ( $S, A_{ij}$ )
    row.progress( $i$ ) = row.progress( $i$ ) + 1
    if row.progress( $N$ ) <  $N$ 
      continue (waitq( $i, j$ ))
    else
      row.progress (·) = 0
    end
  end block.chol.write
end block.chol

```

Этот монитор равнозначен двумерной версии для **chol**. Мы заводим вектор-счетчик *row.progress*( $1:N$ ), отслеживающий число вычисленных  $G_{ij}$  в каждой блочной строке.

В конце вычислений этот вектор обнуляется. Если процессор запрашивает блок  $G_{ik}$  и находит, что он еще не готов, то он становится ожидающим – в зависимости от  $waitq(i, k)$ . Всякий раз, когда процессор вычисляет блок  $G_{ij}$ , он начинает процесс с активизации всех процессоров, дожидающихся соответствующего разрешения от  $waitq(i, j)$ . Собирая все это вместе, получаем

**Алгоритм 6.6.4.** Пусть  $A$  – блочная  $N \times N$ -матрица с  $r \times r$ -блоками. Предположим, что она симметрична положительно определенная и хранится в общей памяти, доступной для  $p$  процессоров. Если каждый процессор исполняет следующий алгоритм, то нижняя треугольная часть матрицы  $A$  заменяется на нижнюю треугольную часть матрицы  $G$  – треугольник Холецкого. (Ссылки на  $A_{ij}$  в действительности являются ссылками на  $A(1 + (i - 1)r : ir, 1 + (j - 1)r : jr)$ .)

```

call cholij.index ( $N, p, more, i, j$ )
while  $more = true$ 
    get ( $A_{ij}, S$ )
    for  $k = 1:j - 1$ 
        call block.chol.canread ( $w, i, k$ )
        call block.chol.canread ( $Y, j, k$ )
         $S = S - WY^T$ 
    end
    if  $i = j$ 
        Решить  $G_{jj}G_{jj}^T = S$ , применив алгоритм 4.2.1 к  $S$ .
    else
        Решить  $G_{ij}G_{jj}^T = S$  относительно  $G_{ij}$ .
    end
    call block.chol.write ( $S, i, j, N$ )
    call cholij.index ( $N, p, more, i, j$ )
end
quit
```

Этот алгоритм можно распространить и на случай произвольного блочного разбиения.

Нетрудно показать, что каждый процессор, занятый исполнением алгоритма 6.6.4, тратит примерно

$$T_{6.6.4} \approx \frac{N^3(\alpha_s + \beta_s r^2)}{sp}$$

секунд на обмен с общей памятью.

### Задачи

**6.6.1.** Используя шаблон (6.6.1), разработать параллельную процедуру для QR-разложения, в предположении что  $m \times n$ -матрица  $A$  хранится в общей памяти.

**6.6.2.** Разработать LU-версию алгоритма 6.6.2.

**6.6.3.** Для алгоритма 6.6.3 разработать LU-версию с частичным выбором.

**6.6.4.** Сравнить обмены с общей памятью, связанными с алгоритмами 6.6.3 и 6.6.4.

**6.6.5.** Приспособить алгоритм 6.6.4 и соответствующие мониторы для того, чтобы они были эффективными в случае блочно-трехдиагональной  $A$ .

**Замечания и литература к § 6.6**

Параллельные методы для разложений на компьютерах с общей памятью обсуждаются в

- Chen S., Dongarra J. and Hsuing C. (1984). "Multiprocessing Linear Algebra Algorithms on the Cray X-MP-2: Experiences with Small Granularity", *J. Parallel and Distributed Computing* 1, 22–31.
- Chen S., Kuck D. and Sameh A. (1978). "Practical Parallel Band Triangular Systems Solvers", *ACM Trans. Math. Soft.* 4, 270–277.
- Dongarra J. J. and Hewitt T. (1986). "Implementing Dense Linear Algebra Algorithms Using Multitasking on the Cray X-MP-4 (or Approaching the Gigaflop)", *SIAM J. Sci. and Stat. Comp.* 7, 347–350.
- Dongarra J. J. and Hiromoto R. E. (1984). "A Collection of Parallel Linear Equation Routines for the Denelcor HEP", *Parallel Computation* 1, 133–142.
- Dongarra J. J. and Sameh A. H. (1984). "On Some Parallel Banded System Solvers", *Parallel Computing* 1, 223–235.
- Dongarra J. J., Sameh A. and Sorensen D. (1986). "Implementation of Some Concurrent Algorithms for Matrix Factorization", *Parallel Computing* 3, 25–34.
- Dongarra J. J. and Sorensen D. C. (1987). "Linear Algebra on High Performance Computers", *Appl. Math. and Comp.* 20, 57–88.
- George A., Heath M. T. and Liu J. (1986). "Parallel Cholesky Factorization on a Shared Memory Multiprocessor", *Lin. Alg. and Its Applic.* 77, 165–187.
- Sameh A. and Kuck D. (1978). "On Stable Parallel Linear System Solvers", *J. Assoc. Comp. Mach.* 25, 81–91.
- Swarztrauber P. (1979). "A Parallel Algorithms for Solving General Tridiagonal Equations", *Math. Comp.* 33, 185–199.

## Глава 7

# Несимметричная проблема собственных значений

Рассмотрев линейные уравнения и метод наименьших квадратов, обратим внимание на третью важную область матричных вычислений – алгебраическую проблему собственных значений. В этой главе рассматривается несимметричная проблема, а более приятный симметричный случай – в следующей.

Наша первоочередная задача – познакомиться с разложениями Шура и Жордана и основными свойствами собственных значений и инвариантных подпространств. Противоположное поведение этих двух разложений положено в основу той части § 7.2, в которой мы исследуем, как на собственные значения и инвариантные подпространства влияет возмущение. Введены числа обусловленности, что позволяет получать оценку погрешности, которую можно ожидать из-за округления.

Основной алгоритм данной главы – заслуженно известный QR-алгоритм. Эта процедура является наиболее сложным алгоритмом, рассмотренным в этой книге, и ее изложение занимает три раздела. Мы получаем базовые QR-итерации в § 7.3 как естественное обобщение простейшего степенного метода. Последующие два раздела посвящены тому, чтобы сделать базовые итерации вычислительно осуществимыми. Это включает введение хессенбергова разложения в § 7.4 и идею начальных сдвигов в § 7.5.

QR-алгоритм вычисляет вещественную форму Шура матрицы – каноническую форму, которая выявляет собственные значения, но не собственные векторы. Поэтому обычно необходимо выполнить дополнительные вычисления, если требуется информация, касающаяся инвариантных подпространств. В § 7.6, который мог бы иметь подзаголовок: «Что делать после того, как найдена вещественная форма Шура», мы обсуждаем различные методы вычисления инвариантных подпространств, которые позволяют довести до конца QR-алгоритм.

Наконец, в последнем разделе мы рассматриваем обобщенную проблему собственных значений  $Ax = \lambda Bx$  и вариант QR-алгоритма, придуманный для ее решения. Этот алгоритм, называемый QZ-алгоритмом, подчеркивает важность ортогональных матриц в проблеме собственных значений – главную тему этой главы.

Сейчас самое время сделать замечание о комплексной арифметике. В этой книге мы сосредоточиваем внимание на развитии вещественных алгоритмов для вещественных задач. Данная глава не представляет исключения, хотя даже вещественная несимметричная матрица может иметь комплексные собственные значения. Однако при получении практического вещественного QR-алгоритма и при математическом анализе самой проблемы собственных значений удобнее работать в поле комплексных чисел. Поэтому читатель обнаружит, что мы перешли к комплексным обозначениям в § 7.1, 7.2 и 7.3. В этих разделах мы используем комплексное QR-разложение ( $A = QR$ ,  $Q$  – унитарная,  $R$  – комплексная верхняя треугольные матрицы) и комплексное SVD ( $A = U\Sigma V^H$ ,  $U$  и  $V$  – унитарные,  $\Sigma$  – ве-

щественная диагональная матрицы). Получение этих комплексных разложений отличается от вещественного случая не так сильно, чтобы оправдать отдельное описание.

## 7.1. Свойства и разложения

В этом разделе мы даем обзор математических основ, необходимых для построения и анализа обсуждаемых далее алгоритмов вычисления собственных значений.

### 7.1.1. Собственные значения и инвариантные подпространства

*Собственные значения* матрицы  $A \in \mathbb{C}^{n \times n}$  – это  $n$  корней ее характеристического многочлена  $p(z) = \det(zI - A)$ . Совокупность корней называют *спектром* и обозначают  $\lambda(A)$ . Если  $\lambda(A) = \{\lambda_1, \dots, \lambda_n\}$ , то

$$\det(A) = \lambda_1 \lambda_2 \dots \lambda_n.$$

Более того, если мы определим *след* (*trace*) матрицы  $A$  как

$$\text{trace}(A) = \sum_{i=1}^n a_{ii},$$

то  $\text{trace}(A) = \lambda_1 + \dots + \lambda_n$ . Это можно показать, рассматривая коэффициент при  $z^{n-1}$  в характеристическом многочлене.

Если  $\lambda \in \lambda(A)$ , то ненулевой вектор  $x \in \mathbb{C}^n$ , удовлетворяющий уравнению

$$Ax = \lambda x,$$

называют *собственным вектором*. Точнее,  $x$  является *правым собственным вектором*, соответствующим  $\lambda$ , если  $Ax = \lambda x$ , и *левым собственным вектором*, если  $x^T A = \lambda x^T$ . В дальнейшем, если не оговорено противное, «собственный вектор» означает «правый собственный вектор».

Собственный вектор определяет одномерное подпространство, инвариантное по отношению к умножению слева на матрицу  $A$ . В общем случае подпространство  $S \subseteq \mathbb{C}^n$ , обладающее свойством

$$x \in S \Rightarrow Ax \in S,$$

называют *инвариантным подпространством* (матрицы  $A$ ). Заметим, что если

$$AX = XB, \quad B \in \mathbb{C}^{k \times k}, \quad X \in \mathbb{C}^{n \times k},$$

то подпространство  $\text{range}(X)$  является инвариантным и

$$By = \lambda y \Rightarrow A(By) = \lambda(Xy).$$

Таким образом, если  $X$  имеет полный столбцовый ранг, то из  $AX = XB$  следует, что  $\lambda(B) \subseteq \lambda(A)$ . Если  $X$  – квадратная невырожденная матрица, то  $\lambda(A) = \lambda(B)$  и говорят, что матрицы  $A$  и  $B = X^{-1}AX$  *подобны*. В этом случае  $X$  называют *преобразованием подобия*.

Многие алгоритмы вычисления собственных значений используют разбиение исходной задачи на ряд задач меньшего размера. Следующий результат является основой таких преобразований.

**Лемма 7.1.1.** *Если матрица  $T \in \mathbb{C}^{n \times n}$  представлена в блочном виде*

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} p \\ q \end{matrix},$$

то  $\lambda(T) = \lambda(T_{11}) \cup \lambda(T_{22})$ .

*Доказательство.* Пусть

$$Tx = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

где  $x_1 \in \mathbb{C}^p$  и  $x_2 \in \mathbb{C}^q$ . Если  $x_2 \neq 0$ , то  $T_{22}x = \lambda x_2$  и  $\lambda \in \lambda(T_{22})$ . Если  $x_2 = 0$ , то  $T_{11}x_1 = \lambda_1 x_1$  и  $\lambda \in \lambda(T_{11})$ . Следовательно,  $\lambda(T) \subset \lambda(T_{11}) \cup \lambda(T_{22})$ . Но поскольку множества  $\lambda(T)$  и  $\lambda(T_{11}) \cup \lambda(T_{22})$  имеют одинаковое число элементов, эти два множества равны.  $\square$ .

### 7.1.2. Основные унитарные разложения

Используя преобразования подобия, можно привести заданную матрицу к любой из нескольких канонических форм. Эти канонические формы отличаются способом представления собственных значений и видом информации об инвариантных подпространствах, которую они позволяют получить. Учитывая численную устойчивость унитарных преобразований, мы начнем с обсуждения ре-дукций, которые реализуются унитарным подобием.

**Лемма 7.1.2.** Если матрицы  $A \in \mathbb{C}^{n \times n}$ ,  $B \in \mathbb{C}^{p \times p}$  и  $X \in \mathbb{C}^{n \times p}$  удовлетворяют условиям

$$AX = XB, \quad \text{rank}(X) = p, \tag{7.1.1}$$

то существует унитарная матрица  $Q \in \mathbb{C}^{n \times n}$ , такая, что

$$Q^H A Q = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \\ p & n-p \end{bmatrix} \quad \begin{matrix} p \\ n-p \end{matrix}, \tag{7.1.2}$$

где  $\lambda(T_{11}) = \lambda(A) \cap \lambda(B)$ .

*Доказательство.* Пусть

$$X = Q \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad Q \in \mathbb{C}^{n \times n}, \quad R \in \mathbb{C}^{p \times p},$$

есть QR-разложение матрицы  $X$ . Подставляя его в (7.1.1) и преобразуя полученное выражение, имеем

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} B,$$

где

$$Q^H A Q = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \\ p & n-p \end{bmatrix} \quad \begin{matrix} p \\ n-p \end{matrix}.$$

Из невырожденности матрицы  $R$  и уравнений  $T_{21}R = 0$  и  $T_{11}R = RB$  следует, что  $T_{21} = 0$  и  $\lambda(T_{11}) = \lambda(B)$ . Окончательный результат получаем, воспользовавшись леммой 7.1.1, согласно которой  $\lambda(A) = \lambda(T) = \lambda(T_{11}) \cup \lambda(T_{22})$ .  $\square$

**Пример 7.1.1.** Если

$$A = \begin{bmatrix} 67.00 & 177.60 & -63.20 \\ -20.40 & 95.88 & -87.16 \\ 22.80 & 67.84 & 12.12 \end{bmatrix},$$

$X = (20, -9, -12)^T$  и  $B = (25)$ , то  $AX = XB$ . Более того, если ортогональную матрицу  $Q$  определить как

$$Q = \begin{bmatrix} -8.00 & 0.360 & 0.480 \\ 0.360 & 0.928 & -0.096 \\ 0.480 & -0.096 & 0.872 \end{bmatrix},$$

то  $Q^T X = (-25, 0, 0)^T$  и

$$Q^T A Q = T = \begin{bmatrix} 25 & -95 & 5 \\ 0 & 147 & -104 \\ 0 & 146 & 3 \end{bmatrix}.$$

Заметим, что  $\lambda(A) = \{25, 75 + 100i, 75 - 100i\}$ .

Лемма 7.1.2 утверждает, что произвольная квадратная матрица может быть приведена к блочно-треугольной форме унитарными преобразованиями подобия, если известно одно из ее инвариантных подпространств. Применяя индукцию, мы можем легко получить разложение Шура [Schur (1909)].

**Теорема 7.1.3 (Разложение Шура).** Если  $A \in \mathbb{C}^{n \times n}$ , то существует унитарная матрица  $Q \in \mathbb{C}^{n \times n}$ , такая, что

$$Q^H A Q = T = D + N, \quad (7.1.3)$$

где  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ , а  $N \in \mathbb{C}^{n \times n}$  есть строго верхняя треугольная матрица. Более того, матрица  $Q$  может быть выбрана такой, чтобы собственные значения  $\lambda_i$  были расположены вдоль диагонали в любом заданном порядке.

**Доказательство.** Теорема очевидно верна при  $n = 1$ . Предположим, что она верна для всех матриц порядка  $n - 1$  или меньше. Если  $Ax = \lambda x$ , где  $x \neq 0$ , то согласно лемме 7.1.2 (при  $B = (\lambda)$ ) существует унитарная матрица  $U$ , такая, что:

$$U^H A U = \begin{bmatrix} \lambda & w^H \\ 0 & C \end{bmatrix} \quad \begin{matrix} 1 \\ n-1 \end{matrix}.$$

По предположению индукции найдется унитарная матрица  $\tilde{U}$ , такая, что матрица  $\tilde{U}^H C \tilde{U}$  – верхняя треугольная. Следовательно, если  $Q = U \text{diag}(1, \tilde{U})$ , то матрица  $Q^H A Q$  – верхняя треугольная.  $\square$

**Пример 7.1.2.** Если

$$A = \begin{bmatrix} 78 & 200 \\ -50 & 75 \end{bmatrix} \text{ и } Q = \begin{bmatrix} 0.8944i & 0.4472 \\ -0.4472 & -0.8944i \end{bmatrix},$$

то  $Q$  – унитарная матрица и

$$Q^H A Q = \begin{bmatrix} 75 + 100i & 150 \\ 0 & 75 + 100i \end{bmatrix}.$$

Если  $Q = [q_1, \dots, q_n]$  есть разбиение на столбцы унитарной матрицы  $Q$  из (7.1.3), то столбцы  $q_i$  называют *векторами Шура*. Сравнивая столбцы в уравнении  $AQ = QT$ , мы видим, что векторы Шура удовлетворяют уравнениям

$$Aq_k = \lambda_k q_k + \sum_{i=1}^{k-1} n_{ik} q_i, \quad k = 1:n. \quad (7.1.4)$$

Отсюда следует, что подпространства

$$S_k = \text{span}\{q_1, \dots, q_k\}, \quad k = 1:n,$$

являются инвариантными. Более того, нетрудно показать, что если  $Q = [q_1, \dots, q_n]$ , то  $\lambda(Q_k^H A Q_k) = \{\lambda_1, \dots, \lambda_k\}$ . Поскольку собственные значения в (7.1.3) могут иметь произвольный порядок, это означает, что существует по крайней мере одно  $k$ -мерное инвариантное подпространство для каждой подпоследовательности  $k$  собственных значений.

Другой вывод из (7.1.4) заключается в том, что вектор Шура  $q_k$  является собственным вектором тогда и только тогда, когда  $k$ -й столбец матрицы  $N$  – нулевой. Оказывается, что если  $k$ -й столбец матрицы  $N$  нулевой для всех  $k$ , то  $A^H A = A A^H$ . Матрицы, удовлетворяющие этому условию, называют *нормальными*.

**Следствие 7.1.4.** *Матрица  $A \in \mathbb{C}^{n \times n}$  является нормальной тогда и только тогда, когда существует унитарная матрица  $Q \in \mathbb{C}^{n \times n}$ , такая, что  $Q^H A Q = \text{diag}(\lambda_1, \dots, \lambda_n)$ .*

*Доказательство.* Нетрудно показать, что если  $A$  унитарно подобна диагональной матрице, то  $A$  нормальная. С другой стороны, если  $A$  является нормальной и  $Q^H A Q = T$  есть ее разложение Шура, то  $T$  является также нормальной. Осталось показать, что нормальная верхняя треугольная матрица является диагональной.  $\square$

Заметим, что если  $Q^H A Q = T = \text{diag}(\lambda_i) + N$  есть разложение Шура произвольной  $n \times n$ -матрицы  $A$ , то  $\|N\|_F$  не зависит от выбора  $Q$ :

$$\|N\|_F^2 = \|A\|_F^2 - \sum_{i=1}^n |\lambda_i|^2 = \Delta^2(A).$$

Эту величину называют отклонением от нормальности. Таким образом, для того чтобы сделать  $T$  «более диагональной», необходимо использовать неунитарные преобразования подобия.

### 7.1.3. Неунитарные преобразования

Для того чтобы увидеть, что представляет собой неунитарное преобразование подобия, мы рассмотрим блочную диагонализацию  $2 \times 2$ -блочно-треугольной матрицы.

**Лемма 7.1.5.** *Пусть матрица  $T \in \mathbb{C}^{n \times n}$  представлена в блочном виде*

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} p \\ q \end{matrix}.$$

*Определим преобразование  $\phi : \mathbb{C}^{p \times q} \rightarrow \mathbb{C}^{p \times q}$  равенством*

$$\phi(X) = T_{11}X - XT_{22},$$

*где  $X \in \mathbb{C}^{p \times q}$ . Преобразование  $\phi$  невырожденно тогда и только тогда, когда*

$\lambda(T_{11}) \cap \lambda(T_{22}) = \emptyset$ . Если  $\phi$  невырожденно и  $Y$  определено как

$$Y = \begin{bmatrix} I_p & Z \\ 0 & I_q \end{bmatrix}, \quad \phi(Z) = -T_{12},$$

то  $Y^{-1}TY = \text{diag}(T_{11}, T_{22})$ .

*Доказательство.* Предположим, что  $\phi(X) = 0$  для  $X \neq 0$  и что

$$U^H X V = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} r \\ p-r \\ r-q-r \end{matrix}$$

есть сингулярное разложение матрицы  $X$  с  $\Sigma_r = \text{diag}(\sigma_i), r = \text{rank}(X)$ . Подстановка этого разложения в уравнение  $T_{11}X = XT_{22}$  дает

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

где  $U^H T_{11} U = (A_{ij})$  и  $V^H T_{22} V = (B_{ij})$ . Сравнивая соответствующие блоки, мы видим, что  $A_{21} = 0$ ,  $B_{12} = 0$  и  $\lambda(A_{11}) = \lambda(B_{11})$ . Следовательно,

$$\emptyset \neq \lambda(A_{11}) = \lambda(B_{11}) \subseteq \lambda(T_{11}) \cap \lambda(T_{22}).$$

С другой стороны, если  $\lambda \in \lambda(T_{11}) \cap \lambda(T_{22})$ , то найдутся такие ненулевые векторы  $x$  и  $y$ , что  $T_{11}x = \lambda x$  и  $y^H T_{22}y = \lambda y^H$ . Можно показать, что  $\phi(x y^H) = 0$ . Наконец, если  $\phi$  невырожденно, то указанная матрица  $Z$  существует и

$$\begin{aligned} Y^{-1}TY &= \begin{bmatrix} I & -Z \\ 0 & I \end{bmatrix} \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} I & Z \\ 0 & I \end{bmatrix} \\ &= \begin{bmatrix} T_{11} & T_{11}Z - ZT_{22} + T_{12} \\ 0 & T_{22} \end{bmatrix} = \begin{bmatrix} T_{11} & 0 \\ 0 & T_{22} \end{bmatrix}. \quad \square \end{aligned}$$

**Пример 7.1.3.** Если

$$T = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & 8 \\ 0 & -2 & 3 \end{bmatrix} \quad \text{и} \quad Y = \begin{bmatrix} 1.0 & 0.5 & -0.5 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix},$$

то

$$Y^{-1}TY = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 8 \\ 0 & -2 & 3 \end{bmatrix}.$$

Применяя повторно лемму 7.1.5, можно установить более общий результат.

**Теорема 7.1.6 (Блочно-диагональное разложение).** Пусть

$$Q^H A Q = T = \begin{bmatrix} T_{11} & T_{12} & \dots & T_{1q} \\ 0 & T_{22} & \dots & T_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & T_{qq} \end{bmatrix} \quad (7.1.5)$$

есть разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$ , и предположим, что блоки  $T_{ii}$  квадратные. Если  $\lambda(T_{ii}) \cap \lambda(T_{jj}) = \emptyset$  при  $i \neq j$ , то существует невырожденная матрица  $Y \in \mathbb{C}^{n \times n}$ , такая, что

$$(QY)^{-1}A(QY) = \text{diag}(T_{11}, \dots, T_{qq}). \quad (7.1.6)$$

*Доказательство.* Доказательство можно выполнить, используя лемму 7.1.5 и индукцию.  $\square$

Если каждому диагональному блоку  $T_{ii}$  соответствует определенное собственное значение, то мы получаем

**Следствие 7.1.7.** *Если  $A \in \mathbb{C}^{n \times n}$ , то существует невырожденная матрица  $X$ , такая, что*

$$X^{-1}AX = \text{diag}(\lambda_1 I + N_1, \dots, \lambda_q I + N_q), \quad N_i \in \mathbb{C}^{n_i \times n_i}, \quad (7.1.7)$$

где  $\lambda_1, \dots, \lambda_q$  различные, целые  $n_1, \dots, n_q$  удовлетворяют равенству  $n_1 + \dots + n_q = n$  и каждая матрица  $N_i$  является строго верхней треугольной.

С разложением (7.1.7) связан ряд важных понятий. Целое число  $n_i$  называют алгебраической кратностью собственного значения  $\lambda_i$ . Если  $n_i = 1$ , то говорят, что собственное значение  $\lambda_i$  – простое. Геометрическая кратность  $\lambda_i$  равна размерности  $\text{null}(N_i)$ , т. е. числу независимых собственных векторов, соответствующих  $\lambda_i$ . Если алгебраическая кратность  $\lambda_i$  превышает его геометрическую кратность, то говорят, что  $\lambda_i$  – дефектное собственное значение. Матрицу, имеющую дефектное собственное значение, называют дефектной. Недефектные матрицы называют также диагонализуемыми в соответствии со следующим результатом:

**Следствие 7.1.8 (Диагональная форма).** *Матрица  $A \in \mathbb{C}^{n \times n}$  является недефектной тогда и только тогда, когда существует невырожденная матрица  $X \in \mathbb{C}^{n \times n}$ , такая, что*

$$X^{-1}AX = \text{diag}(\lambda_1, \dots, \lambda_n). \quad (7.1.8)$$

*Доказательство.* Матрица  $A$  недефектная тогда и только тогда, когда существуют линейно независимые векторы  $x_1, \dots, x_n \in \mathbb{C}^n$  и скаляры  $\lambda_1, \dots, \lambda_n$ , такие, что  $Ax_i = \lambda_i x_i$  при  $i = 1:n$ . Это эквивалентно существованию невырожденной матрицы  $X = [x_1, \dots, x_n] \in \mathbb{C}^{n \times n}$ , такой, что  $AX = XD$ , где  $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ .  $\square$

Заметим, что если  $y_i^H$  есть  $i$ -я строка матрицы  $X^{-1}$ , то  $y_i^H A = \lambda_i y_i^H$ . Таким образом, столбцы матрицы  $X^T$  являются левыми собственными векторами, а столбцы матрицы  $X$  – правыми.

**Пример 7.1.4.** Если

$$A = \begin{bmatrix} 5 & -1 \\ -2 & 6 \end{bmatrix} \text{ и } X = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix},$$

то  $X^{-1}AX = \text{diag}(4, 7)$ .

Если матрицу  $X$  в (7.1.7) разбить на блоки

$$X = [X_1, \dots, X_q],$$

то  $\mathbb{C}^n = \text{range}(X_1) \oplus \dots \oplus \text{range}(X_q)$  есть прямая сумма инвариантных подпространств. Если базисы в этих подпространствах выбрать специальным образом, то можно получить еще больше нулей в верхней треугольной части матрицы  $X^{-1}AX$ .

**Теорема 7.1.9 (Жорданово разложение).** *Если  $A \in \mathbb{C}^{n \times n}$ , то существует невырожденная матрица  $X \in \mathbb{C}^{n \times n}$ , такая, что  $X^{-1}AX = \text{diag}(J_1, \dots, J_r)$ , где*

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \cdots & 0 \\ 0 & \lambda_i & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & & \lambda_i \end{bmatrix}$$

есть  $m_i \times m_i$ -матрица и  $m_1 + \dots + m_t = n$ .

*Доказательство.* Смотри Halmos (1958), с. 112.  $\square$

Блоки  $J_i$  называют **жордановыми**. Число и размеры жордановых блоков, связанных с каждым собственным значением, определяются однозначно, хотя их расположение вдоль диагонали неоднозначно.

#### 7.1.4. Некоторые замечания о неунитарном подобии

Жорданову блочную структуру дефектной матрицы трудно определять численно. Множество диагонализуемых  $n \times n$ -матриц является плотным в  $\mathbb{C}^{n \times n}$ , и, таким образом, малые изменения в дефектной матрице могут значительно изменить ее жорданову форму. Более подробно мы поговорим об этом в разд. 7.6.

Трудности, возникающие в задаче нахождения собственных значений, состоят в том, что у почти дефектных матриц плохо обусловлены матрицы собственных векторов. Например, любая матрица  $X$ , диагонализующая матрицу

$$A = \begin{bmatrix} 1 + \varepsilon & 1 \\ 0 & 1 - \varepsilon \end{bmatrix}, \quad 0 < \varepsilon \ll 1, \quad (7.1.9)$$

имеет число обусловленности во 2-й норме порядка  $1/\varepsilon$ .

Эти наблюдения показывают трудности, связанные с плохой обусловленностью преобразования подобия. Поскольку

$$f(X^{-1}AX) = X^{-1}AX + E, \quad (7.1.10)$$

где

$$\|E\|_2 \approx \text{uk}_2(X) \|A\|_2, \quad (7.1.11)$$

ясно, что большая ошибка может появиться при вычислении собственного значения, когда мы выходим за рамки унитарных преобразований подобия.

#### Задачи

**7.1.1.** Показать, что если матрица  $T \in \mathbb{C}^{n \times n}$  является верхней треугольной и нормальной, то  $T$ -диагональная.

**7.1.2.** Проверить, что если  $X$  диагонализует  $2 \times 2$ -матрицу (7.1.9) и  $\varepsilon \leq 1/2$ , то  $k_1(X) \geq 1/\varepsilon$ .

**7.1.3.** Пусть собственные значения матрицы  $A \in \mathbb{C}^{n \times n}$  различны. Показать, что если  $Q^H A Q = T$  есть ее разложение Шура и  $AB = BA$ , то матрица  $Q^H B Q$  верхняя треугольная.

**7.1.4.** Показать, что если матрицы  $A$  и  $B^H$  принадлежат  $\mathbb{C}^{m \times n}$  с  $m \geq n$ , то:

$$\lambda(AB) = \lambda(BA) \cup \{0, \dots, 0\}.$$

**7.1.5.** Задана матрица  $A \in \mathbb{C}^{n \times n}$ . Используя разложение Шура, показать, что для каждого  $\varepsilon > 0$  существует диагонализуемая матрица  $B$ , такая, что  $\|A - B\|_2 \leq \varepsilon$ . Это показывает, что множество диагонализуемых матриц плотно в  $\mathbb{C}^{n \times n}$  и что каноническая форма Жордана не является непрерывным матричным разложением.

**7.1.6.** Предположим, что  $A_k \rightarrow A$  и  $Q_k^H A_k Q_k = T_k$  есть разложение Шура матрицы  $A_k$ . Показать, что последовательность  $\{Q_k\}$  имеет сходящуюся подпоследовательность  $\{Q_{k_i}\}$ , обладающую следующим свойством:

$$\lim_{i \rightarrow \infty} Q_{k_i} = Q,$$

где  $Q^H A Q = T$  — верхняя треугольная. Это показывает, что собственные векторы матрицы являются непрерывными функциями ее элементов.

**7.1.7.** Обосновать (7.1.10), (7.1.11).

### Замечания и литература к § 7.1

Математические свойства алгебраической проблемы собственных значений изящно рассмотрены в Wilkinson (AEP), гл. 1, и Stewart (IMC), гл. 6. Для тех, кому требуются дополнительные сведения, мы также рекомендуем

Bellman R. (1970). Introduction to Matrix Analysis, 2nd ed., McGraw-Hill, New York.

Gohberg I.C., Lancaster P. and Rodman L. (1986). Invariant Subspaces of Matrices With Applications, John Wiley and Sons, New York.

Marcus M. and Minc H. (1964). A Survey of Matrix Theory and Matrix Inequalities, Allyn and Bacon, Boston.

Mirsky L. (1963). An Introduction to Linear Algebra, Oxford University Press, Oxford

Разложение Шура впервые опубликовано в

Schur I. (1909). “On the Characteristic Roots of a Linear Substitution with an Application to the Theory of Integral Equations”, Math. Ann. 66, 488–510 (German).

Доказательство, очень похожее на наше, дано на стр. 105 работы

Turnbill H. W. and Aitken A. C. (1961). An Introduction to the Theory of Canonical Forms, Dover, New York.

Иногда можно установить особую структуру разложения Шура для матриц специальной структуры. Смотри

Paige C. C. and Van Loan C. (1981). “A Schur Decomposition for Hamiltonian Matrices”, Lin. Alg. and Its Applic. 41, 11–32.

## 7.2. Теория возмущения

Некакие из разложений, описанных в предыдущем разделе, не могут быть вычислены точно из-за ошибок округления и из-за того, что алгоритмы вычисления собственных значений являются итерационными и должны быть остановлены после некоторого конечного числа шагов. Поэтому важно разработать удобную теорию возмущений, чтобы руководствоваться ею в последующих разделах.

### 7.2.1. Чувствительность собственного значения

Ряд стандартных программ вычисления собственных значений выполняют последовательность преобразования подобия  $X_k$ , обладающих тем свойством, что матрицы  $X_k^{-1}AX_k$  становятся «более диагональными». Возникает, естественно, вопрос: «Насколько хорошо диагональные элементы матрицы аппроксимируют ее собственные значения?»

**Теорема 7.2.1 (О кругах Гершгорина).** Если  $X^{-1}AX = D + F$ , где  $D = \text{diag}(d_1, \dots, d_n)$  и  $F$  имеет нулевые диагональные элементы, то

$$\lambda(A) \subseteq \bigcup_{i=1}^n D_i,$$

где

$$D_i = \{z \in \mathbf{C} : |z - d_i| \leq \sum_{j=1}^n |f_{ij}| \}.$$

**Доказательство.** Пусть  $\lambda \in \lambda(A)$  и предположим, не нарушая общности, что  $\lambda \neq d_i$  для  $i = 1 : n$ . Так как матрица  $(D - \lambda I) + F$  – вырожденная, то из леммы 2.3.3 следует, что

$$1 \leq \| (D - \lambda I)^{-1} F \|_{\infty} = \sum_{j=1}^n \frac{|f_{kj}|}{|d_k - \lambda|}$$

для некоторого  $k, 1 \leq k \leq n$ . Но это не означает, что  $\lambda \in D_k$ .  $\square$

**Пример 7.2.1.** Если

$$A = \begin{bmatrix} 10 & 2 & 3 \\ -1 & 0 & 2 \\ 1 & -2 & 1 \end{bmatrix},$$

то  $\lambda(A) \approx \{10.226, 0.3870 + 2.2216i, 0.3870 - 2.2216i\}$ , а круги Гершгорина –  $D_1 = \{|z| : |z - 10| \leq 5\}$ ,  $D_2 = \{|z| : |z| \leq 3\}$  и  $D_3 = \{|z| : |z - 1| \leq 3\}$ .

Можно также показать, что если круг Гершгорина  $D_i$  изолирован от остальных кругов, то он содержит в точности одно собственное значение матрицы  $A$ . Смотри Wilkinson (АЕР), с. 71. Для некоторых очень важных процедур вычисления собственных значений можно показать, что вычисленные собственные значения являются точными собственными значениями некоторой матрицы  $A + E$ , где матрица  $E$  мала по норме. Следовательно, мы должны понять, как малые возмущения влияют на собственные значения матрицы. Простым результатом, проливающим свет на этот вопрос, является следующая теорема.

**Теорема 7.2.2 (Баэр–Файк).** Если  $\mu$  является собственным значением матрицы  $A + E \in \mathbf{C}^{n \times n}$  и  $X^{-1}AX = D = \text{diag}(\lambda_1, \dots, \lambda_n)$ , то

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq k_p(X) \|E\|_p,$$

где  $\|\cdot\|_p$  означает любую из  $p$ -норм.

**Доказательство.** Достаточно рассмотреть лишь тот случай, когда  $\mu$  не принадлежит  $\lambda(A)$ . Если матрица  $X^{-1}(A + E - \mu I)X$  вырожденная, то матрица  $I + (D - \mu I)^{-1}(X^{-1}EX)$  также вырожденная. Поэтому из леммы 2.3.3 получаем, что

$$\begin{aligned} 1 &\leq \| (D - \mu I)^{-1}(X^{-1}EX) \|_p \leq \\ &\leq \| (D - \mu I)^{-1} \|_p \| X \|_p \| E \|_p \| X^{-1} \|_p. \end{aligned}$$

Так как матрица  $(D - \mu I)^{-1}$  диагональная, а  $p$ -норма диагональной матрицы есть максимальное из абсолютных значений ее диагональных элементов, то

$$\| (D - \mu I)^{-1} \|_p = \min_{\lambda \in \lambda(A)} \frac{1}{|\lambda - \mu|},$$

что и доказывает теорему.  $\square$

Аналогичный результат может быть получен через разложение Шура:

**Теорема 7.2.3.** Пусть  $Q^H A Q = D + N$  – разложение Шура (7.1.3) матрицы  $A \in \mathbb{C}^{n \times n}$ . Если  $\mu \in \lambda(A + E)$  и  $p$  – наименьшее положительное целое, такое, что  $|N|^p = 0$ , то

$$\min_{\lambda \in \lambda(A)} |\lambda - \mu| \leq \max(\theta, \theta^{1/p}),$$

где

$$\theta = \|E\|_2 \sum_{k=0}^{p-1} \|N\|_2^k.$$

*Доказательство.* Обозначим

$$\delta = \min_{\lambda \in \lambda(A)} |\lambda - \mu| = \frac{1}{\|(\mu I - D)^{-1}\|_2}.$$

Теорема очевидно верна, если  $\delta = 0$ . Если  $\delta > 0$ , то матрица  $I - (\mu I - A)^{-1}$  вырожденная, и, применяя лемму 2.3.3, имеем:

$$\begin{aligned} 1 &\leq \|(\mu I - A)^{-1} E\|_2 \leq \|(\mu I - A)^{-1}\|_2 \|E\|_2 = \\ &= \|((\mu I - D) - N)^{-1}\|_2 \|E\|_2. \end{aligned} \quad (7.2.1)$$

Учитывая, что матрица  $(\mu I - D)^{-1}$  диагональная и  $|N|^p = 0$ , несложно показать, что  $((\mu I - D)^{-1} N)^p = 0$ . Поэтому

$$((\mu I - D) - N)^{-1} = \sum_{k=0}^{p-1} ((\mu I - D)^{-1} N)^k (\mu I - D)^{-1}$$

и, следовательно,

$$\|((\mu I - D) - N)^{-1}\|_2 \leq \frac{1}{\delta} \sum_{k=0}^{p-1} \left( \frac{\|N\|_2}{\delta} \right)^k.$$

Если  $\delta > 1$ , то

$$\|((\mu I - D) - N)^{-1}\|_2 \leq \frac{1}{\delta} \sum_{k=0}^{p-1} \|N\|_2^k$$

и из (7.2.1) следует, что  $\delta \leq \theta$ . Если  $\delta \leq 1$ , то

$$\|((\mu I - D) - N)^{-1}\|_2 \leq \frac{1}{\delta^p} \sum_{k=0}^{p-1} \|N\|_2^k$$

и из (7.2.1) следует, что  $\delta^p \leq \theta$ . Таким образом,  $\delta \leq \max(\theta, \theta^{1/p})$ .  $\square$

**Пример 7.2.2.** Если

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 4.001 \end{bmatrix} \text{ и } E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.001 & 0 & 0 \end{bmatrix},$$

то  $\lambda(A + E) \approx \{1.0001, 4.0582, 3.9427\}$  и матрица собственных векторов матрицы  $A$  удовлетворяет условию  $k_2(X) \approx 10^7$ . Оценка Баэзра–Файка из теоремы 7.2.2 имеет порядок  $10^4$ , в то время как оценка Шура имеет порядок  $10^0$ .

Каждая из теорем 7.2.2 и 7.2.3 указывает возможную чувствительность собственного значения, если матрица  $A$  не является нормальной. В частности, если величина  $k_2(X)$

или  $\|N\|_2^{p-1}$  велика, то малое возмущение элементов матрицы  $A$  может вызвать большие возмущения ее собственных значений.

**Пример 7.2.3.** Если

$$A = \begin{bmatrix} 0 & I_9 \\ 0 & 0 \end{bmatrix} \text{ и } E = \begin{bmatrix} 0 & 0 \\ 10^{-10} & 0 \end{bmatrix},$$

то для всех  $\lambda \in \lambda(A)$  и  $\mu \in \lambda(A + E)$  величина  $|\lambda - \mu| = 10^{-1}$ . В этом примере возмущение порядка  $10^{-10}$  матрицы  $A$  приводит к возмущению порядка  $10^{-1}$  ее собственных значений.

## 7.2.2. Обусловленность простого собственного значения

Высокая чувствительность собственного значения матрицы  $A$  не может иметь места, если матрица  $A$  – нормальная. С другой стороны, «ненормальность» не обязательно подразумевает высокую чувствительность собственного значения. Действительно, матрица, не являющаяся нормальной, может иметь как хорошо обусловленные, так и плохо обусловленные собственные значения. Следовательно, полезно уточнить нашу теорию возмущения таким образом, чтобы ее можно было применять к отдельным собственным значениям, а не к спектру в целом.

Для этого предположим, что  $\lambda$  – простое собственное значение матрицы  $A \in \mathbb{C}^{n \times n}$  и что векторы  $x$  и  $y$  удовлетворяют равенствам  $Ax = \lambda x$  и  $y^H A = \lambda y^H$ , причем  $\|x\|_2 = \|y\|_2 = 1$ . Используя классические результаты теории функций можно показать, что в некоторой окрестности точки  $\varepsilon = 0$  существуют дифференцируемые функции  $x(\varepsilon)$  и  $\lambda(\varepsilon)$ , такие, что

$$(A + \varepsilon F)x(\varepsilon) = \lambda(\varepsilon)x(\varepsilon) \quad \|F\|_2 = 1, \|x(\varepsilon)\|_2 \equiv 1,$$

где  $\lambda(0) = \lambda$  и  $x(0) = x$ . Дифференцируя это уравнение по  $\varepsilon$  и полагая затем  $\varepsilon = 0$ , получим следующее уравнение:

$$A \dot{x}(0) + Fx = \dot{\lambda}(0)x + \lambda \dot{x}(0).$$

Умножив на  $y^H$  обе части этого уравнения, разделив их на  $y^H x$  и взяв по абсолютной величине, получим, что

$$|\dot{\lambda}(0)| = \left| \frac{y^H F x}{y^H x} \right| \leq \frac{1}{|y^H x|}.$$

Верхняя граница достигается, если  $F = yx^H$ . Поэтому мы будем называть величину, обратную

$$s(\lambda) = |y^H x|,$$

обусловленностью собственного значения  $\lambda$ .

Грубо говоря, выполненный выше анализ показывает, что если матрицу  $A$  изменить на величину порядка  $\varepsilon$ , то собственное значение  $\lambda$  может измениться на величину порядка  $\varepsilon/s(\lambda)$ . Следовательно, если  $s(\lambda)$  – мало, то собственное значение  $\lambda$  следует рассматривать как плохо обусловленное. Заметим, что  $s(\lambda)$  является косинусом угла между левым и правым собственными векторами, соответствующими  $\lambda$ , и равно единице только в том случае, когда  $\lambda$  – простое.

Малое  $s(\lambda)$  означает, что матрица  $A$  близка к матрице, имеющей кратное собственное значение. Точнее, если  $\lambda$  – изолированное собственное значение и  $s(\lambda) < 1$ , то найдется такая матрица  $E$ , что  $\lambda$  будет кратным собственным значением

матрицы  $A + E$  и

$$\frac{\|E\|_2}{\|A\|_2} \leq \frac{s(\lambda)}{\sqrt{1 - s(\lambda)^2}}.$$

Этот результат доказан в Wilkinson (1972).

**Пример 7.2.4.** Если

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 4.001 \end{bmatrix} \text{ и } E = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.001 & 0 & 0 \end{bmatrix},$$

то  $\lambda(A + E) \approx \{1.0001, 4.0582, 3.9427\}$ , а  $s(1) \approx 0.79 \times 10^0$ ,  $s(4) \approx 0.16 \times 10^{-3}$  и  $s(4.001) \approx 0.16 \times 10^{-3}$ . Заметим, что  $\|E\|_2/s(\lambda)$  является хорошей оценкой возмущения, которому подверглось каждое из собственных значений.

### 7.2.3. Чувствительность кратных собственных значений

Если  $\lambda$  – кратное собственное значение, то вопрос о чувствительности этого собственного значения более сложен. Например, если

$$A = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \text{ и } F = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$

то  $\lambda(A + \varepsilon F) = \{1 \pm \sqrt{\varepsilon} a\}$ . Заметим, что если  $a \neq 0$ , то собственные значения матрицы  $A + \varepsilon F$  не дифференцируемы в нуле; скорость их изменения в окрестности точки  $\varepsilon = 0$  бесконечна. Вообще говоря, если  $\lambda$  – дефектное собственное значение матрицы  $A$ , то возмущения порядка  $O(\varepsilon)$  матрицы  $A$  могут привести к возмущениям порядка  $O(\varepsilon^{1/p})$  в  $\lambda$ , если  $\lambda$  соответствует жорданову блоку порядка  $p$ . Более подробно это обсуждается в Wilkinson (AEP), с. 77.

### 7.2.4. Чувствительность собственного вектора

Сейчас мы посмотрим, как малые изменения матрицы влияют на ее инвариантные подпространства. Рассмотрим сначала случай собственных векторов. Предположим, что матрица  $A \in \mathbb{C}^{n \times n}$  имеет попарно различные собственные значения  $\lambda_1, \dots, \lambda_n$  и матрица  $F \in \mathbb{C}^{n \times n}$  удовлетворяет условию  $\|F\|_2 = 1$ . Из соображений непрерывности следует, что для всех  $\varepsilon$  в некоторой окрестности точки  $\varepsilon = 0$  имеем:

$$(A + \varepsilon F)x_k(\varepsilon) = \lambda_k(\varepsilon)x_k(\varepsilon), \quad \|x_k(\varepsilon)\|_2 \equiv 1,$$

$$y_k(\varepsilon)^H(A + \varepsilon F) = \lambda_k(\varepsilon)y_k(\varepsilon)^H, \quad \|y_k(\varepsilon)\|_2 \equiv 1, \quad k = 1 : n,$$

где каждая из функций  $\lambda_k(\varepsilon)$ ,  $x_k(\varepsilon)$  и  $y_k(\varepsilon)$  дифференцируема. Если мы про-дифференцируем первое уравнение по  $\varepsilon$  и положим  $\varepsilon = 0$ , то получим следующее равенство:

$$A \dot{x}_k(0) + F x_k = \lambda_k(0)x_k + \lambda_k \dot{x}_k(0),$$

где  $\lambda_k = \lambda_k(0)$  и  $x_k = x_k(0)$ . Так как собственные векторы матрицы  $A$  образуют базис, мы можем записать

$$\dot{x}_k(0) = \sum_{i=1}^n a_i x_i$$

и, следовательно,

$$\sum_{\substack{i=1 \\ i \neq k}}^n a_i (\lambda_i - \lambda_k) x_i + F x_k = \lambda_k(0) x_k.$$

Для того чтобы получить выражения для  $a_i$ , заметим, что  $y_i(0)^H x_k \equiv y_i^H x_k = 0$  при всех  $i \neq k$  и, следовательно,

$$a_i = \frac{y_i^H F x_k}{(\lambda_k - \lambda_i) y_i^H x_i}, \quad i \neq k.$$

Поэтому ряд Тейлора для  $x_k(\varepsilon)$  имеет следующий вид:

$$x_k(\varepsilon) = x_k + \sum_{\substack{i=1 \\ i \neq k}}^n \frac{y_i^H F x_k}{(\lambda_k - \lambda_i) y_i^H x_i} x_i + O(\varepsilon^2).$$

Таким образом, чувствительность собственного вектора  $x_k$  зависит от чувствительности собственного значения  $\lambda_k$  и удаленности  $\lambda_k$  от остальных собственных значений.

То, что удаленность собственных значений влияет на чувствительность собственного вектора, не должно вызывать удивления. Действительно, если  $\lambda$  – не дефектное кратное собственное значение, то существует бесконечное число возможных базисов в соответствующем ему инвариантном подпространстве. Предыдущий анализ просто показал, что эта неопределенность начинает чувствоватьться, когда собственные значения сливаются. Другими словами, собственные векторы, соответствующие близким собственным значениям являются «шаткими».

**Пример 7.2.5.** Если

$$A = \begin{bmatrix} 1.01 & 0.001 \\ 0.00 & 0.99 \end{bmatrix},$$

то собственное значение  $\lambda = 0.99$  имеет обусловленность  $1/s(0.99) \approx 1.118$  и собственный вектор  $x = (0.4472, -0.8944)$ . С другой стороны, собственное значение  $\hat{\lambda} = 1.00$  «близкой» матрицы

$$A + E = \begin{bmatrix} 1.001 & 0.01 \\ 0.00 & 1.00 \end{bmatrix}$$

имеет собственный вектор  $\hat{x} = (0.7071, -7.071)^T$ .

### 7.2.5. Чувствительность инвариантного подпространства

Набор чувствительных собственных векторов может составить нечувствительное инвариантное подпространство при условии, что соответствующий кластер собственных значений изолирован. Для определенности предположим, что

$$Q^H A Q = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix} \quad (7.2.2)$$

– разложение Шура матрицы  $A$  с матрицей

$$Q = [Q_1 \quad Q_2]. \quad (7.2.3)$$

Из нашего анализа чувствительности собственного вектора ясно, что чувствительность инвариантного подпространства  $\text{range}(Q_1)$  зависит от расстояния между  $\lambda(T_{11})$  и  $\lambda(T_{22})$ . Правильной мерой этого расстояния, оказывается, является наименьшее сингулярное число линейного преобразования  $X \rightarrow T_{11}X - X T_{22}$ . (Напомним, что это преобразование фигурирует в лемме 7.1.5.) Более точно: если мы определим *отделенность* матриц  $T_{11}$  и  $T_{22}$  как

$$\text{sep}(T_{11}, T_{22}) = \min_{X \neq 0} \frac{\|T_{11}X - X T_{22}\|_F}{\|X\|_F},$$

то получим следующий общий результат.

**Теорема 7.2.4.** *Предположим, что имеют место равенства (7.2.2) и (7.2.3) и что для любой матрицы  $E \in \mathbb{C}^{n \times n}$  мы разбиваем матрицу  $Q^H E Q$  на блоки следующим образом*

$$Q^H E Q = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix} \begin{array}{c} p \\ n-p \end{array}.$$

Если  $\delta = \text{sep}(T_{11}, T_{22}) - \|E_{11}\|_2 - \|E_{22}\|_2 > 0$  и

$$\|E_{21}\|(\|T_{12}\|_2 + \|E_{21}\|_2) \leq \delta^2/4,$$

то существует матрица  $P \in \mathbb{C}^{(n-p) \times p}$ , такая, что

$$\|P\|_2 \leq 2\|E_{21}\|_2/\delta,$$

а столбцы матрицы  $\hat{Q} = (Q_1 + Q_2 P)(I + P^H P)^{-1/2}$  образуют ортонормированный базис в некотором инвариантном подпространстве матрицы  $A + E$ .

*Доказательство.* Смотри Stewart (1973б) теорема 4.1.1.  $\square$

Так как  $Q_1^H \hat{Q}_1 = (I + P^H P)^{-1/2}$ , то сингулярные значения матрицы  $P$  являются тангенсами главных углов между  $\text{range}(\hat{Q}_1)$  и  $\text{range}(Q_1)$ . (См. § 12.4.3.) Теорема 7.2.4 показывает, что эти тангенсы по существу ограничены величиной  $\|E\|_2 / \text{sep}(T_{11}, T_{22})$ . Таким образом, величину, обратную  $\text{sep}(T_{11}, T_{22})$ , можно трактовать как число обусловленности, определяющее чувствительность  $\text{range}(Q_1)$  как инвариантного подпространства.

**Пример 7.2.6.** Пусть

$$T_{11} = \begin{bmatrix} 3 & 10 \\ 0 & 1 \end{bmatrix}, \quad T_{22} = \begin{bmatrix} 0 & -20 \\ 0 & 3.01 \end{bmatrix} \quad \text{и} \quad T_{12} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$$

$$A = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}.$$

Заметим, что  $AQ_1 = Q_1 T_{11}$ , где  $Q_1 = [e_1, e_2] \in \mathbb{R}^{4 \times 2}$ . Вычисления показывают, что  $\text{sep}(T_{11}, T_{22}) \approx 0.0003$ . Если мы положим

$$E_{21} = 10^{-6} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

и исследуем разложение Шура матрицы

$$A + E = \begin{bmatrix} T_{11} & T_{12} \\ E_{21} & T_{22} \end{bmatrix},$$

то обнаружим, что матрица  $\hat{Q}_1$  стала равна

$$\hat{Q}_1 = \begin{bmatrix} -0.9999 & -0.0003 \\ 0.0003 & -0.9999 \\ -0.0005 & -0.0026 \\ 0.0000 & 0.0003 \end{bmatrix}.$$

Таким образом, имеем  $\text{dist}(\hat{Q}_1, Q_1) \approx 0.0027 \approx 10^{-6}/\text{sep}(T_{11}, T_{22})$ .

### Задачи

**7.2.1.** Предположим, что  $Q^H A Q = \text{diag}(\lambda_i) + N$  – разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$ , и обозначим  $v(A) = \|A^H A - AA^H\|_F$ . Верхняя и нижняя границы

$$\frac{v(A)^2}{6\|A\|_F^2} \leq \|N\|_F^2 \leq \sqrt{\frac{n^3 - n}{12}} v(A)$$

были установлены Хенричи (Henrici (1962)) и Эберлейном (Eberlein (1965)) соответственно. Проверить эти результаты для случая  $n = 2$ .

**7.2.2.** Предположим, что матрица  $A \in \mathbb{C}^{n \times n}$  имеет различные собственные значения и

$$X^{-1}AX = \text{diag}(\lambda_1, \dots, \lambda_n).$$

Показать, что если столбцы матрицы  $X$  имеют единичную вторую норму, то

$$k_F(X) = \sqrt{n \sum_{i=1}^n \frac{1}{s(\lambda_i)^2}}.$$

**7.2.3.** Пусть  $Q^H A Q = \text{diag}(\lambda_i) + N$  – разложение Шура матрицы  $A$  и  $X^{-1}AX = \text{diag}(\lambda_i)$ . Показать, что

$$\kappa_2(X) \geq \sqrt{1 + \frac{\|N\|_F^2}{\|A\|_F^2}}.$$

Смотри Loizou (1969).

**7.2.4.** Если  $X^{-1}AX = \text{diag}(\lambda_i)$  и  $|\lambda_1| \geq \dots \geq |\lambda_n|$ , то

$$\frac{\sigma_i(A)}{\kappa_2(X)} \leq |\lambda_i| \leq \kappa_2(X) \sigma_i(A).$$

Доказать этот результат для случая  $n = 2$  (Ruhe (1975)).

**7.2.5.** Показать, что если  $A = \begin{bmatrix} a & c \\ 0 & b \end{bmatrix}$ , то  $s(a) = s(b) = (1 + |c/(a-b)|^2)^{-1}$ .

**7.2.6.** Показать, что обусловленность собственного значения сохраняется при унитарном преобразовании подобия.

### Замечания и литература к § 7.2

Многие результаты, представленные в этом разделе, можно найти в Wilkinson (АЕР), гл. 2, так же, как и в

Householder A. S. (1964). The Theory of Matrices in Numerical Analysis. Blaisdell, New York.

Bauer F. L., Fike C. T. (1960). "Norms and Exclusion Theorems", Numer. Math. 2, 123–144.

Следующие статьи посвящены действию возмущений на собственные значения матрицы общего вида

Kahan W., Parlett B. N. and Jiang E. (1982). "Residual Bounds on Approximate Eigensystems of Nonnormal Matrices", SIAM J. Numer. Anal. 19, 470–484.

- Ruhe A. (1970). "Perturbation Bounds for Means of Eigenvalues and Invariant Subspaces", BIT 10, 343–354.
- Ruhe A. (1970). "Properties of a Matrix with a Very Ill-Conditioned Eigenproblem", Numer. Math. 15, 57–60.
- Wilkinson J. H. (1972). "Note on Matrices with a Very Ill-Conditioned Eigenproblem", Numer. Math. 19, 176–178.
- Wilkinson J. H. (1984). "On Neighboring Matrices with Quadratic Elementary Divisors", Numer. Math. 44, 1–21.

Работа Уилкинсона о ближайших дефектных матрицах является типичным примером разрастающейся литературы, посвященной проблеме «близости» матриц. Смотри

- Byers R. (1988). "A Bisection Method for Measuring the Distance of a Stable Matrix to the Unstable Matrices", SIAM J. Sci. and Stat. Comp. 9, 875–881.
- Demmel J. W. (1987). "On the Distance to the Nearest Ill-Posed Problem", Numer. Math. 51, 251–289.
- Demmel J. W. (1987). "A Counterexample for two Conjectures About Stability", IEEE Trans. Auto. Cont. AC-32, 340–342.
- Demmel J. W. (1988). "The Probability that a Numerical Analysis Problem is Difficult", Math. Comp. 50, 449–480.
- Higham N. J. (1985). "Nearness Problems in Numerical Linear Algebra", PhD Thesis, University of Manchester, England.
- Higham N. J. (1988). "Matrix Nearness Problems and Applications", Numerical Analysis Report 161, University of Manchester, England. (To appear in the Proceedings of the IMA Conference on Applications of Matrix Theory, S. Barnett and M. J. C. Gover (eds), Oxford University Press.)
- Ruhe A. (1987). "Closest Normal Matrix Found", BII 27, 585–598.
- Van Loan C. (1985). "How Near is a Stable Matrix to an Unstable Matrix?", Contemporary Mathematics, Vol. 47, 465–477.

Взаимосвязь между числом обусловленности собственного значения, отклонением от нормальности и обусловленностью матрицы собственных векторов обсуждается в

- Eberlein P. (1965). "On Measures of Non-Normality for Matrices", Amer. Math. Soc. Monthly 72, 995–996.
- Henrici P. (1962). "Bounds for Iterates, Inverses, Spectral Variation and Fields of Values of Non-normal Matrices", Numer. Math. 4, 24–40.
- Loizou G. (1969). "Nonnormality and Jordan Condition Numbers of Matrices", J. ACM 16, 580–540.
- Smith R. A. (1967). "The Condition Numbers of the Matrix Eigenvalue Problem", Numer. Math. 10, 232–240.
- van der Sluis A. (1975). "Perturbations of Eigenvalues of Non-normal Matrices", Comm. ACM 18, 30–36.

Статья Хенричи также содержит результат, похожий на теорему 7.2.3. Глубокие исследования возмущений инвариантных подпространств включают

- Davis C. and Kahan W. M. (1970). "The Rotation of Eigenvectors by a Perturbation, III", SIAM J. Num. Anal. 7, 1–46.
- Stewart G. W. (1971). "Error Bounds for Approximate Invariant Subspace of Closed Linear Operators", SIAM J. Num. Anal. 8, 796–808.
- Stewart G. W. (1973b). "Error and Perturbation Bounds for Subspaces Associated with Certain Eigenvalue Problems", SIAM Review 15, 727–764.

Детальный анализ функции  $\text{sep}(.,.)$  и преобразования  $X \rightarrow AX + XA^T$  дан в

- Byers R. and Nash S. G. (1987). "On the Singular Vectors of the Lyapunov Operator", SIAM J. Alg. and Disc. Methods 8, 59–66.

Varah J. (1979). "On the Separation of Two Matrices", SIAM J. Num. Anal. 16, 216–222.

Теорему Гершгорина можно использовать для получения исчерпывающей теории возмущений. Смотри Wilkinson (AEP), гл. 2. Эта теорема может быть обобщена и расширена различными способами; см.

- Johnston R. J. (1971). "Gershgorin Theorems for Partitioned Matrices", Lin. Alg. and Its Applic. 4, 205–220.

Varga R.S. (1970). "Minimal Gershgorin Sets for Partitioned Matrices", SIAM J. Num. Anal. 7, 493–507.

Наконец, мы упомянем классическую монографию

Kato T. (1966). Perturbation Theory for Linear Operators, Springer-Verlag, New York.

Глава 2 этой работы содержит исчерпывающее рассмотрение конечномерного случая.

## 7.3. Степенные итерации

Пусть заданы матрица  $A \in \mathbb{C}^{n \times n}$  и унитарная матрица  $U_0 \in \mathbb{C}^{n \times n}$ . Предположим, что ортогонализация Хаусхолдера (алгоритм 5.2.1) может быть распространена на комплексные матрицы (это возможно), и рассмотрим следующий итерационный процесс

$$\begin{aligned} T_0 &= U_0^H A U_0 \\ \text{for } k &= 1, 2, \dots \\ T_{k-1} &= U_k R_k \quad (\text{QR-разложение}) \\ T_k &= R_k U_k \\ \text{end} \end{aligned} \tag{7.3.1}$$

Так как  $T_k = R_k U_k = U_k^T (U_k R_k) U_k = U_k^T T_{k-1} U_k$ , то по индукции

$$T_k = (U_0 U_1 \dots U_k)^H A (U_0 U_1 \dots U_k). \tag{7.3.2}$$

Следовательно, каждая матрица  $T_k$  унитарно подобна матрице  $A$ . Не так заметно, но именно это центральная тема данного раздела, что матрицы  $T_k$  почти всегда сходятся к верхней треугольной форме. То есть (7.3.2) почти всегда «сходится» к разложению Шура матрицы  $A$ .

Процесс (7.3.1) называют *QR-итерациями*, и он составляет основу наиболее эффективного алгоритма вычисления разложения Шура. Для того чтобы обосновать этот метод и установить характер его сходимости, рассмотрим сначала два других итерационных процесса, представляющих самостоятельный интерес.

### 7.3.1. Степенной метод

Пусть матрица  $A \in \mathbb{C}^{n \times n}$  диагонализуема,  $X^{-1}AX = \text{diag}(\lambda_1, \dots, \lambda_n)$ , где  $X = [x_1, \dots, x_n]$  и  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$ . Для заданного вектора  $q^{(0)} \in \mathbb{C}^n$ , имеющего единичную 2-норму, *степенной метод* генерирует последовательность векторов  $q^{(k)}$  следующим образом:

$$\begin{aligned} \text{for } k &= 1, 2, \dots \\ z^{(k)} &= A q^{(k-1)} \\ q^{(k)} &= z^{(k)} / \|z^{(k)}\|_2 \\ \lambda^{(k)} &= [q^{(k)}]^H A q^{(k)} \\ \text{end} \end{aligned} \tag{7.3.3}$$

Нет никакой другой причины нормализовать по второй норме, кроме той, что это придает большее единство всему обсуждаемому в данном разделе.

Рассмотрим свойства сходимости степенных итераций. Если

$$q^{(0)} = a_1 x_1 + a_2 x_2 + \dots + a_n x_n$$

и  $a_1 \neq 0$ , то

$$A^k q^{(0)} = a_1 \lambda_1^k \left( x_1 + \sum_{j=2}^n \frac{a_j}{a_1} \left( \frac{\lambda_j}{\lambda_1} \right)^k x_j \right).$$

Так как  $q^{(k)} \in \text{span}\{A^k q^{(0)}\}$ , можно заключить, что

$$\text{dist}(\text{span}\{q^{(k)}\}, \text{span}\{x_1\}) = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right)$$

и, кроме того,

$$|\lambda_1 - \lambda^{(k)}| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

Если  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|$ , то говорят, что  $\lambda_1$  является *доминирующим собственным значением*. Итак, степенной метод сходится, если  $\lambda_1$  доминирует и если  $q^{(0)}$  имеет компоненту в направлении, соответствующем *доминирующему собственному вектору*  $x_1$ .

Поведение степенного метода без этих предположений рассматривается в Wilkinson (APE), с. 570, и Parlett, Poole (1973).

**Пример 7.3.1.** Если

$$A = \begin{bmatrix} -261 & 209 & -49 \\ -530 & 422 & -98 \\ -800 & 631 & -144 \end{bmatrix},$$

то  $\lambda(A) = \{10, 4, 3\}$ . Применяя (7.3.3) с  $q^{(0)} = (1, 0, 0)^T$ , находим:

$k$	$\lambda^{(k)}$
1	994.4900
2	13.0606
3	10.7191
4	10.2073
5	10.0633
6	10.0198
7	10.0063
8	10.0020
9	10.0007
10	10.0002

На практике эффективность степенного метода зависит от отношения  $|\lambda_2|/|\lambda_1|$ , так как оно определяет скорость сходимости. Опасность, что  $q^{(0)}$  не содержит  $x_1$ , не повод для беспокойства, поскольку ошибки округления, появляющиеся при выполнении итераций, обычно приводят к тому, что последующие  $q^{(k)}$  имеют компоненту в этом направлении. Более того, типичен случай, когда в приложениях, где требуется доминирующее собственное значение и собственный вектор, известно априорное приближение вектора  $x_1$ . Обычно, полагая  $q^{(0)}$  равным этому приближению, уменьшают опасность малого  $a_1$ .

Заметим, что для реализации степенного метода требуется только программа, способная вычислять матрично-векторное произведение вида  $Aq$ . Нет необходимости хранить матрицу  $A$  в массиве размера  $n \times n$ . По этой причине степенной метод может представлять интерес, когда матрица  $A$  – большая и разреженная и когда существует достаточный зазор между  $|\lambda_1|$  и  $|\lambda_2|$ .

Оценки погрешности  $|\lambda^{(k)} - \lambda_1|$  можно получить, используя теорию возмущения, развитую в предыдущем параграфе. Определим вектор  $r^{(k)} = Aq^{(k)} - \lambda^{(k)}q^{(k)}$  и заметим, что  $(A + E^{(k)})q^{(k)} = \lambda^{(k)}q^{(k)}$ , где  $E^{(k)} = -r^{(k)}[q^{(k)}]^H$ . Следовательно,

$\lambda^{(k)}$  – собственное значение  $A + E^{(k)}$  и

$$|\lambda^{(k)} - \lambda| \approx \frac{\|E^{(k)}\|_2}{s(\lambda_1)} = \frac{\|r^{(k)}\|_2}{s(\lambda_1)}.$$

Если мы используем степенной метод для получения правого и левого доминирующих собственных векторов, то можно получить оценку  $s(\lambda_1)$ . В частности, если  $w^{(k)}$  – вектор с единичной 2-нормой, направленный вдоль  $(A^H)^k w^{(0)}$ , то мы можем использовать следующее приближенное равенство  $s(\lambda_1) \approx |w^{(k)H} q^{(k)}|$ .

### 7.3.2. Ортогональные итерации

Прямое обобщение степенного метода может быть использовано для определения инвариантных подпространств большой размерности. Пусть  $p$  – выбранное некоторым образом целое число, удовлетворяющее неравенствам  $1 \leq p \leq n$ . Для заданной  $n \times n$ -матрицы  $Q_0$  с ортонормированными столбцами метод ортогональных итераций генерирует последовательность матриц  $\{Q_k\} \subseteq \mathbb{C}^{n \times p}$  следующим образом:

```
for k = 1, 2, ...
    Z_k = A Q_{k-1}
    Q_k R_k = Z_k      (QR-разложение)
end
```

(7.3.4)

Заметим, что если  $p = 1$ , то (7.3.4) – просто степенной метод. Более того, последовательность  $\{Q_k e_1\}$  – это в точности последовательность векторов, полученных степенными итерациями с начальным вектором  $q^{(0)} = Q_0 e_1$ .

Для анализа ортогональных итераций предположим, что

$$Q^H A Q = T = \text{diag}(\lambda_i) + N, \quad |\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|, \quad (7.3.5)$$

есть разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$ . Предположим, что  $1 \leq p < n$  и разобьем матрицы  $Q$ ,  $T$  и  $N$  на блоки следующим образом:

$$\begin{aligned} Q &= [Q_\alpha \quad Q_\beta], & T &= \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}, & p & \\ &\quad p \quad n-p & & p & n-p, \\ N &= \begin{bmatrix} N_{11} & N_{12} \\ 0 & N_{22} \end{bmatrix}, & p & \\ & & p & n-p. \end{aligned} \quad (7.3.6)$$

Если  $|\lambda_p| > |\lambda_{p+1}|$ , то подпространство  $D_p(A) = \text{range}(Q_\alpha)$  называют доминирующим инвариантным подпространством. Это единственный инвариантное подпространство, соответствующее собственным значениям  $\lambda_1, \dots, \lambda_p$ . Следующая теорема показывает, что при разумных предположениях подпространства  $\text{range}(Q_k)$ , генерируемые методом (7.3.4), сходятся к  $D_p(A)$  со скоростью, пропорциональной  $|\lambda_{p+1}/\lambda_p|^k$ .

**Теорема 7.3.1.** Пусть разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$  задано в виде (7.3.5) и (7.3.6) с  $n \geq 2$ . Предположим, что  $|\lambda_p| > |\lambda_{p+1}|$  и что  $\theta \geq 0$  удовлетворяет неравенству

$$(1 + \theta)|\lambda_p| > \|N\|_F.$$

Если матрица  $Q_0 \in \mathbb{C}^{n \times p}$  имеет ортонормированные столбцы и

$$d = \text{dist}(D_p(A^H), \text{range}(Q_0)) < 1,$$

то матрицы  $Q_k$ , генерируемые методом (7.3.4), удовлетворяют неравенству

$$\text{dist}(D_p(A), \text{range}(Q_k)) \leq$$

$$\leq \frac{(1 + \theta)^{n-2}}{\sqrt{1 - d^2}} \left( 1 + \frac{\|T_{12}\|_F}{\text{sep}(T_{11}, T_{22})} \right) \left( \frac{|\lambda_{p+1}| + \|N\|_F/(1 + \theta)}{|\lambda_p| - \|N\|_F/(1 + \theta)} \right)^k.$$

*Доказательство.* Доказательство длинное и представлено в конце раздела.  $\square$

Условие  $d < 1$  в теореме 7.3.1 гарантирует, что начальная матрица  $Q$  содержит все собственные направления:

$$d < 1 \leftrightarrow D_p(A^H)^\perp \cap \text{range}(Q_0) = \{0\}.$$

Теорема 7.3.1 в действительности говорит, что если это условие выполнено и если  $\theta$  выбрано достаточно большим, то

$$\text{dist}(D_p(A), \text{range}(Q_k)) \leq c \left| \frac{\lambda_{p+1}}{\lambda_p} \right|^k,$$

где константа  $c$  зависит от  $\text{sep}(T_{11}, T_{22})$  и отклонения от нормальности матрицы  $A$ . Очевидно, что сходимость может быть очень медленной, если зазор между  $|\lambda_p|$  и  $|\lambda_{p+1}|$  недостаточно широкий.

**Пример 7.3.2.** Если (7.3.4) применить к матрице  $A$  из примера 7.3.1 с  $Q_0 = [e_1, e_2]$ , то

$k$	$\text{dist}(D_2(A), \text{range}(Q_k))$
1	.0052
2	.0047
3	.0039
4	.0030
5	.0023
6	.0017
7	.0013

Погрешность стремится к нулю со скоростью  $(\lambda_3/\lambda_2)^k = (3/4)^k$ .

Можно ускорить сходимость ортогональных итераций, используя прием, описанный Стюартом [Stewart(1976)]. В ускоренной схеме приближенное собственное значение  $\lambda_i^{(k)}$  удовлетворяет неравенству

$$|\lambda_i^{(k)} - \lambda_i| \approx \left| \frac{\lambda_{p+1}}{\lambda_i} \right|^k, \quad i = 1:p.$$

(Без ускорения правая часть равна  $|\lambda_{i+1}/\lambda_i|^k$ .) В алгоритме Стюарта время от времени вычисляется разложение Шура матриц  $Q_k^T A Q_k$ . Этот метод может быть очень полезным в тех ситуациях, когда матрица  $A$  большая и разреженная и требуются несколько ее старших собственных значений.

### 7.3.3. QR-итерации

Мы сейчас «выведем» QR-итерации (7.3.1) и исследуем их сходимость. Пусть  $p = n$  в (7.3.4) и собственные значения матрицы  $A$  удовлетворяют условию

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|.$$

Разобьем на блоки матрицу  $Q$  из (7.3.5) и матрицу  $Q_k$  из (7.3.4) следующим образом:

$$Q = [q_1, \dots, q_n], \quad Q_k = [q_1^{(k)}, \dots, q_n^{(k)}].$$

Если

$$\text{dist}(D_i(A^H), \text{span}\{q_1^{(0)}, \dots, q_i^{(0)}\}) < 1, \quad i = 1:n, \quad (7.3.7)$$

то из теоремы 7.3.1 следует, что

$$\text{dist}(\text{span}\{q_1^{(k)}, \dots, q_i^{(k)}\}, \text{span}\{q_1, \dots, q_i\}) \rightarrow 0$$

для  $i = 1:n$ . Это означает, что матрицы  $T_k$ , определяемые равенством

$$T_k = Q_k^H A Q_k,$$

сходятся к верхней треугольной форме. Следовательно, можно сказать, что метод ортогональных итераций вычисляет разложение Шура при условии, что начальное приближение  $Q_0 \in \mathbf{C}^{n \times n}$  полное в смысле (7.3.7).

QR-итерации возникают естественно при размышлении о том, как получить матрицу  $T_k$  непосредственно из ее предшественника  $T_{k-1}$ . С одной стороны, мы имеем из (7.3.4) и определения матрицы  $T_{k-1}$ , что

$$T_{k-1} = Q_{k-1}^H A Q_{k-1} = Q_{k-1}^H (A Q_{k-1}) = (Q_{k-1}^H Q_k) R_k.$$

С другой стороны,

$$T_k = Q_k^H A Q_k = (Q_k^H A Q_{k-1})(Q_{k-1}^H Q_k) = R_k (Q_{k-1}^H Q_k).$$

Следовательно, матрицу  $T_k$  можно определить, вычисляя QR-разложение матрицы  $T_{k-1}$  и затем перемножая те же сомножители в обратном порядке. Это в точности то, что делается в (7.3.1).

**Пример 7.3.3.** Если итерационный процесс

```
for k = 1, 2, ...
    A = QR
    A = RQ
end
```

применить к матрице из примера 7.3.1, то строго нижние треугольные элементы будут уменьшаться следующим образом:

$k$	$O( a_{21} )$	$O( a_{31} )$	$O( a_{32} )$
1	$10^{-1}$	$10^{-1}$	$10^{-2}$
2	$10^{-2}$	$10^{-2}$	$10^{-3}$
3	$10^{-2}$	$10^{-3}$	$10^{-3}$
4	$10^{-3}$	$10^{-3}$	$10^{-3}$
5	$10^{-3}$	$10^{-4}$	$10^{-3}$
6	$10^{-4}$	$10^{-5}$	$10^{-3}$
7	$10^{-4}$	$10^{-5}$	$10^{-3}$
8	$10^{-5}$	$10^{-6}$	$10^{-4}$
9	$10^{-5}$	$10^{-7}$	$10^{-4}$
10	$10^{-6}$	$10^{-8}$	$10^{-4}$

Заметим, что одна QR-итерация требует  $O(n^3)$  вычислений. Более того, так как сходимость только линейная (если она имеет место), то понятно, что этот метод является недоступно дорогим способом вычисления разложения Шура. К счастью, как мы покажем в § 7.4 и § 7.5, эти практические трудности можно обойти.

### 7.3.4. LR-итерации

В заключение сделаем несколько замечаний о степенных итерациях, которые основаны на LU-разложении, а не на QR-разложении. Пусть матрица  $C_0 \in \mathbb{C}^{n \times p}$  имеет ранг  $p$ . В соответствии с (7.3.4) мы имеем следующий итерационный процесс:

```

for  $k = 1, 2, \dots$ 
     $Z_k = QG_{k-1}$ 
     $Z_k = G_kR_k$  (LU-разложение)
end

```

(7.3.8)

Предположим, что  $p = n$  и что мы определяем матрицы  $T_k$  равенством

$$T_k = G_k^{-1}AG_k. \quad (7.3.9)$$

Можно показать, что если  $L_0 = G_0$ , то эти матрицы  $T_k$  могут быть получены следующим образом:

```

 $T_0 = L_0^{-1}AL_0$ 
for  $k = 1, 2, \dots$ 
     $T_{k-1} = L_kR_k$  (LU-разложение)
     $T_k = R_kL_k$ 
end

```

(7.3.10)

Итерационные процессы (7.3.8) и (7.3.10) называют *ступенчатыми* и *LR-итерациями* соответственно. Для успешного применения обоих методов необходим выбор ведущего элемента. Смотри [Wilkinson, AEP], с. 602.

## Приложение

Для того чтобы установить теорему 7.3.1, нам потребуется следующая лемма, которая оценивает норму степеней матрицы и обратных им матриц.

**Лемма 7.3.2.** Пусть  $Q^H A Q = T = D + N$  – разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$ , где  $D$  – диагональная, а  $N$  – строго верхняя треугольная матрицы. Пусть  $\lambda$  и  $\mu$  означают максимальное и минимальное по абсолютной величине собственные числа матрицы  $A$ . Если  $\theta \geq 0$ , то для всех  $k \geq 0$  имеем

$$\|A^k\|_2 \leq (1 + \theta)^{n-1} \left( |\lambda| + \frac{\|N\|_F}{1 + \theta} \right)^k. \quad (7.3.11)$$

Если матрица  $A$  невырожденная и число  $\theta \geq 0$  удовлетворяет неравенству  $(1 + \theta)|\mu| > \|N\|_F$ , то для всех  $k \geq 0$  мы также имеем

$$\|A^{-k}\|_2 \leq (1 + \theta)^{n-1} \left( \frac{1}{|\mu| - \|N\|_F/(1 + \theta)} \right)^k. \quad (7.3.12)$$

*Доказательство.* Для  $\theta \geq 0$  определим диагональную матрицу  $\Delta$  следующим равенством

$$\Delta = \text{diag}(1, (1 + \theta), (1 + \theta)^2, \dots, (1 + \theta)^{n-1})$$

и заметим, что  $k_2(\Delta) = (1 + \theta)^{n-1}$ . Так как матрица  $N$  – строго верхняя треугольная, то несложно проверить, что  $\|\Delta N \Delta^{-1}\|_F \leq \|N\|_F/(1 + \theta)$ . Следовательно,

$$\begin{aligned} \|A^k\|_2 &= \|T^k\|_2 = \|\Delta^{-1}(D + \Delta N \Delta^{-1})^k \Delta\|_2 \leq \\ &\leq k_2(\Delta)(\|D\|_2 + \|\Delta N \Delta^{-1}\|_2)^k \leq (1 + \theta)^{n-1} \left( |\lambda| + \frac{\|N\|_F}{1 + \theta} \right)^k. \end{aligned}$$

С другой стороны, если матрица  $A$  невырожденная и  $(1 + \theta)|\mu| > \|N\|_F$ , то  $\|\Delta D^{-1}N\Delta^{-1}\|_2 < 1$  и, используя лемму 2.3.3, мы получим

$$\begin{aligned} \|A^{-k}\|_2 &= \|T^{-k}\|_2 = \|\Delta^{-1}[(I + \Delta D^{-1}N\Delta^{-1})^{-1}D^{-1}]^k\Delta\|_2 \leq \\ &\leq \chi_2(\Delta) \left( \frac{\|D^{-1}\|_2}{1 - \|\Delta D^{-1}N\Delta^{-1}\|_2} \right)^k \leq \\ &\leq (1 + \theta)^{n-1} \left( \frac{1}{|\mu| - \|N\|_F / (1 + \theta)} \right)^k. \quad \square \end{aligned}$$

*Доказательство теоремы 7.3.1.* Легко показать по индукции, что  $A^k Q_0 = Q_k (R_k \dots R_1)$ . Подставляя (7.3.5) и (7.3.6) в это уравнение, получаем

$$T^k \begin{bmatrix} V_0 \\ W_0 \end{bmatrix} = \begin{bmatrix} V_k \\ W_k \end{bmatrix} (R_k \dots R_1),$$

где  $V_k = Q_\alpha^H Q_k$  и  $W_k = Q_\beta^H Q_k$ . Из леммы 7.1.4 известно, что существует матрица  $X \in \mathbb{C}^{p \times (n-p)}$ , такая, что

$$\begin{bmatrix} I_p & X \\ 0 & I_{n-p} \end{bmatrix}^{-1} \begin{bmatrix} T_{11} & T_{22} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} I_p & X \\ 0 & I_{n-p} \end{bmatrix} = \begin{bmatrix} T_{11} & 0 \\ 0 & T_{22} \end{bmatrix}.$$

Более того, так как  $\text{sep}(T_{11}, T_{22})$  является наименьшим сингулярным значением линейного преобразования  $\phi(X) = T_{11}X - XT_{22}$ , то из уравнения  $\phi(X) = -T_{12}$  непосредственно следует, что

$$\|X\|_F \leq \frac{\|T_{12}\|_F}{\text{sep}(T_{11}, T_{22})}. \quad (7.3.13)$$

Поэтому

$$\begin{bmatrix} T_{11}^k & 0 \\ 0 & T_{22}^k \end{bmatrix} \begin{bmatrix} V_0 - XW_0 \\ W_0 \end{bmatrix} = \begin{bmatrix} V_k - XW_k \\ W_k \end{bmatrix} (R_k \dots R_1).$$

Если предположить, что матрица  $V_0 - XW_0$  невырожденная, то это уравнение можно решить относительно  $W_k$ :

$$W_k = T_{22}^k (V_0 - XW_0)^{-1} T_{11}^{-k} (V_k - XW_k).$$

Беря это выражение по норме и учитывая, что

$$\text{dist}(D_p(A), \text{range}(Q_k)) = \|Q_\beta^H Q_k\|_2 = \|W_k\|_2,$$

получим

$$\text{dist}(D_p(A), \text{range}(Q_k)) \leq \|T_{22}^k\|_2 \|(V_0 - XW_0)^{-1}\|_2 \|T_{11}^{-k}\|_2 (1 + \|X\|_F). \quad (7.3.14)$$

Для того чтобы обосновать это неравенство, мы должны показать, что матрица  $V_0 - XW_0$  невырожденная.

Из уравнения  $A^H Q = QT^H$  следует, что

$$A^H (Q_\alpha - Q_\beta X^H) = (Q_\alpha - Q_\beta X^H) T_{11}^H,$$

т. е. ортогональные столбцы матрицы

$$Z = (Q_\alpha - Q_\beta X^H)(I + XX^H)^{1/2}$$

образуют базис в  $D_p(A^H)$ . Можно показать, что

$$V_0 - XW_0 = (I + XX^H)^{1/2}Z^HQ_0$$

и, следовательно,

$$\begin{aligned}\sigma_p(V_0 - XW_0) &\geq \sigma_p(Z^HQ_0) = \\ &= \sigma_p(V_0 - XW_0) \geq \sigma_p(Z^HQ_0) = \sqrt{1 - d^2} > 0.\end{aligned}$$

Это показывает, что матрица  $V_0 - XW_0$  действительно обратима и

$$\|(V_0 - XW_0)^{-1}\|_2 \leq 1/\sqrt{1 - d^2}. \quad (7.3.15)$$

Наконец, используя лемму 7.3.2, можно показать, что

$$\|T_{22}^k\|_2 \leq (1 + \theta)^{n-p-1} \left( |\lambda_{p+1}| + \frac{\|N\|_F}{1 + \theta} \right)^k$$

и

$$\|T_{11}^{-k}\|_2 \leq (1 + \theta)^{p-1} \left( |\lambda_p| - \frac{\|N\|_F}{1 + \theta} \right)^k.$$

Окончательный результат получаем, подставляя эти неравенства вместе с (7.3.13) и (7.3.15) в (7.3.14).  $\square$

### Задачи

**7.3.1.(а)** Показать, что если матрица  $X \in \mathbb{C}^{n \times n}$  невырожденная, то  $\|A\|_X = \|X^{-1}AX\|_2$  есть матричная норма, обладающая свойством  $\|AB\|_X \leq \|A\|_X \|B\|_X$ . (б) Пусть  $A \in \mathbb{C}^{n \times n}$  и  $\rho = \max|\lambda_i|$ . Показать, что для любого  $\varepsilon > 0$  существует невырожденная матрица  $X \in \mathbb{C}^{n \times n}$ , такая, что  $\|A\|_X = \|X^{-1}AX\|_2 \leq \rho + \varepsilon$ . Доказать, что существует константа  $M$ , такая, что  $\|A^k\|_2 \leq M(\rho + \varepsilon)^k$  для любого неотрицательного целого. (Подсказка: положить  $X = Q \text{diag}(1, a, \dots, a^{n-1})$ , где  $Q^H A Q = D + N$  – разложение Шура матрицы  $A$ .)

**7.3.2.** Проверить, что (7.3.10) вычисляет матрицы  $T_k$ , определяемые равенством (7.3.9).

**7.3.3.** Предположим, что матрица  $A \in \mathbb{C}^{n \times n}$  невырожденная, а матрица  $Q_0 \in \mathbb{C}^{n \times p}$  имеет ортонормированные столбцы. Следующий итерационный процесс называют *обратными ортогональными итерациями*

for  $k = 1, 2, \dots$

Решить  $AZ_k = Q_{k-1}$  относительно  $Z_k \in \mathbb{C}^{n \times p}$ .

$Z_k = Q_k R_k$  (QR-разложение)

end

Объяснить, почему этот процесс может обычно быть использован для вычисления  $p$  минимальных по абсолютной величине собственных значений матрицы  $A$ . Заметим, что для выполнения этих итераций необходимо, чтобы была возможность решать системы линейных уравнений с матрицей  $A$ . Когда  $p = 1$ , этот метод называют методом обратных (степенных) итераций.

### Замечания и литература к § 7.3

Подробное практическое обсуждение степенного метода дано в Wilkinson (AEP), гл. 10. Рассмотрены методы ускорения базового итерационного процесса, вычисление недоминирующих собственных значений, работа с комплексно сопряженными парами собственных значений. Связь между различными степенными итерациями обсуждается в

Parlett B. N. and Poole W. G. (1973). "A Geometric Theory for the QR, LU, and Power Iterations", SIAM J. Num. Anal. 10, 389–412.

QR-итерации были независимо предложены в

Francis J. G. F. (1961). "The QR Transformation: A Unitary Analogue to the LR Transformation", Comp. J. 4, 265–71, 332–334.

Kublanovskaya V. N. (1961). "On Some Algorithms for the Solution of the Complete Eigenvalue Problem", USSR Comp. Math. Phys. 3, 637–657.

Из названия первой статьи можно заключить, что LR-итерации включают в себя QR-итерации. Первый, более фундаментальный алгоритм был предложен в

Rutishauser H. (1958). "Solution of Eigenvalue Problems with the LR Transformation", Nat. Bur. Stand. App. Math. Ser 49, 47–81.

Было опубликовано много работ о сходимости QR-итераций. В их числе

Parlett B. N. (1965). "Convergence of the QR Algorithm", Numer. Math. 7, 187–193.

Parlett B. N. (1966). "Singular and Invariant Matrices Under the QR Algorithm", Math. Comp. 20, 611–615.

Parlett B. N. (1968). "Global Convergence of the Basic QR Algorithm on Hessenberg Matrices", Math. Comp. 22, 803–817.

Wilkinson J. H. (1965). "Convergence of the LR, QR, and Related Algorithms", Comp. J. 8, 77–84.

Уилкинсон [Wilkinson (AEP), гл. 9] также рассматривает теорию сходимости этого важного алгоритма.

Более глубокого понимания сходимости QR-алгоритма можно достичь, читая

Nanda T. (1985). "Differential Equations and the QR Algorithm", SIAM J. Numer. Anal. 22, 310–321.

Watkins D. S. (1982). "Understanding the QR Algorithm", SIAM Review 24, 427–440.

Следующие статьи посвящены различным практическим и теоретическим аспектам одновременных итераций:

Clint M. and Jennings A. (1971). "A Simultaneous Iteration Method for the Unsymmetric Eigenvalue Problem", J. Inst. Math. Applic. 8, 11–21.

Jennings A. and Orr D. R. L. (1971). "Application of the Simultaneous Iteration Method to Undamped Vibration Problems", Inst. J. Numer. Math. Eng. 3, 13–24.

Jennings A. and Stewart W. J. (1975). "Simultaneous Iteration for the Partial Eigensolution of Real Matrices", J. Inst. Math. Applic. 15, 351–362.

Rutishauser H. (1970). "Simultaneous Iteration Method for Symmetric Matrices", Numer. Math. 16, 205–23. See also Hacla, pp. 284–302.

Stewart G. W. (1975c). "Methods of Simultaneous Iteration for Calculating Eigenvectors of Matrices" in Topics in Numerical Analysis II, ed. John. J. H. Miller, Academic Press, New York, pp. 185–196.

Stewart G. W. (1976d). "Simultaneous Iteration for Computing Invariant Subspaces of Non-Hermitian Matrices", Numer. Math. 25, 123–136.

Смотри также гл. 10 работы

Jennings A. (1977b). Matrix Computation for Engineers and Scientists, John Wiley, and Sons, New York.

Одновременные итерации и алгоритмы Ланцюша (гл. 9) являются основными методами вычисления нескольких собственных значений произвольной разреженной матрицы.

## 7.4. Хессенбергова форма и вещественная форма Шура

В этом и следующем разделах мы покажем, как сделать QR-итерации (7.3.1) быстрым и эффективным методом вычисления разложения Шура. Поскольку большинство проблем собственных значений и собственных подпространств вещественные, мы остановимся на развитии вещественного аналога (7.3.1), который запишем следующим образом:

$$\begin{aligned}
 H_0 &= U_0^T A U_0 \\
 \text{for } k &= 1, 2, \dots \\
 H_{k-1} &= U_k R_k \quad (\text{QR-разложение}) \\
 H_k &= R_k U_k \\
 \text{end}
 \end{aligned} \tag{7.4.1}$$

Здесь  $A \in \mathbb{R}^{n \times n}$ , каждая матрица  $U_k \in \mathbb{R}^{n \times n}$  – ортогональная и каждая матрица  $R_k \in \mathbb{R}^{n \times n}$  – верхняя треугольная. Трудность, связанная с этими вещественными итерациями, состоит в том, что матрицы  $H_k$  могут никогда не сойтись к строго «открывающей собственные значения» треугольной форме в случае, если матрица  $A$  имеет комплексные собственные значения. По этой причине мы должны оставить бесплодные надежды и быть довольны вычислением альтернативного разложения, известного как вещественное разложение Шура. Заметим, что в численной линейной алгебре обычно, не нарушая общности, фокусируют внимание на вещественных матричных задачах, поскольку большинство вещественных алгоритмов имеют очевидные комплексные аналоги. QR-итерации не являются исключением.

Для того чтобы эффективно вычислять вещественное разложение Шура, мы должны тщательно выбрать начальное ортогональное преобразование подобия  $U_0$  в (7.4.1). В частности, если мы выберем  $U_0$  так, что матрица  $H_0$  будет верхней хессенберговой, то количество работы на итерации уменьшится с  $O(n^3)$  до  $O(n^2)$ . Начальное преобразование к хессенберговой форме ( $U_0$ -вычисление) само по себе является очень важным алгоритмом и может быть реализовано последовательностью матричных операций Хаусхолдера.

#### 7.4.1. Вещественное разложение Шура

Вещественное разложение Шура представляет собой блочную верхнюю триангуляризацию с диагональными блоками размера  $1 \times 1$  и  $2 \times 2$ .

**Теорема 7.4.1 (Вещественное разложение Шура).** *Если матрица  $A \in \mathbb{R}^{n \times n}$ , то существует ортогональная матрица  $Q \in \mathbb{R}^{n \times n}$ , такая, что*

$$Q^T A Q = \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ 0 & R_{22} & \dots & R_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_{mm} \end{bmatrix}, \quad (7.4.2)$$

где каждый блок  $R_{ii}$  – либо  $1 \times 1$ -матрица, либо  $2 \times 2$ -матрица, имеющая комплексно сопряженные собственные значения.

**Доказательство.** Комплексные собственные значения матрицы  $A$  должны входить в  $\lambda(A)$  комплексно сопряженными парами, так как характеристический многочлен  $\det(zI - A)$  имеет вещественные коэффициенты. Пусть  $k$  – число комплексно сопряженных пар в  $\lambda(A)$ . Докажем теорему индукцией по  $k$ .

Заметим прежде всего, что лемма 7.1.2 и теорема 7.1.3 имеют очевидные вещественные аналоги. Следовательно, теорема верна, если  $k = 0$ . Предположим теперь, что  $k \geq 1$ . Если  $\lambda = \gamma + i\mu \in \lambda(A)$  и  $\mu \neq 0$ , то в  $\mathbb{R}^n$  существуют векторы  $y$  и  $z$  ( $z \neq 0$ ), такие, что

$$A(y + iz) = (\gamma + i\mu)(y + iz),$$

т. е.

$$A[y z] = [y z] \begin{bmatrix} \gamma & \mu \\ -\mu & \gamma \end{bmatrix}.$$

Предположение, что  $\mu \neq 0$ , подразумевает, что  $y$  и  $z$  образуют двумерное вещественное инвариантное подпространство матрицы  $A$ . Из леммы 7.1.1 следует, что существует ортогональная матрица  $U \in \mathbb{R}^{n \times n}$ , такая, что

$$U^T A U = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} 2 \\ n-2 \\ 2 \\ n-2 \end{matrix},$$

где  $\lambda(T_{11}) = \{\lambda, \bar{\lambda}\}$ . Согласно предположению индукции, существует ортогональная матрица  $\tilde{U}$ , такая, что матрица  $\tilde{U}^T T_{22} \tilde{U}$  имеет требуемую структуру. Полагая  $Q = U \text{diag}(I_2, \tilde{U})$ , получаем искомое разложение.  $\square$

Теорема 7.4.1 показывает, что любая вещественная матрица ортогонально подобна некоторой верхней почти треугольной матрице. Понятно, что вещественные и мнимые части комплексных собственных значений могут быть легко получены из диагональных блоков размера  $2 \times 2$ .

#### 7.4.2. Хессенбергов QR-шаг

Вернемся к быстрому вычислению одного QR-шага в (7.4.1). В этом отношении самый страшный недостаток (7.4.1) состоит в том, что каждый шаг требует полного QR-разложения, стоящего  $O(n^3)$  флопов. К счастью, количество работы на одной итерации может быть уменьшено на порядок, если разумно выбрать ортогональную матрицу  $U_0$ . В частности, если матрица  $U_0^T A U_0 = H_0 = (h_{ij})$  — верхняя хессенбергова ( $h_{ij} = 0$ ,  $i > j + 1$ ), то для вычисления каждой матрицы  $H_k$  требуется только  $O(n^2)$  флопов. Для того чтобы это увидеть, рассмотрим процесс вычисления матриц  $H = QR$  и  $H_+ = RQ$ , когда  $H$  — верхняя хессенбергова. Используя процедуры вращения **givens**, **row.rot** и **col.rot**, которые мы описали в § 5.1.8, мы можем триангуляризовать  $H$  последовательностью  $n - 1$  вращений Гивенса. В частности, мы можем найти вращения  $G_1, \dots, G_{n-1}$  с  $G_i = G(i, i + 1, \theta)$ , такие, что матрица  $G_{n-1}^T \dots G_1^T H = R$  будет верхняя треугольная:

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G_1} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G_2} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{G_3} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}$$

Смотри алгоритм 5.2.3.

Вычисление матрицы  $R G_{n-1} \dots G_1$  будет столь же простым, если использовать столбцовую процедуру вращения **col.rot**:

$$\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{G_3} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G_2} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{G_1} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

В целом получаем

**Алгоритм 7.4.1.** Если  $H - n \times n$  верхняя хессенбергова матрица, то этот алгоритм записывает в  $H$  матрицу  $H_+ = RQ$ , где  $H = QR$  – QR-разложение матрицы  $H$ . Кроме того,  $Q = G_1, \dots, G_{n-1}$ , произведение вращений Гивенса, где  $G_k$  – вращение в плоскостях  $k$  и  $k+1$ .

```

for  $k = 1:n - 1$ 
    {Сгенерировать  $G_k$  и применить его:  $H = G(k, k + 1, \theta_k)^T H.$ }
    [ $c(k) s(k)$ ] = givens( $H(k, k), H(k + 1, k)$ )
     $H(k:k + 1, k:n) = \text{row.rot}(H(k:k + 1, k:n), c(k), s(k))$ 
end
for  $k = n - 1:-1:1$ 
    {Обновить:  $H = HG(k, k + 1, \theta_k).$ }
     $H(1:k + 1, k:k + 1) = \text{col.rot}(H(1:k + 1, k:k + 1), c(k), s(k))$ 
end
```

Учитывая, что  $G_k = G(k, k + 1, \theta_k)$ , легко проверить, что матрица  $Q = G_{n-1} \dots G_1$  – верхняя хессенбергова. Поэтому матрица  $RQ = H_+$  – также верхняя хессенбергова. Этот алгоритм требует примерно  $6n^2$  флопов, и, следовательно, он на порядок более быстрый, чем QR-шаг для полной матрицы.

**Пример 7.4.1.** Если алгоритм 7.4.1 применить к матрице

$$H = \begin{bmatrix} 3 & 1 & 2 \\ 4 & 2 & 3 \\ 0 & 0.01 & 1 \end{bmatrix},$$

то

$$G_1 = \begin{bmatrix} 0.6 & -0.8 & 0 \\ 0.8 & 0.6 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.9996 & -0.0249 \\ 0 & 0.0249 & 0.9996 \end{bmatrix}$$

и

$$H_+ = \begin{bmatrix} 4.7600 & -2.5442 & 5.4653 \\ 0.3200 & 0.1856 & -2.1796 \\ 0.0000 & 0.0263 & 1.0540 \end{bmatrix}.$$

### 7.4.3. Разложение Хессенберга

Нам осталось показать, как может быть вычислено разложение Хессенберга

$$U_0^T A U_0 = H \quad U_0^T U_0 = I. \quad (7.4.3)$$

Это вычисление не сложное. Преобразование  $U_0$  является произведением матриц Хаусхолдера  $P_1, \dots, P_{n-2}$ . Роль матрицы  $P_k$  заключается в обнулении  $k$ -го столбца ниже поддиагонали. В случае  $n = 6$  мы имеем

$$A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_2} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{bmatrix}$$

$$\begin{array}{c}
 \left[ \begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \end{array} \right] \xrightarrow{P_3} \left[ \begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \end{array} \right] \\
 \\
 \xrightarrow{P_4} \left[ \begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{array} \right]
 \end{array}$$

В общем случае, после  $k - 1$  шагов мы вычислим  $k - 1$  матриц Хаусхолдера  $P_1, \dots, P_{k-1}$ , таких, что матрица

$$(P_1 \dots P_{k-1})^T A (P_1 \dots P_{k-1}) = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \bar{P}_k \\ 0 & \bar{P}_k B_{32} & \bar{P}_k B_{33} \bar{P}_k \end{bmatrix}$$

— верхняя хессенбергова в первых  $k$  столбцах. Используя процедуры Хаусхолдера **house**, **row. house** и **col. house** из § 5.1.8, получаем

**Алгоритм 7.4.2 (Преобразование Хаусхолдера к хессенберговой форме).** Для заданной матрицы  $A \in \mathbb{R}^{n \times n}$ , следующий алгоритм записывает в  $A$  матрицу  $H = U_0^T A U_0$ , где  $H$  — верхняя хессенбергова матрица и  $U_0 = (P_1 \dots P_{n-2})$  — произведение матриц Хаусхолдера. Значимая часть  $k$ -го вектора Хаусхолдера хранится в  $A(k+2:n, k)$ .

```

for  $k = 1 : n - 2$ 
     $u(k+1:n) = \text{house}(A(k+1:n, k))$ 
     $\{ A = P_k^T A P_k, \text{ где } P_k = \text{diag}(I_k, \bar{P}_k). \}$ 
     $A(k+1:n, k:n) = \text{row. house}(A(k+1:n, k:n), v(k+1:n))$ 
     $A(1:n, k+1:n) = \text{col. house}(A(1:n, k+1:n), v(k+1:n))$ 
     $A(k+2:n, k) = v(k+2:n)$ 
end

```

Этот алгоритм требует  $10 n^3 / 3$  флопов. Если матрица  $U_0$  формируется явно, то дополнительно требуется  $4 n^3 / 3$  флопов. Подробное описание см. в Martin, Wilkinson (1968 d).

Свойства погрешности округления описанного метода приведения матрицы  $A$  к хессенберговой форме весьма привлекательны. Уилкинсон [Wilkinson (AEP), с. 351] установил, что вычисленная хессенбергова матрица  $\hat{H}$  удовлетворяет равенству  $\hat{H} = Q^T (A + E) Q$ , где  $Q^T Q = I$  и

$$\|E\|_F \leq c n^2 u \|A\|_F$$

с малой константой  $c$ .

**Пример 7.4.2.** Если

$$A = \begin{bmatrix} 1 & 5 & 7 \\ 3 & 0 & 6 \\ 4 & 3 & 1 \end{bmatrix} \quad \text{и} \quad U_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.6 & 0.8 \\ 0 & 0.8 & -0.6 \end{bmatrix},$$

то

$$U_0^T A U_0 = H = \begin{bmatrix} 1.00 & 8.60 & -0.020 \\ 5.00 & 4.96 & -0.072 \\ 0.00 & 2.28 & -3.96 \end{bmatrix}.$$

#### 7.4.4. Трехуровневые варианты

Преобразование Хессенберга (алгоритм 7.4.2) богато двухуровневыми операциями: половина – гахру и половина – внешние произведения. Мы кратко обсудим два способа введения в этот процесс трехуровневых вычислений.

Первый подход состоит в блочном преобразовании к блочно-хессенберговой форме и прямо приводит к цели. Предположим (для простоты), что  $n = rN$  и запишем

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}_{n-r}^r.$$

Пусть мы вычислили QR-разложение матрицы  $A_{21} = \bar{Q}_1 R_1$  и матрица  $\bar{Q}_1$  представлена в WY-форме. То есть мы имеем матрицы  $W_1 Y_1 \in \mathbb{R}^{(n-r)r}$ , такие, что  $\bar{Q}_1 = I - W_1 Y_1^T$ . (Подробнее см. в § 5.2.2.) Если  $Q_1 = \text{diag}(I_2, \bar{Q}_1)$ , то

$$Q_1^T A Q_1 = \begin{bmatrix} A_{11} & A_{12} \bar{Q}_1 \\ R_1 & \bar{Q}_1^T A_{22} \bar{Q}_1 \end{bmatrix}.$$

Заметим, что перевычисления блоков (1,2) и (2,2) богаты трехуровневыми операциями благодаря тому, что матрица  $\bar{Q}_1$  представлена в WY-форме. Это полностью поясняет весь процесс, поскольку матрица  $Q_1^T A Q$  – блочная верхняя хессенбергова в первом блочном столбце. Мы далее повторяем те же вычисления для первых  $r$  столбцов матрицы  $\bar{Q}_1^T A_{22} \bar{Q}_1$ . После  $N-2$  таких шагов получаем матрицу

$$H = U_0^T A U_0 = \begin{bmatrix} H_{11} & H_{12} & \dots & \dots & H_{1N} \\ H_{21} & H_{22} & \dots & \dots & H_{2N} \\ 0 & \ddots & \ddots & \dots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & H_{N,N-1} & H_{NN} \end{bmatrix},$$

где каждый блок  $H_{ij}$  – матрица размера  $r \times r$  и  $U_0 = Q_1 \dots Q_{N-2}$  с матрицами  $Q_i$  в WY-форме. Весь алгоритм имеет трехуровневую часть вида  $1 - O(1/N)$ .

Заметим, что поддиагональные блоки матрицы  $H$  являются верхними треугольными и, следовательно, эта матрица имеет нижнюю ширину ленты  $r$ . Можно преобразовать  $H$  к настоящей хессенберговой форме, используя вращения Гивенса, для того чтобы обнулить все поддиагонали, кроме первой. Остается открытым вопрос – насколько эта «подчистка» ухудшит трехуровневое представление.

Донгарра, Хаммерлинг и Соренсон [Dongarra, Hammarling, Sorenson (1987)] показали, как переходить непосредственно к хессенберговой форме, используя смесь операций гахру и трехуровневых замещений. Их идея требует минимальных

замещений после генерации каждого преобразования Хаусхолдера. Предположим, например, что первая матрица Хаусхолдера  $P_1$  вычислена. Чтобы сгенерировать матрицу  $P_2$ , нам нужен только второй столбец матрицы  $P_1^T A P_1$ , а не результат полного внешнего произведения. Чтобы сгенерировать  $P_3$ , нам нужен только третий столбец матрицы  $P_2^T P_1^T A P_1 P_2$  и т. д. Таким образом, матрицы Хаусхолдера можно определить, используя только операции гахру. Не требуется вычисление внешних произведений. Когда нужное число матриц Хаусхолдера станет известно, их можно собрать вместе и применить в трехуровневом виде.

#### 7.4.5. Важные свойства матрицы Хессенберга

Рассмотренное разложение Хессенберга не единственно. Если  $Z$ -любая ортогональная матрица размера  $n \times n$  и мы применим алгоритм 7.4.2 к матрице  $Z^T A Z$ , то матрица  $Q^T A Q = H$  будет верхней хессенберговой, где  $Q = Z U_0$ . Однако равенство  $Q e_1 = Z(U_0 e_1) = Z e_1$  наводит на мысль, что матрица  $H$  может быть единственна, если фиксирован первый столбец матрицы  $Q$ . Именно эту ситуацию обеспечивает отсутствие у матрицы  $H$  нулевых поддиагональных элементов. Хессенберговы матрицы, обладающие этим свойством, называют *неприводимыми*. Имеет место очень важная теорема, проясняющая однозначность разложения Хессенберга.

**Теорема 7.4.2 (Теорема о неявном  $Q$ )<sup>1)</sup>.** Пусть  $Q = [q_1, \dots, q_n]$  и  $V = [v_1, \dots, v_n]$  – ортогональные матрицы, обладающие тем свойством, что обе матрицы  $Q^T A Q = H$  и  $V^T A V = G$ , являются верхними хессенберговыми, где  $A \in \mathbb{R}^{n \times n}$ . Пусть  $k$  означает наименьшее положительное целое, для которого  $h_{k+1,k} = 0$ , в предположении, что  $k = n$ , если матрица  $H$  – неприводимая. Если  $U_1 = q_1$ , то  $v_i = \pm q_i$  и  $|h_{i,i-1}| = |g_{i,i-1}|$  для  $i = 2:k$ . Кроме того, если  $k < n$ , то  $g_{k+1,k} = 0$ .

**Доказательство.** Определим ортогональную матрицу  $W = [w_1, \dots, w_n]$  равенством  $W = V^T Q$ . Заметим, что  $G W = W H$ . Поэтому для  $i = 2:k$  имеем:

$$h_{i,i-1} w_i = G w_{i-1} - \sum_{j=1}^{i-1} h_{j,i-1} w_j.$$

Так как  $w_1 = e_1$ , это означает, что матрица  $[w_1, \dots, w_k]$  – верхняя треугольная и, следовательно,  $w_i = \pm e_i$  для  $i = 2:k$ . Поэтому из равенств  $w_i = V^T q_i$  и  $h_{i,i-1} = w_i^T G w_{i-1}$  следует, что  $v_i = \pm q_i$  и  $|h_{i,i-1}| = |g_{i,i-1}|$  для  $i = 2:k$ . Если  $h_{k+1,k} = 0$ , то, игнорируя знаки, имеем:

$$g_{k+1,k} = e_{k+1}^T G e_k = e_{k+1}^T G W e_k = (e_{k+1}^T W)(H e_k) = e_{k+1}^T \sum_{i=1}^k h_{ik} W e_i = \sum_{i=1}^k h_{ik} e_{k+1}^T e_i = 0. \square$$

Суть теоремы о неявном  $Q$  в том, что если каждая из матриц  $Q^T A Q = H$  и  $Z^T A Z = G$  – неприводимая верхняя хессенбергова и матрицы  $Q$  и  $Z$  имеют один и тот же первый столбец, то матрицы  $G$  и  $H$  «в основном равные» в том смысле, что  $G = D^{-1} H D$ , где  $D = \text{diag}(\pm 1, \dots, \pm 1)$ .

Наша следующая теорема показывает, что существует связь между QR-разложением матрицы Крылова  $K(A, Q(:, 1), n)$  и разложением Хессенберга  $Q^T A Q = H$ .

**Теорема 7.4.3.** Пусть  $Q \in \mathbb{R}^{n \times n}$  – ортогональная матрица и  $A \in \mathbb{R}^{n \times n}$ . Тогда  $Q^T A Q = H$  есть неприводимая верхняя хессенбергова матрица тогда и только тогда, когда матрица  $Q^T K(A, Q(:, 1), n) = R$  – невырожденная и верхняя треугольная.

<sup>1)</sup> Название этой теоремы (Implicit Q Theorem) подчеркивает сходство с теоремой о неявной функции (Implicit Function Theorem). – Прим. перев.

*Доказательство.* Предположим, что матрица  $Q \in \mathbb{R}^{n \times n}$  – ортогональная и положим  $H = Q^T A Q$ . Рассмотрим следующее тождество

$$Q^T K(A, Q(:, 1), n) = [e_1, H e_1, \dots, H^{n-1} e_1] \equiv R.$$

Если  $H$  – неприводимая верхняя хессенбергова матрица, то ясно, что матрица  $R$  – верхняя треугольная с элементами  $r_{ii} = h_{21} h_{32} \dots h_{i,i-1}$  для  $i = 2:n$ . Это означает, что матрица  $R$  невырожденная, так как  $r_{11} = 1$ .

Для доказательства обратного утверждения предположим, что матрица  $R$  – верхняя треугольная и невырожденная. Следовательно,  $H(:, k) \in \text{span}\{e_1, e_{k+1}\}$ , так как  $R(:, k+1) = HR(:, k)$ . Это означает, что матрица  $H$  – верхняя хессенбергова. Поскольку  $r_{nn} = h_{21} h_{32} \dots h_{n,n-1} \neq 0$ , матрица  $H$  является также неприводимой.  $\square$

Таким образом, существует более или менее явное соответствие между невырожденными матрицами Крылова  $K(A, x, n-1)$  и ортогональными преобразованиями подобия, приводящими к неприводимой хессенберговой форме.

Наш последний результат касается собственных значений неприводимой верхней хессенберговой матрицы.

**Теорема 7.4.4.** *Если  $\lambda$  – собственное значение неприводимой верхней хессенберговой матрицы  $H \in \mathbb{R}^{n \times n}$ , то его геометрическая кратность равна единице.*

*Доказательство.* Для любого  $\lambda \in \mathbb{C}$  имеем:  $\text{rank}(A - \lambda I) \geq n - 1$ , так как первые  $n - 1$  столбцов матрицы  $H - \lambda I$  линейно независимы.  $\square$

#### 7.4.6. Сопровождающая матрица

Точно так же, как разложение Шура имеет неунитарный аналог – разложение Жордана, разложение Хессенберга имеет неунитарный аналог – сопровождающее разложение. Пусть  $h \in \mathbb{R}^n$ , и пусть матрица Крылова  $K(A, x, n-1)$  невырожденная. Если  $C = C(0:n-1)$  – решение линейной системы  $K(A, x, n-1)C = -A^n x$ , то  $A K(A, x, n-1) = K(A, x, n-1)C$ , где матрица  $C$  имеет следующий вид:

$$C = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{n-1} \end{bmatrix}. \quad (7.4.4)$$

Матрицу  $C$  называют *сопровождающей матрицей*<sup>1)</sup>. Так как

$$\det(zI - C) = c_0 + c_1 z + \dots + c_{n-1} z^{n-1} + z^n,$$

то, если матрица  $Y = K(A, x, n-1)$  невырожденная, разложение  $Y^{-1} A Y = C$  выявляет характеристический многочлен матрицы  $A$ . Это вместе с разреженностью  $C$  привело в различных прикладных областях к «методам сопровождающей матрицы». Эти методы обычно включают:

- Вычисление Хессенбергова разложения  $U_0^T A U_0 = H$ .
- Надежду на неприводимость матрицы  $H$  и вычисление матрицы  $Y = [e_1, H e_1, \dots, H^{n-1} e_1]$ .
- Решение системы  $YC = HY$  относительно  $C$ .

<sup>1)</sup> Сопровождающую матрицу иногда называют матрицей Фробениуса. Наряду с (7.4.4) используют также определения сопровождающей матрицы. – Прим. перев.

К несчастью, эти вычисления могут быть очень неустойчивыми. Матрица  $A$  подобна неприводимой хессенберговой только тогда, когда каждое собственное значение имеет единичную геометрическую кратность. Матрицы, обладающие этим свойством, называют *простыми*. Следовательно, матрица  $Y$  может быть очень плохо обусловлена, если матрица  $A$  близка к некоторой матрице, не являющейся простой.

Полное обсуждение опасностей, связанных с вычислением сопровождающей матрицы, можно найти в Wilkinson (APE), с. 405.

#### 7.4.7. Хессенбергово преобразование через преобразования Гаусса

Коль скоро мы заговорили о неортогональных преобразованиях к хессенберговой форме, мы должны упомянуть о том, что преобразования Гаусса можно использовать вместо преобразований Хаусхолдера в алгоритме 7.4.2. Для определенности предположим, что перестановки  $\Pi_1, \dots, \Pi_{k-1}$  и преобразования Гаусса  $M_1, \dots, M_{k-1}$  выбирались так, что

$$(M_{k-1} \Pi_{k-1} \dots M_1 \Pi_1) A (M_{k-1} \Pi_{k-1} \dots M_1 \Pi_1)^{-1} = B,$$

где

$$B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \\ k-1 & 1 & n-k \end{bmatrix} \quad \begin{matrix} k-1 \\ 1 \\ n-k \end{matrix}$$

— верхняя хессенбергова матрица в первых  $k-1$  столбцах. Перестановку  $\bar{\Pi}_k$  порядка  $n-k$  выбирают затем так, чтобы первый элемент столбца  $\bar{\Pi}_k B_{32}$  был максимальным по абсолютной величине. Это дает возможность найти устойчивое преобразование Гаусса  $\bar{M}_k = I - z_k e_1^T$  того же порядка  $n-k$ , такое, что все компоненты столбца  $\bar{M}_k (\bar{\Pi}_k B_{32})$ , кроме первой, — нули. Полагая  $\Pi_k = \text{diag}(I_k, \bar{\Pi}_k)$  и  $M_k = \text{diag}(I_k, \bar{M}_k)$ , мы видим, что матрица

$$(M_k \Pi_k, \dots, M_1 \Pi_1) A (M_k \Pi_k, \dots, M_1 \Pi_1)^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \Pi_k^T \bar{M}_k^{-1} \\ 0 & \bar{M}_k \bar{\Pi}_k B_{32} & \bar{M}_k \Pi_k B_{33} \bar{\Pi}_k^T \bar{M}_k^{-1} \end{bmatrix}$$

является верхней хессенберговой в первых  $k$  столбцах. Заметим, что  $\bar{M}_k^{-1} = I + z_k e_1^T$ , следовательно, это преобразование состоит из некоторых очень простых одноранговых замещений.

Тщательный подсчет количества операций показывает, что преобразование Гаусса к хессенберговой форме требует только половины числа флопов метода Хаусхолдера. Однако, как и в случае исключения Гаусса с частичным выбором ведущего элемента, есть вероятность (весьма незначительная) роста ведущего элемента со скоростью  $2^n$ . Смотри Businger (1969). Другая проблема, связанная с гауссовским подходом, состоит в том, что числа обусловленности собственных значений  $-s(\lambda)^{-1}$  не сохраняются при неортогональных преобразованиях подобия. Это несколько усложняет схему оценки погрешности, приведенную в § 7.6.

#### Задачи

- 7.4.1. Пусть  $A \in \mathbb{R}^{n \times n}$  и  $z \in \mathbb{R}^n$ . Предложить подробный алгоритм вычисления ортогональной матрицы  $Q$ , такой, что матрица  $Q^T A Q$  — верхняя хессенбергова, а вектор  $Q^T z$  коллинеарен  $e_1$ . (Подсказка: сначала преобразовать  $z$ , а затем применить алгоритм 7.4.2.)

**7.4.2.** Описать полное преобразование Гаусса к хессенберговой форме при помощи процедур `gauss` и `gauss.app` и проверить, что оно требует только  $5n^3/3$  флопов.

**7.4.3.** В некоторых ситуациях, необходимо решать линейную систему  $(A + zI)x = b$  для различных значений  $z \in \mathbb{R}$  и  $b \in \mathbb{R}^n$ . Показать, как эта задача может быть эффективно и устойчиво решена при помощи разложения Хессенберга.

**7.4.4.** Предложить подробный алгоритм явного вычисления матрицы  $U_0$  в алгоритме 7.4.2. Сконструировать алгоритм так, чтобы матрица  $H$  замещалась матрицей  $U_0$ .

**7.4.5.** Пусть  $H \in \mathbb{R}^{n \times n}$  – неприводимая верхняя хессенбергова матрица. Показать, что существует диагональная матрица  $D$ , такая, что каждый поддиагональный элемент матрицы  $D^{-1}HD$  равен единице. Чему равно  $k_2(D)$ ?

**7.4.6.** Пусть  $W, Y \in \mathbb{R}^{n \times n}$  и

$$C = W + iY, \quad B = \begin{bmatrix} W & -Y \\ Y & W \end{bmatrix}.$$

Показать, что если  $\lambda \in \lambda(C)$  – вещественно, то  $\lambda \in \lambda(B)$ . Найти соответствующие собственные векторы.

**7.4.7.** Пусть  $A = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$  – вещественная матрица, имеющая собственные значения  $\lambda \pm i\mu$ , где  $\mu$  не равно нулю. Предложить алгоритм устойчивого определения  $c = \cos(\theta)$  и  $s = \sin(\theta)$ , таких, что

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} w & x \\ y & z \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \lambda & \beta \\ \alpha & \lambda \end{bmatrix},$$

где  $\alpha\beta = -\mu^2$ .

**7.4.8.** Пусть известна собственная пара  $(\lambda, x)$  верхней хессенберговой матрицы  $H \in \mathbb{R}^{n \times n}$ . Предложить алгоритм вычисления ортогональной матрицы  $P$ , такой, что

$$P^T HP = \begin{bmatrix} \lambda & w^T \\ 0 & H_1 \end{bmatrix},$$

где  $H_1 \in \mathbb{R}^{(n-1) \times (n-1)}$  – верхняя хессенбергова матрица. Вычислить  $P$  как произведение вращений Гивенса.

### Замечания и литература к § 7.4

Вещественное разложение Шура впервые опубликовано в

Murnaghan F. D. and Wintner A. (1931). "A Canonical Form for Real Matrices Under Orthogonal Transformations", Proc. Nat. Acad. Sci. 17, 417–20.

Полное описание преобразования к хессенберговой форме дано в Wilkinson (AEP), гл. 6, а остальные процедуры для методов Хаусхольдера приведены в

Martin R. S. and Wilkinson J. H. (1968d). "Similarity Reduction of a General Matrix to Hessenberg Form", Numer. Math. 12, 349–68. See also HACLA, pp. 339–358.

Фортранные версии алгольных процедур из упомянутой выше работы входят в EISPACK. Вращения Гивенса также можно использовать для вычисления хессенбергова разложения. Смотри

Rath W. (1982). "Fast Givens Rotations for Orthogonal Similarity", Numer. Math. 40, 47–56.

Реализация хессенберговой редукции на высокопроизводительных компьютерах обсуждается в

Dongarra J. J., Hammarling S. and Sorensen D. C. (1987). "Block Reduction of Matrices to Condensed form for Eigenvalue Computations", ANL-MCS-TM 99, Argonne National Laboratory, Argonne, Illinois.

Dongarra J.J., Kaufman L. and Hammarling S. (1986). "Squeezing the Most Out of Eigenvalue Solvers on High Performance Computers", Lin. Alg. and Its Applic. 77, 113–136.

На возможность экспоненциального роста ведущего элемента в методе преобразований Гаусса впервые было указано в

Businger P. (1969). "Reducing a matrix to Hessenberg Form", Math. Comp. 23, 819–821.

Однако этот алгоритм можно характеризовать, как и метод Гаусса с частичным выбором ведущего элемента – устойчив для всех практических целей. Смотри Eispack, с. 56–58.

Варианты хессенбергова разложения для разреженных матриц обсуждаются в

Duff I. S. and Reid J. K. (1975). "On the Reduction of Sparse Matrices to Condensed Forms by Similarity Transformations", J. Inst. Math. Applic. 15, 217–224.

Блочная редукция к хессенберговой форме рассматривается в

Dongarra J. J., Hammarling S. and Sorensen D. C. (1987). "Block Reduction of Matrices to Condensed form for Eigenvalue Computations", ANL–MCS–TM 99, Argonne National Laboratory, Argonne, Illinois.

Если некоторое собственное значение неприводимой верхней хессенберговой матрицы известно, то можно обнулить последний поддиагональный элемент, используя преобразование подобия Гивенса. Смотри

Businger P. A. (1971). "Numerically Stable Deflation of Hessenberg and Symmetric Tridiagonal Matrices", BIT 11, 262–270.

Некоторые интересные свойства хессенберговой формы можно найти в

Ikebe Y. (1979). "On Inverses of Hessenberg Matrices", Lin. Alg. and Its Applic. 24, 93–97.

Parlett B. N. (1967). "Canonical Decomposition of Hessenberg Matrices", Math. Comp. 21, 223–227.

Хотя хессенбергово разложение в большей степени ценится как «предварительное» разложение для QR-итераций, растет его популярность как дешевой альтернативы более дорогому разложению Шура. Примеры приложений, где оно оказалось очень полезным, можно найти в

Enright W. (1979). "On the Efficient and Reliable Numerical Solution of Large Linear Systems of O.D.E. "s", IEEE Trans. Auto. Cont. AC-24, 905–908.

Golub G. H., Nash S. and Van Loan C. (1979). "A Hessenberg-Schur Method for the Problem  $AX + XB = C$ ", IEEE Trans. Auto. Cont. AC-24, 909–913.

Laub A. (1981). "Efficient Multivariable Frequency Response Computations", IEEE Trans. Auto. Cont. AC-26, 407–408.

Miminis G. and Paige C. C. (1982). "An Algorithm for Pole Assignment of Time Invariant Linear Systems", International J. of Control 35, 341–354.

Paige C. C. (1981). "Properties of Numerical Algorithms Related to Computing Controllability", IEEE Trans. Auto. Cont. AC-26, 130–138.

Van Loan C. (1982b). "Using the Hessenberg Decomposition in Control Theory", in Algorithms and Theory in Filtering and Control, Sorenson D. C. and Wets R. J. (eds.), Mathematical Programming Study No. 18, North Holland, Amsterdam, pp. 102–111.

## 7.5. Практический QR-алгоритм

Вернемся к хессенберговым QR-итерациям, которые мы запишем следующим образом:

$$H = U_0^T A U_0 \quad (\text{Преобразование Хессенберга})$$

for  $k = 1, 2, \dots$

$$H = UR \quad (\text{QR-разложение})$$

$$H = RU$$

end

(7.5.1)

Цель этого раздела – описать сходимость матрицы  $H$  к верхней квазитреугольной форме и показать, как скорость сходимости можно увеличить, вводя сдвиги.

### 7.5.1. Исчерпывание

Не нарушая общности, мы можем предположить, что каждая хессенбергова матрица  $H$  в (7.5.1) является неприводимой. Если это не так, то на некотором этапе мы имеем

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \\ p & n-p \end{bmatrix}$$

где  $1 \leq p < n$ , и исходная задача расщепляется на две задачи меньшего размера с матрицами  $H_{11}$  и  $H_{22}$ . Термин *исчерпывание* также используют в этом контексте, обычно когда  $p = n - 1$  и  $n - 2$ .

На практике расщепление осуществляют всякий раз, когда поддиагональный элемент матрицы  $H$  достаточно мал. Например, в EISPACK, если

$$|h_{p+1,p}| \leq c \mathbf{u}(|h_{pp}| + |h_{p+1,p+1}|) \quad (7.5.2)$$

при малой константе  $c$ , то  $h_{p+1,p}$  «объявляют» равным нулю. Это оправдано, так как погрешность округления порядка  $\mathbf{u}||H||$  уже присутствует во всех элементах матрицы.

### 7.5.2. QR-итерации со сдвигами

Пусть  $\mu \in \mathbb{R}$ . Рассмотрим итерации:

$$H = U_0^T A U_0 \quad (\text{Преобразование Хессенберга})$$

for  $k = 1, 2, \dots$

Определить скаляр  $\mu$  (7.5.3)

$$H - \mu I = UR \quad (\text{QR-разложение})$$

$$H = RU + \mu I$$

end

Скаляр  $\mu$  называют *сдвигом*. Каждая матрица  $H$ , порождаемая (7.5.3), подобна матрице  $A$ , так как  $RU + \mu I = U^T (UR + \mu I)U = U^T HU$ . Если мы упорядочили собственные значения  $\lambda_i$  матрицы  $A$  так, что

$$|\lambda_1 - \mu| \geq \dots \geq |\lambda_n - \mu|$$

и величина  $\mu$  не меняется от итерации к итерации, то, согласно теории § 7.3,  $p$ -й поддиагональный элемент матрицы  $H$  сходится к нулю со скоростью

$$\left| \frac{\lambda_{p+1} - \mu}{\lambda_p - \mu} \right|^k.$$

Конечно, если  $\lambda_p = \lambda_{p+1}$ , то сходимости нет совсем. Но если, например,  $\mu$  более близко к  $\lambda_n$ , чем к остальным собственным значениям, то обнуление элемента  $(n, n-1)$  ускорится. В экстремальном случае имеем:

**Теорема 7.5.1.** Пусть  $\mu$  – собственное значение неприводимой хессенберговой матрицы размера  $n \times n$ . Если  $\bar{H} = RU + \mu I$ , где  $H - \mu I = UR$  – QR-разложение матрицы  $H - \mu I$ , то  $h_{n,n-1} = 0$  и  $h_{nn} = \mu$ .

*Доказательство.* Так как  $H$  – неприводимая хессенбергова матрица, первые  $n - 1$  столбцов матрицы  $H - \mu I$  линейно независимы при любом  $\mu$ . Следовательно, если  $UR = (H - \mu I) - QR$ -разложение, то  $r_{ii} \neq 0$  для  $i = 1:n - 1$ . Но если матрица  $H - \mu I$  вырожденная, то  $r_{11} \dots r_{nn} = 0$ . Поэтому  $r_{nn} = 0$  и  $\bar{H}(n) = [0, \dots, 0, \mu]$ .  $\square$

Теорема говорит, что если мы сдвигаем на точное собственное значение, то в точной арифметике исчерпывание происходит на первом шаге.

**Пример 7.5.1.** Если

$$H = \begin{bmatrix} 9 & -1 & -2 \\ 2 & 6 & -2 \\ 0 & 1 & 5 \end{bmatrix},$$

то  $G \in \lambda(H)$ . Если  $UR = H - 6I$  есть QR-разложение, то равенство  $\bar{H} = RU + 6I$  обеспечивает матрица

$$\bar{H} = \begin{bmatrix} 8.5384 & -3.7313 & -1.0090 \\ 0.6343 & 5.4615 & 1.3867 \\ 0.0000 & 0.0000 & 6.0000 \end{bmatrix}.$$

### 7.5.3. Стратегия одинарного сдвига

Рассмотрим варьирование  $\mu$  от итерации к итерации, учитывающее новую информацию о  $\lambda(A)$  по мере того как поддиагональные элементы сходятся к нулю. Хороший эвристический подход состоит в том, чтобы считать элемент  $h_{nn}$  наилучшим приближением к собственному значению среди всех диагональных элементов. Если мы на каждой итерации осуществляем сдвиг на эту величину, то получаем QR-итерации с одинарным сдвигом:

```
for k = 1, 2, ...
    μ = H(n, n)
    H - μI = UR (QR-разложение)
    H = RU + μI
end
```

(7.5.4)

Если элемент  $(n, n - 1)$  стремится к нулю, то это вероятнее всего будет происходить с квадратичной скоростью. Для того чтобы увидеть это, мы воспользуемся примером из Stewart (IMC), с. 366. Пусть  $H$  – неприводимая верхняя хессенбергова матрица вида

$$H = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \varepsilon & h_{nn} \end{bmatrix}$$

и мы выполнили один шаг QR алгоритма с одинарным сдвигом:  $UR = H - h_{nn}I$ ,  $\bar{H} = RU + h_{nn}I$ . После  $n - 2$  шагов приведения матрицы  $H - h_{nn}I$  к верхнему треугольному виду мы получим матрицу следующей структуры:

$$H = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & a & b \\ 0 & 0 & 0 & \varepsilon & 0 \end{bmatrix}.$$

Нетрудно показать, что элемент  $(n, n - 1)$  матрицы  $\bar{H} = RU + h_{nn}I$  равен  $-\varepsilon^2 b / (\varepsilon^2 + a^2)$ . Если предположить, что  $\varepsilon \ll a$ , то ясно, что новый элемент  $(n, n - 1)$  имеет порядок  $\varepsilon^2$ , точно того же мы ждем от алгоритма, сходящегося квадратично.

**Пример 7.5.2.** Если

$$H = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 0.001 & 7 \end{bmatrix}$$

и  $UR = H - 7I$  – QR-разложение, то равенство  $\bar{H} = RU + 7I$  обеспечивает

$$\bar{H} \approx \begin{bmatrix} -0.5384 & 1.6908 & 0.8351 \\ 0.3076 & 6.5264 & -6.6555 \\ 0.0000 & 2.10^{-5} & 7.0119 \end{bmatrix}.$$

Близкие к оптимальным сдвиги, такие, как рассмотренный выше, почти всегда гарантируют малое  $\bar{h}_{n, n-1}$ . Однако это лишь эвристика. Есть примеры, в которых  $\bar{h}_{n, n-1}$  – относительно большой элемент матрицы, несмотря на то что  $\sigma_{\min}(H - \mu I) \approx u$ .

#### 7.5.4. Стратегия двойного сдвига

К несчастью, при использовании (7.5.4) можно ожидать трудности, если на некотором этапе собственные значения  $a_1$  и  $a_2$  матрицы

$$G = \begin{bmatrix} h_{mm} & h_{mn} \\ h_{nm} & h_{nn} \end{bmatrix} m = n - 1 \quad (7.5.5)$$

станут комплексными, вследствие чего  $h_{nn}$  будет плохим приближением собственного значения.

Один из способов преодолеть это затруднение состоит в том, чтобы последовательно выполнять два QR-шага с одинарными сдвигами, используя в качестве сдвигов  $a_1$  и  $a_2$ :

$$\begin{aligned} H - a_1 I &= U_1 R_1, \\ H_1 &= R_1 U_1 + a_1 I, \\ H_1 - a_2 I &= U_2 R_2, \\ H_2 &= R_2 U_2 + a_2 I. \end{aligned} \quad (7.5.6)$$

Можно показать, что

$$(U_1 U_2)(R_2 R_1) = M, \quad (7.5.7)$$

где  $M$  – матрица, определяемая равенством

$$M = (H - a_1 I)(H - a_2 I). \quad (7.5.8)$$

Заметим, что  $M$  – вещественная матрица, даже если собственные значения матрицы

$G$  комплексные, так как

$$M = H^2 - sH + tI,$$

где

$$\begin{aligned}s &= a_1 + a_2 = h_{mm} + h_{nn} = \text{trace}(G) \in \mathbb{R} \\ t &= a_1 a_2 = h_{mm} h_{nn} - h_{mn} h_{nm} = \det(G) \in \mathbb{R}.\end{aligned}$$

Поэтому (7.5.7) – QR-разложение вещественной матрицы и мы можем выбрать  $U_1$  и  $U_2$  так, что  $Z = U_1 U_2$  будет вещественной ортогональной матрицей. Следовательно, матрица

$$H_2 = U_2^H H_1 U_2 = U_2^H (U_1^H H U_1) U_2 = (U_1 U_2)^H (U_1 U_2) = Z^T H Z$$

– вещественная.

К сожалению, погрешности округления почти всегда мешают точному возвращению в поле вещественных чисел. Вещественность матрицы  $H_2$  можно гарантировать, если мы

- явно формируем вещественную матрицу  $M = H^2 - sH + tI$ ,
- вычисляем вещественное QR-разложение  $M = ZR$ , и
- полагаем  $H_2 = Z^T H Z$ .

Но поскольку первый из этих шагов требует  $O(n^3)$  флопов, это непрактичный образ действия.

### 7.5.5. Стратегия неявного двойного сдвига

К счастью, оказывается, что мы можем выполнить шаг двойного сдвига за  $O(n^2)$  флопов, обратившись к теореме о неявном  $Q$ . В частности, мы можем осуществить переход от  $H$  к  $H_2$  за  $O(n^2)$  флопов, если

- Вычислим  $Me_1$  – первый столбец матрицы  $M$ .
- Найдем матрицу Хаусхолдера  $P_0$ , такую, что вектор  $P_0(Me_1)$  коллинеарен  $e_1$ .
- Вычислим матрицы Хаусхолдера  $P_1, \dots, P_{n-2}$ , такие, что если  $Z_1$  – произведение  $Z_1 = P_0 P_1 \dots P_{n-2}$ , то  $Z_1^T H Z_1$  – верхняя хессенбергова матрица и первые столбцы матриц  $Z$  и  $Z_1$  одинаковые.

При данных обстоятельствах теорема о неявном  $Q$  позволяет нам заключить, что если матрицы  $Z^T H Z$  и  $Z_1^T H Z_1$  являются неприводимыми верхними хессенберговыми, то они в основном равны<sup>1)</sup>. Заметим, что если эти хессенберговы матрицы не являются неприводимыми, то мы можем выполнить расщепление и перейти к неприводимым подзадачам меньшего размера.

Давайте отработаем детали. Заметим прежде всего, что матрицу  $P_0$  можно определить за  $O(1)$  флопов, так как  $Me_1 = (x, y, z, 0, \dots, 0)^T$ , где

$$\begin{aligned}x &= h_{11}^2 + h_{12} h_{21} - sh_{11} + t, \\ y &= h_{21} (h_{11} + h_{22} - s), \\ h &= h_{21} h_{32}.\end{aligned}$$

Поскольку преобразование подобия с матрицей  $P_0$  меняет только строки и столбцы

<sup>1)</sup> В основном равные (essentially equal) здесь и в дальнейшем означает, что столбцы матриц, имеющие одинаковые номера, равны с точностью до знака. Смотри теорему 7.4.2. – Прим. перев.

1, 2 и 3, понятно, что

$$P_0 H P_0 = \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

Теперь роль матриц Хаусхолдера  $P_1, \dots, P_{n-2}$  заключается в том, чтобы вернуть этой матрице верхний хессенбергов вид

$$\begin{array}{c} \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix} \xrightarrow{P_2} \\ \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & x & x & x & x \end{bmatrix} \xrightarrow{P_3} \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix} \\ \xrightarrow{P_4} \begin{bmatrix} x & x & x & x & x & x \\ x & x & x & x & x & x \\ 0 & x & x & x & x & x \\ 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & x & x \end{bmatrix} \end{array}$$

Ясно, что матрица  $P_k$  имеет, вообще говоря, вид  $P_k = \text{diag}(I_k, \bar{P}_k, I_{n-k-3})$ , где  $\bar{P}_k$ -матрица Хаусхолдера размера  $3 \times 3$ . Например,

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Заметим, что матрица  $P_{n-2}$  является исключением, так как  $P_{n-2} = \text{diag}(I_{n-2}, P_{n-2})$ .

Применимость теоремы 7.4.2 (теоремы о неявном  $Q$ ) следует из того, что  $P_k e_1 = e_1$  при  $k = 1:n-2$  и матрицы  $P_0$  и  $Z$  имеют один и тот же первый столбец. Следовательно,  $Z_1 e_1 = Z e_1$ , и мы можем утверждать, что  $Z_1$  в основном равна  $Z$  при условии, что верхние хессенберговы матрицы  $Z^T HZ$  и  $Z_1^T HZ_1$  неприводимые.

Неявное определение  $H_2$  по  $H$ , рассмотренное выше в общих чертах, было впервые описано Фрэнсисом [Francis (1961)], и мы будем называть его *QR-шагом Фрэнсиса*. Полный шаг Фрэнсиса подытоживает следующий

**Алгоритм 7.5.1 (QR-шаг Фрэнсиса).** Для заданной неприводимой хессенберговой матрицы  $H \in \mathbf{R}^{n \times n}$ , чья нижняя главная подматрица размера  $2 \times 2$  имеет собственные числа  $a_1$  и  $a_2$ , этот алгоритм записывает в  $H$  матрицу  $Z^T H Z$ , где  $Z = P_1 \dots P_{n-2}$  – произведение матриц Хаусхолдера, а матрица  $Z^T(H - a_1 I) \times (H - a_2 I)$  – верхняя треугольная.

```

 $m = n - 1$ 
{Вычислить первый столбец матрицы  $(H - a_1 I)(H - a_2 I)$ .}
 $s = H(m, m) + H(n, n)$ 
 $t = H(m, m)H(n, n) - H(m, n)H(n, m)$ 
 $x = H(1, 1)H(1, 1) + H(1, 2)H(2, 1) - sH(1, 1) + t$ 
 $y = H(2, 1)(H(1, 1) + H(2, 2) - s)$ 
 $z = H(2, 1)H(3, 2)$ 
for  $k = 0 : n - 3$ 
  {Записать в  $H$  матрицу  $P_k H P_k^T$ , где  $P_k = \text{diag}(I_k, \bar{P}_k, I_{n-k-3})$ }
   $v = \text{house}([x \ y]^T)$ 
   $H(k+1:k+3, k+1:n) = \text{row. house}(H(k+1:k+3, k+1:n), v)$ 
   $r = \min\{k+4, n\}$ 
   $H(1:r, k+1:k+3) = \text{col. house}(H(1:r, k+1:k+3), v)$ 
   $x = H(k+2, k+1)$ 
   $y = H(k+3, k+1)$ 
  if  $k < n - 3$ 
     $z = H(k+4, k+1)$ 
  end
end
{ $H = P_{n-2} H P_{n-2}^T$ , где  $P_{n-2} = \text{diag}(I_{n-2}, \bar{P}_{n-2})$ }
 $v = \text{house}([x \ y]^T)$ 
 $H(n-1:n, n-2:n) = \text{row.house}(H(n-1:n, n-2:n), v)$ 
 $H(1:n, n-1, n) = \text{col. house}(H(1:n, n-1, n), v)$ 
```

Этот алгоритм требует  $10n^2$  флоков. Если матрицу  $Z$  накапливают в заданной ортогональной матрице, то необходимо дополнительно  $10n^2$  флоков.

### 7.5.6. Полный процесс

Приведение матрицы  $A$  к хессенбергову виду, используя алгоритм 7.4.1 и последующее итерирование по алгоритму 7.5.1, для того чтобы получить вещественную форму Шура – стандартный способ решения плотной несимметрической проблемы собственных значений. В процессе итераций необходимо контролировать поддиагональные элементы матрицы  $H$  для того, чтобы обнаружить любую возможность расщепления. Как это делают, поясняется в следующем алгоритме.

**Алгоритм 7.5.2 (QR-алгоритм).** Для заданной матрицы  $A \in \mathbf{R}^{n \times n}$  и заданной погрешности  $tol$ , большей чем машинная точность, этот алгоритм вычисляет вещественную каноническую форму Шура  $Q^T A Q = T$ . Записывает в матрицу  $A$  разложение Хессенберга. Если необходимы матрицы  $Q$  и  $T$ , то  $T$  хранят в  $H$ . Если необходимы только собственные значения, то блоки матрицы  $T$  хранят в соответствующих позициях матрицы  $H$ .

Использовать алгоритм 7.4.2 для вычисления хессенбергова разложения

$$H = U_0^T A U_0, \quad \text{где} \quad U_0 = P_0 \dots P_{n-2}.$$

Если необходима матрица  $Q$ , то сформировать  $Q = P_0 \dots P_{n-2}$ . Смотри § 5.1.6.

**until**  $q = n$

Положить равными нулю все поддиагональные элементы, удовлетворяющие неравенству:

$$|h_{i,i-1}| \leq tol(|h_{ii}| + |h_{i-1,i-1}|).$$

Найти наибольшее неотрицательное  $q$  и наименьшее неотрицательное  $p$ , такие, что

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix},$$

где  $H_{33}$  – верхняя квазитреугольная матрица, а матрица  $H_{22}$  – не-приводимая. (Замечание: и  $p$ , и  $q$  может быть нулем<sup>1)</sup>.)

**if**  $q < n$

Выполнить QR-шаг Фрэнсиса для матрицы  $H_{22}: H_{22} = Z^T H_{22} Z$

**if** требуется  $Q$

$$Q = Q \operatorname{diag}(I_p, Z, I_q)$$

$$H_{12} = H_{12} Z$$

$$H_{23} = Z^T H_{23}$$

**end**

**end**

**end**

Привести к верхнему треугольному виду все диагональные блоки размера  $2 \times 2$ -матрицы  $H$ , имеющие вещественные собственные значения, и, если это необходимо, накопить эти преобразования.

Этот алгоритм требует  $25n^3$  флопов, если вычисляются матрицы  $Q$  и  $T$ . Если требуются только собственные значения, то необходимо  $10n^3$  флопов. Эти оценки числа флопов очень приблизительные и основаны на том эмпирическом наблюдении, что в среднем требуются только две итерации Фрэнсиса для отщепления нижнего блока размера  $1 \times 1$  или  $2 \times 2$ . Более детальный анализ времени, необходимого для выполнения QR-алгоритма, приведен в EISPACK, с. 119.

**Пример 7.5.3.** Если алгоритм 7.5.2 применить к матрице

$$A = H = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 4 & 4 & 5 & 6 & 7 \\ 0 & 3 & 6 & 7 & 8 \\ 0 & 0 & 2 & 8 & 9 \\ 0 & 0 & 0 & 1 & 10 \end{bmatrix},$$

то поддиагональные элементы будут сходиться следующим образом:

<sup>1)</sup> Если матрица  $H$  неприводимая (это возможно при первом подходе), то  $p = q = 0$ . – Прим. перев.

Итерация	$O( h_{21} )$	$O( h_{32} )$	$O( h_{43} )$	$O( h_{54} )$
1	$10^0$	$10^0$	$10^0$	$10^0$
2	$10^0$	$10^0$	$10^0$	$10^0$
3	$10^0$	$10^0$	$10^{-1}$	$10^0$
4	$10^0$	$10^0$	$10^{-3}$	$10^{-3}$
5	$10^0$	$10^0$	$10^{-6}$	$10^{-5}$
6	$10^{-1}$	$10^0$	$10^{-13}$	$10^{-13}$
7	$10^{-1}$	$10^0$	$10^{-28}$	$10^{-13}$
8	$10^{-4}$	$10^0$	сошелся	сошелся
9	$10^{-8}$	$10^0$		
10	$10^{-8}$	$10^0$		
11	$10^{-16}$	$10^0$		
12	$10^{-32}$	$10^0$		
13	сошелся	сошелся		

Свойства погрешностей округления QR-алгоритма такие же, какие можно ожидать от любого метода ортогональных матриц. Вычисленная вещественная форма Шура  $\hat{T}$  ортогонально подобна матрице  $A$ , т.е.

$$Q^T(A + E)Q = \hat{T},$$

где  $Q^T Q = I$  и  $\|E\|_2 \approx u \|A\|_2$ . Вычисленные матрицы  $\hat{Q}$  – почти ортогональные, в том смысле, что  $\hat{Q}^T \hat{Q} = I + F$ , где  $\|F\|_2 \approx u$ .

Расположение собственных значений в матрице  $\hat{T}$  довольно произвольное. Но мы обсудим в § 7.6, как можно получить любое расположение, используя простую процедуру, меняющую местами два соседних диагональных элемента.

### 7.5.7. Масштабирование

В заключение заметим, что если элементы матрицы  $A$  сильно отличаются по величине, то матрицу  $A$  нужно масштабировать, прежде чем применять QR-алгоритм. Это  $O(n^2)$  алгоритм, в котором вычисляется диагональная матрица  $D$ , такая, что если

$$D^{-1}AD = [c_1, \dots, c_n] = \begin{bmatrix} r_1^T \\ \vdots \\ r_n^T \end{bmatrix},$$

то  $\|r_i\|_\infty \approx \|c_i\|_\infty$  для  $i = 1:n$ . Диагональную матрицу  $D$  ищут в виде  $D = \text{diag}(\beta^{i_1}, \dots, \beta^{i_n})$ , где  $\beta$  – основание используемой арифметики с плавающей запятой. Заметим, что матрица  $D^{-1}AD$  может быть вычислена без округления. Когда матрица  $A$  масштабированная, вычисленные собственные значения часто более точные. Смотри Parlett, Reinsch (1969).

### Задачи

7.5.1. Показать, что если матрица  $\bar{H} = Q^T H Q$  получена при помощи одного QR-шага с одинарным сдвигом и  $\bar{H} = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$ , то  $|h_{21}| \leq |y^2 x| / [(w - z)^2 + y^2]$ .

7.5.2. Вывести формулы для элементов диагональной матрицы  $D$  размера  $2 \times 2$ , минимизирующей  $\|D^{-1}AD\|_F$ , где  $A = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$ .

**7.5.3.** Объяснить, как QR-шаг с одинарным сдвигом:  $H - \mu I = UR$ ,  $\tilde{H} = RU + \mu I$  может быть выполнен явно. То есть показать, как переход от  $\tilde{H}$  к  $H$  может быть выполнен без вычитания сдвига  $\mu$  из диагональных элементов матрицы  $H$ .

**7.5.4.** Предположим, что матрица  $H$  верхняя хессенбергова и мы нашли разложение  $RH = LU$ , посредством гауссовского исключения с частичным выбором ведущего элемента. (См. алгоритм 4.3.4.) Показать, что матрица  $H = U(P^T L)$  – верхняя хессенбергова и подобна  $H$ . (Это основа *модифицированного LR-алгоритма*.)

**7.5.5.** Показать, что если задана матрица  $H = H_0$  и мы генерируем матрицы  $H_k$ :  $H_k - \mu_k I = U_k R_k$ ,  $H_{k+1} = R_k U_k + \mu_k I$ , то

$$(U_1, \dots, U_j)(R_j, \dots, R_1) = (H - \mu_1 I), \dots, (H - \mu_j I).$$

### Замечания и литература к § 7.5

Развитие практического QR-алгоритма началось с важной статьи

Rutishauser H. (1958). "Solution of Eigenvalue Problems with the LR Transformation", Nat. Bur. Stand. App. Math. Ser. 49, 47–81.

Алгоритм, описанный здесь, был затем «ортогоанализован» в

Francis J. G. F. (1961). "The QR Transformation: A Unitary Analogue to the LR Transformation, Parts I and II", Comp. J. 4, 265–72, 332–345.

Описания практического QR-алгоритма можно найти в Wilkinson (AEP), глава 8 и Stewart (IMC), гл. 7. Алгольные процедуры для LR- и QR-методов приведены в

Martin R. S., Peters G. and Wilkinson J. H. (1970). "The QR Algorithm for Real Hessenberg Matrices", Numer. Math. 14, 219–231. See also HACLA, pp. 359–371.

Martin R. S. Wilkinson J. H. (1968). "The Modified LR Algorithm for Complex Hessenberg Matrices", Numer. Math. 12, 369–76. See also HACLA, pp. 396–403.

Их фортрановские аналоги входят в EISPACK.

Аспекты проблемы масштабирования обсуждаются в

Osborne E. F. (1960). "On Preconditioning of Matrices", JACM 7, 338–345.

Parlett B. N. and Reinsch C. (1969). "Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors", Numer. Math. 13, 292–304. See also HACLA, pp. 315–326.

## 7.6. Методы вычисления инвариантных подпространств

Различные важные проблемы инвариантных подпространств могут быть решены, если найдено вещественное разложение Шура. В этом разделе мы обсуждаем как

- Вычислить собственные векторы, соответствующие некоторому подмножеству  $\lambda(A)$ .
- Найти ортогональный базис для заданного инвариантного подпространства.
- Блоочно диагонализировать  $A$ , используя хорошо обусловленные преобразования подобия.
- Найти базис из собственных векторов, невзирая на их обусловленность.
- Найти приближенную каноническую форму Жордана матрицы  $A$ .

Вычисление собственного вектора (инвариантного подпространства) разреженной матрицы обсуждается в другом месте. Смотри § 7.3, а также гл. 8 и 9.

### 7.6.1. Выделение собственных векторов при помощи обратных итераций

Пусть  $q^{(0)} \in \mathbb{C}^n$  – заданный вектор, нормированный в 2-норме и пусть матрица  $A - \mu I \in \mathbb{R}^{n \times n}$  – невырожденная. Следующий процесс называют обратными

итерациями:

**for**  $k = 1, 2, \dots$

Решить систему  $(A - \mu I)z^{(k)} = q^{(k-1)}$

$$q^{(k)} = z^{(k)} / \|z^{(k)}\|_2$$

$$\lambda^{(k)} = q^{(k)H} A q^{(k)}$$

**end**

Обратные итерации – это степенной метод, примененный к матрице  $(A - \mu I)^{-1}$ .

Для того чтобы анализировать поведение процесса (7.6.1), предположим, что матрица  $A$  имеет базис из собственных векторов  $\{x_1, \dots, x_n\}$  и что  $Ax_i = \lambda_i x_i$  для  $i = 1 : n$ . Если

$$q^{(0)} = \sum_{i=1}^n \beta_i x_i,$$

то  $q^{(k)}$  – нормированный вектор, сонаправленный с

$$(A - \mu I)^{-k} q^{(0)} = \sum_{i=1}^n \frac{\beta_i}{(\lambda_i - \mu)^k} x_i.$$

Ясно, что если  $\mu$  более близко к собственному значению  $\lambda_i$ , чем к остальным собственным значениям, то в векторе  $q^{(k)}$  преобладает компонента, соответствующая направлению  $x_i$  при условии, что  $\beta_j \neq 0$ .

Простым правилом остановки итераций (7.6.1) может служить следующее правило: остановить итерации, если невязка

$$r^{(k)} = (A - \mu I)q^{(k)}$$

удовлетворяет неравенству

$$\|r^{(k)}\|_\infty \leq c\|A\|_\infty, \quad (7.6.2)$$

где  $c$  – некоторая константа порядка единицы. Так как

$$(A + E_k)q^{(k)} = \mu q^{(k)},$$

где  $E_k = -r^{(k)}q^{(k)T}$ , то неравенство (7.6.2) обязывает  $\mu$  и  $q^{(k)}$  быть собственной парой близкой матрицы.

Обратные итерации можно использовать совместно с QR-алгоритмом следующим образом:

- Найти хессенбергово разложение  $U_0^T A U_0 = H$ .
- Применить к матрице  $H$  итерации с двойным неявным сдвигом Фрэнсиса без накопления преобразований.
- Для каждого вычисленного собственного значения  $\lambda$ , соответствующий собственный вектор которого нужно найти, применить (7.6.1) с матрицей  $A = H$  и  $\mu = \lambda$ , чтобы получить вектор  $z$ , такой, что  $Hz \approx \mu z$ .
- Положить  $x = U_0 z$ .

Обратные итерации с матрицей  $H$  очень экономичны, так как 1) нам совсем не нужно накапливать преобразования в процессе двойных итераций Фрэнсиса; 2) мы можем факторизовывать<sup>1)</sup> матрицы вида  $H - \lambda I$  за  $O(n^2)$  флопов, и 3) обычно только одна итерация требуется для получения достаточно точного собственного вектора.

Этот последний пункт является, возможно, наиболее интересным аспектом обратных итераций и требует некоторого обоснования, так как  $\lambda$  может быть

<sup>1)</sup> Имеются в виду QR- или LR-разложения. – Прим. перев.

сравнительно неточным, если оно плохо обусловлено. Предположим для простоты, что  $\lambda$  вещественно и пусть

$$H - \lambda I = \sum_{i=1}^n \sigma_i u_i v_i^T = U \sum V^T$$

-SVD матрицы  $H - \lambda I$ . Из того что мы говорили о свойствах ошибок округления QR-алгоритма в § 7.5.6, следует, что существует матрица  $E \in \mathbb{R}^{n \times n}$ , такая, что матрица  $H + E - \lambda I$  - вырожденная и  $\|E\|_2 \approx \mu \|H\|_2$ . Это означает, что  $\sigma_n \approx \mu \sigma_1$  и  $\|(H - \lambda I)v_n\|_2 \approx \mu \sigma_1$ , т. е.  $v_n$  - хорошее приближение собственного вектора. Ясно, что если начальный вектор  $q^{(0)}$  имеет разложение

$$q^{(0)} = \sum_{i=1}^n \gamma_i u_i,$$

то в векторе

$$z^{(1)} = \sum_{i=1}^n \frac{\gamma_i}{\sigma_i} v_i$$

преобладает направление  $v_n$ . Заметим, что если величина  $s(\lambda) \approx |u_n^T v_n|$  мала, то в векторе  $z^{(1)}$  скорее всего не хватает направления  $u_n$ . Это объясняет (эмпирически), почему другой шаг обратных итераций, вероятно, не даст более хорошей аппроксимации собственного вектора, особенно если  $\lambda$  плохо обусловлено. Более подробно смотри в Peters, Wilkinson (1979).

**Пример 7.6.1.** Матрица

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-10} & 1 \end{bmatrix}$$

имеет собственные значения  $\lambda_1 = 0.99999$  и  $\lambda_2 = 1.00001$  и соответствующие собственные векторы  $x_1 = (1, -10^{-5})^T$  и  $x_2 = (1, 10^{-5})^T$ . Обусловленность обоих собственных векторов порядка  $10^5$ . Приближенное собственное значение  $\mu = 1$  есть точное собственное значение матрицы  $A + E$ , где

$$E = \begin{bmatrix} 0 & 0 \\ -10^{-10} & 0 \end{bmatrix}.$$

Следовательно, качество  $\mu$  соответствует обычному качеству собственных значений, найденных с помощью QR-алгоритма, когда его реализуют в 10-разрядной арифметике с плавающей запятой.

Если применить (7.6.1) с начальным вектором  $q^{(0)} = (0, 1)$ , то  $q^{(1)} = (1, 0)$  и  $\|Aq^{(1)} - \mu q^{(1)}\|_2 = 10^{-10}$ . Однако следующий шаг даст  $q^{(2)} = (0, 1)$ , для которого  $\|Aq^{(2)} - \mu q^{(2)}\|_2 = 1$ . Этот пример обсуждается в Peters, Wilkinson (1979).

## 7.6.2. Упорядочение собственных значений в вещественной форме Шура

Напомним, что вещественное разложение Шура дает информацию об инвариантных подпространствах. Если

$$Q^T A Q = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix}$$

и  $\lambda(T_{11}) \cap \lambda(T_{22}) = \emptyset$ , то первые  $p$  столбцов матрицы  $Q$  образуют единственное унитарное инвариантное подпространство, соответствующее  $\lambda(T_{11})$  (см. § 7.2.5). К

сожалению, итерации Фрэнсиса дают вещественное разложение Шура  $Q_F^T A Q_F = T_F$ , в котором собственные значения довольно случайно расположены вдоль главной диагонали матрицы  $T_F$ . Это создает проблему, если нам нужен ортогональный базис для инвариантного подпространства, которому соответствуют собственные значения, не расположенные в верхней части диагонали матрицы  $T_F$ . Ясно, что нам нужен метод вычисления ортогональной матрицы  $Q_D$ , такой, что матрица  $Q_D^T T_F Q_D$  – верхняя квазитреугольная с подходящим расположением собственных значений.

Пример  $2 \times 2$  наводит на мысль, как это может быть выполнено. Предположим, что

$$Q_F^T A Q_F = T_F = \begin{bmatrix} \lambda_1 & t_{12} \\ 0 & \lambda_2 \end{bmatrix}, \quad \lambda_1 \neq \lambda_2,$$

и мы хотим изменить порядок собственных значений. Заметим, что  $T_F x = \lambda_2 x$ , где

$$x = \begin{bmatrix} t_{12} \\ \lambda_2 - \lambda_1 \end{bmatrix}.$$

Пусть  $Q_D$  – вращение Гивенса, такое, что вторая компонента вектора  $Q_D^T x$  нулевая. Если  $Q = Q_F Q_D$ , то

$$(Q^T A Q) e_1 = Q_D^T T_F (Q_D e_1) = \lambda_2 Q_D^T (Q_D e_1) = \lambda_2 e_1$$

и, следовательно, матрица  $Q^T A Q$  должна иметь вид

$$Q^T A Q = \begin{bmatrix} \lambda_2 & \pm t_{12} \\ 0 & \lambda_1 \end{bmatrix}.$$

Систематически выполняя перестановки в соседних парах собственных значений, используя этот метод, мы можем передвинуть любое подмножество спектра  $\lambda(A)$  в верхнюю часть диагональной матрицы, при условии что мы не столкнемся с выступами размера  $2 \times 2$ .

**Алгоритм 7.6.1.** Для заданных ортогональной матрицы  $Q \in \mathbb{R}^{n \times n}$ , верхней треугольной матрицы  $T = Q^T A Q$  и подмножества  $\Delta = \{\lambda_1, \dots, \lambda_p\}$  спектра  $\lambda(A)$  следующий алгоритм вычисляет ортогональную матрицу  $Q_D$ , такую, что матрица  $Q_D^T T Q_D = S$  – верхняя треугольная и  $\{s_{11}, \dots, s_{pp}\} = \Delta$ . Матрицы  $Q$  и  $T$  замещаются матрицами  $Q Q_D$  и  $S$  соответственно.

```

while {t11, ..., tpp} ≠ Δ
    for k = 1:n - 1
        if tkk ∉ Δ и tk+1,k+1 ∈ Δ
            {Найти QD и обновить: T = QDT T QD, Q = Q QD}
            [c, s] = givens(T(k, k + 1), T(k + 1, k + 1) - T(k, k))
            T(k:k + 1, k:n) = row. rot(T(k:k + 1, k:n), c, s)
            T(1:k + 1, k:k + 1) = col. rot(T(1:k + 1, k:k + 1), c, s)
            Q(1:n, k + 1) = col. rot(Q(1:n, k:k + 1), c, s)
        end
    end
end

```

Этот алгоритм требует  $k(12n)$  флопов, где  $k$  – общее число требуемых перестановок. Число  $k$  никогда не превышает  $(n - p)p$ .

Перестановки становятся немного более сложными, когда матрица  $T$  имеет на диагонали блоки размера  $2 \times 2$ . Детали см. в Ruhe (1970a) и Stewart (1976a).

Понятно, что эти методы перестановок могут быть использованы для того, чтобы упорядочить собственные значения, скажем, от максимального до минимального по модулю.

Вычисление инвариантных подпространств посредством преобразования вещественного разложения Шура чрезвычайно устойчиво. Если  $\hat{Q} = [\hat{q}_1, \dots, \hat{q}_n]$  означает вычисленную ортогональную матрицу  $Q$ , то  $\|\hat{Q}^T \hat{Q} - I\|_2 \approx u$  и существует матрица  $E$ , удовлетворяющая неравенству  $\|E\|_2 \approx u \|A\|_2$ , такая, что  $(A + E)\hat{q}_i \in \text{span}\{\hat{q}_1, \dots, \hat{q}_p\}$ ,  $i = 1:p$ .

### 7.6.3. Блочная диагонализация

Пусть

$$T = \begin{bmatrix} T_{11} & T_{12} & \dots & T_{1q} \\ 0 & T_{22} & \dots & T_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & T_{qq} \\ n_1 & n_2 & & n_q \end{bmatrix} \quad (7.6.3)$$

– разбиение некоторой вещественной канонической формы Шура  $Q^T A Q = T \in \mathbb{R}^{n \times n}$ , такое, что попарные пересечения подмножеств  $\lambda(T_{11}), \dots, \lambda(T_{qq})$  пусты. Согласно теореме 7.1.6 существует матрица  $Y$ , такая, что  $Y^{-1} T Y = \text{diag}(T_{11}, \dots, T_{qq})$ . Теперь мы опишем практическую процедуру определения  $Y$  вместе с анализом чувствительности  $Y$  как функции вышеуказанного разбиения.

Разобьем на блоки единичную матрицу  $I_n = [E_1, \dots, E_q]$  сообразно разбиению матрицы  $T$  и определим матрицу  $Y_{ij} \in \mathbb{R}^{n_i \times n_j}$  следующим образом

$$Y_{ij} = I_n + E_i Z_{ij} E_j^T, \quad i < j, \quad Z_{ij} \in \mathbb{R}^{n_i \times n_j}.$$

Другими словами, матрица  $Y_{ij}$  выглядит точно так же, как единичная матрица, за исключением того, что  $(i, j)$ -ю блочную позицию занимает матрица  $Z_{ij}$ . Следовательно, если  $Y_{ij}^{-1} T Y_{ij} = \bar{T} = (\bar{T}_{ij})$ , то матрицы  $T$  и  $\bar{T}$  одинаковые, за исключением блоков

$$\begin{aligned} \bar{T}_{ij} &= T_{ii} Z_{ij} - Z_{ij} T_{jj} + T_{ij}, \\ \bar{T}_{ik} &= T_{ik} - Z_{ij} T_{jk} \quad (k = j + 1:q), \\ \bar{T}_{kj} &= T_{ki} Z_{ij} + T_{kj} \quad (k = 1:i-1). \end{aligned}$$

Следовательно, блок  $T_{ij}$  можно обнулить, если есть алгоритм решения уравнения Сильвестра

$$FZ - ZG = C, \quad (7.6.4)$$

где  $F \in \mathbb{R}^{p \times p}$  и  $G \in \mathbb{R}^{r \times r}$  – заданные верхние квазитреугольные матрицы и  $C \in \mathbb{R}^{p \times r}$ .

Бартелс и Стюарт [Bartels, Stewart, 1972] придумали метод, позволяющий это сделать. Пусть  $C = [c_1, \dots, c_r]$  и  $Z = [z_1, \dots, z_r]$  – столбцовые разбиения. Если  $g_{k+1,k} = 0$ , то, сравнивая столбцы в (7.6.4), мы находим, что

$$Fz_k - \sum_{i=1}^k g_{ik} z_i = c_k.$$

Следовательно, если мы знаем  $z_1, \dots, z_{k-1}$ , то можем решить квазитреугольную

систему

$$(F - g_{kk} I) z_k = c_k + \sum_{i=1}^{k-1} g_{ik} z_i$$

относительно  $z_k$ . Если  $g_{k+1,k} \neq 0$ , то  $z_k$  и  $z_{k+1}$  можно найти одновременно, решая систему размера  $2 \times 2$

$$\begin{bmatrix} F - g_{kk} I & -g_{mk} I \\ -g_{km} & F - g_{mm} I \end{bmatrix} \begin{bmatrix} z_k \\ z_m \end{bmatrix} = \begin{bmatrix} c_k \\ c_m \end{bmatrix} + \sum_{i=1}^{k-1} \begin{bmatrix} g_{ik} z_i \\ g_{im} z_i \end{bmatrix}, \quad (7.6.5)$$

где  $m = k + 1$ . Переставляя уравнения в соответствии с перестановкой  $(1, p+1, 2, p+2, \dots, p, 2p)$ , получаем ленточную систему, которая может быть решена за  $O(p^2)$  флоков. Детали можно найти в [Bartels, Stewart, 1972]. Мы подытожим весь процесс для простейшего случая, когда обе матрицы  $F$  и  $G$  – треугольные.

**Алгоритм 7.6.2 (Алгоритм Бартелса–Стюарта).** Для заданных матрицы  $C \in \mathbb{R}^{p \times r}$  и верхних треугольных матриц  $F \in \mathbb{R}^{p \times p}$  и  $G \in \mathbb{R}^{r \times r}$ , удовлетворяющих равенству  $\lambda(F) \cap \lambda(G) = \emptyset$ , следующий алгоритм записывает в матрицу  $C$  решение уравнения  $FZ - ZG = C$ .

```
for k = 1:r
    {Вычислить  $b = c_k + \sum_{i=1}^{k-1} g_{ik} z_i$ }
    C(1:p, k) = C(1:p, k) + C(1:p, 1:k-1) G(1:k-1, k)
    Решить  $(F - g_{kk} I)z = b$  относительно  $z$ .
    C(1:p, k) = z
end
```

Этот алгоритм требует  $pr(p+r)$  флоков.

Ясно, что, обнуляя наддиагональные блоки матрицы  $T$  в соответствующем порядке, мы можем преобразовать эту матрицу к блочно-диагональной форме.

**Алгоритм 7.6.3.** Для заданных ортогональной матрицы  $Q \in \mathbb{R}^{n \times n}$ , верхней квази-треугольной матрицы  $T = Q^T A Q$  и разбиения (7.6.3) следующий алгоритм записывает в матрицу  $Q$  матрицу  $QY$ , где  $Y^{-1} TY = \text{diag}(T_{11}, \dots, T_{qq})$ .

```
for j = z:q
    for i = 1:j-1
        Решить  $T_{ii} Z - Z T_{jj} = -T_{ij}$  относительно  $Z$ , используя алгоритм 7.6.2.
        for k = j+1:q
             $T_{ik} = T_{ik} - Z T_{jk}$ 
        end
        for k = 1:q
             $Q_{kj} = Q_{kj} Z + Q_{kj}$ 
        end
    end
end
```

Число флоков, которое требует этот алгоритм, является сложной функцией от размеров блоков в (7.6.3).

Выбор вещественной формы Шура и ее разбиения в (7.6.3) определяет устойчивость уравнений Сильвестра, которые должны быть решены в алгоритме 7.6.3. Это в свою очередь влияет на обусловленность матрицы  $Y$  и эффективность блочной диагонализации. Причина этих зависимостей в том, что относительная ошибка

вычисленного решения  $\hat{Z}$  уравнения

$$T_{ii} Z - Z T_{jj} = -T_{ij} \quad (7.6.6)$$

удовлетворяет приближенному равенству

$$\frac{\| \hat{Z} - Z \|_F}{\| Z \|_F} \approx u \frac{\| T \|_F}{\text{sep}(T_{ii}, T_{jj})}.$$

Детали смотри в [Golub, Nash, Van Loan, 1979]. Поскольку

$$\text{sep}(T_{ii}, T_{jj}) = \min_{X \neq 0} \frac{\| T_{ii} X - X T_{jj} \|_F}{\| X \|_F} \leq \min_{\substack{\lambda \in \lambda(T_{ii}) \\ \mu \in \lambda(T_{jj})}} |\lambda - \mu|,$$

точность может существенно уменьшиться, когда подмножества  $\lambda(T_{ii})$  недостаточно отделены. Более того, если матрица  $Z$  удовлетворяет (7.6.6), то

$$\| Z \|_F \leq \frac{\| T_{ij} \|_F}{\text{sep}(T_{ii}, T_{jj})}.$$

Следовательно, большое по норме решение можно ожидать, если величина  $\text{sep}(T_{ii}, T_{jj})$  мала. Это делает матрицу  $Y$  в алгоритме 7.6.3 плохо обусловленной, так как она является произведением матриц

$$Y_{ij} = \begin{bmatrix} I & Z \\ 0 & I \end{bmatrix}.$$

Замечание:  $\kappa_F(Y_{ij}) = 2n + \| Z \|_F^2$ .

Учитывая эти трудности, Бэвли и Стюарт [Bavely, Stewart (1979)] разработали алгоритм блочной диагонализации, который динамически определяет расположение собственных значений и разбиение (7.6.3) так, что все  $Z$  матрицы в алгоритме 7.6.3 ограничены по норме некоторой величиной, задаваемой пользователем. Они обнаружили, что обусловленность  $Y$  можно контролировать, контролируя обусловленность матриц  $Y_{ij}$ .

#### 7.6.4. Базис собственных векторов

Если блоки в разбиении (7.6.3) – все размера  $1 \times 1$ , то алгоритм 7.6.3 порождает базис собственных векторов. Как и в случае метода обратных итераций, вычисленные пары собственное значение – собственный вектор являются точными для некоторой близкой матрицы. Широко распространенное правило выбора подходящего метода определения собственных векторов следующее: использовать обратные итерации всякий раз, когда требуется найти меньше чем 25% собственных векторов.

Заметим, однако, что вещественную форму Шура можно использовать для определения отдельных собственных векторов. Пусть матрица

$$Q^T A Q = \begin{bmatrix} T_{11} & u & T_{13} \\ 0 & \lambda & v^T \\ 0 & 0 & T_{33} \end{bmatrix} \begin{bmatrix} k-1 \\ 1 \\ n-1 \end{bmatrix}$$

– верхняя квазитреугольная и  $\lambda \notin \lambda(T_{11}) \cup \lambda(T_{33})$ . Тогда, если мы решим линейные

системы  $(T_{11} - \lambda I)w = -u$  и  $(T_{33} - \lambda I)^T z = -v$ , то векторы

$$x = Q \begin{bmatrix} w \\ 1 \\ 0 \end{bmatrix} \quad \text{и} \quad y = Q \begin{bmatrix} 0 \\ 1 \\ z \end{bmatrix}$$

будут для  $\lambda$  правым и левым собственными векторами соответственно. Заметим, что обусловленность собственного значения  $\lambda$  определяется величиной  $1/s(\lambda) = \sqrt{(1 + w^T w)(1 + z^T z)}$ .

### 7.6.5. Определение блочных жордановых структур

Предположим, что мы вычислили вещественное разложение  $A = QTQ^T$ , определили кластеры «равных» собственных значений и выполнили соответствующую блочную диагонализацию  $T = Y \text{diag}(T_{11}, \dots, T_{qq}) Y^{-1}$ . Как мы видели, это может быть трудной задачей. Однако перед нами встанут еще более значительные численные проблемы, если мы попытаемся установить блочную жорданову структуру каждой матрицы  $T_{ii}$ . Краткое исследование этих трудностей выявит недостатки жорданова разложения.

Предположим для ясности, что спектр  $\lambda(T_{ii})$  – вещественный. Преобразование матрицы  $T_{ii}$  к жордановой форме начинается замещением ее матрицей вида  $C = \lambda I + N$ , где  $N$  – строго верхняя треугольная часть  $T_{ii}$  и  $\lambda$ , скажем, среднее арифметическое ее собственных значений.

Напомним, что порядок жорданова блока  $J(\lambda)$  есть наименьшее неотрицательное целое  $k$ , для которого  $[J(\lambda) - \lambda I]^k = 0$ . Следовательно, если  $p_i = \dim[\text{null}(N^i)]$ , для  $i = 0:n$ , то разность  $p_i - p_{i-1}$  равна числу блоков жордановой формы матрицы  $C$ , которые имеют порядок  $i$  или выше. Конкретный пример позволяет сделать это утверждение понятным и иллюстрирует роль SVD в вычислении жордановой формы.

Предположим, что  $C$  – матрица размера  $7 \times 7$ . Пусть мы вычислили SVD  $U_1^T NV_1 = \sum_1$  и «обнаружили», что матрица  $N$  имеет ранг 3. Если мы располагаем сингулярные числа в порядке возрастания, то это означает, что матрица  $N_1 = V_1^T NV_1$  имеет вид

$$N_1 = \begin{bmatrix} 0 & K \\ 0 & L \\ 4 & 3 \end{bmatrix} \begin{matrix} 4 \\ 3 \end{matrix}.$$

На этом этапе мы знаем, что геометрическая кратность  $\lambda$  равна 4, т. е. жорданова форма матрицы  $C$  имеет 4 блока ( $p_1 - p_0 = 4 - 0 = 4$ ).

Теперь предположим, что  $\tilde{U}_2^T L \tilde{V}_2 = \Sigma_2$  – SVD матрицы  $L$  и мы нашли, что матрица  $L$  имеет единичный ранг. Если мы опять располагали сингулярные числа в порядке возрастания, то матрица  $L_2 = \tilde{V}_2^T L \tilde{V}_2$ , очевидно, имеет следующую структуру:

$$L_2 = \begin{bmatrix} 0 & 0 & a \\ 0 & 0 & b \\ 0 & 0 & c \end{bmatrix}.$$

Однако  $\lambda(L_2) = \lambda(L) = \{0, 0, 0\}$  и, следовательно,  $c = 0$ . Таким образом, если

$$V_2 = \text{diag}(I_4, \tilde{V}_2),$$

то  $N_2 = V_2^T N_1 V_2$  имеет следующий вид

$$N_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & a \\ 0 & 0 & 0 & 0 & 0 & 0 & b \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Помимо того что SVD матрицы  $L$  позволяет ввести больше нулей в верхний треугольник, оно позволяет сделать вывод о размерности ядра матрицы  $N^2$ . Так как

$$N_1^2 = \begin{bmatrix} 0 & KL \\ 0 & L^2 \end{bmatrix} = \begin{bmatrix} 0 & K \\ 0 & L \end{bmatrix} \begin{bmatrix} 0 & K \\ 0 & L \end{bmatrix}$$

и  $\begin{bmatrix} K \\ L \end{bmatrix}$  имеет полный столбцовый ранг, то

$$p_2 = \dim(\text{null}(N^2)) = \dim(\text{null}(N_1^2)) = 4 + \dim(\text{null}(L)) = p_1 + 2.$$

Поэтому мы можем заключить на этом этапе, что жорданова форма матрицы  $C$  имеет по меньшей мере два блока порядка 2 или выше.

Наконец, легко видеть, что  $N_1^3 = 0$ , откуда мы делаем вывод, что существует  $p_3 - p_2 = 7 - 6 = 1$  блока порядка 3 или выше. Если положить  $V = V_1 V_2$ , то разложение

$$V^T C V = \begin{bmatrix} \lambda & 0 & 0 & 0 & \times & \times & \times \\ 0 & \lambda & 0 & 0 & \times & \times & \times \\ 0 & 0 & \lambda & 0 & \times & \times & \times \\ 0 & 0 & 0 & \lambda & \times & \times & \times \\ 0 & 0 & 0 & 0 & \lambda & \times & a \\ 0 & 0 & 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \quad \left. \begin{array}{l} \text{4 блока} \\ \text{порядка 1 или} \\ \text{выше} \end{array} \right\} \quad \left. \begin{array}{l} \text{2 блока} \\ \text{порядка 2 или} \\ \text{выше} \end{array} \right\} \quad \left. \begin{array}{l} \text{1 блок} \\ \text{порядка 3 или} \\ \text{выше} \end{array} \right\}$$

«выявит» жорданову блочную структуру матрицы  $C$ : 2 блока порядка 1, 1 блок порядка 2 и 1 блок порядка 3.

Для того чтобы найти жорданово разложение, необходимо прибегнуть к неортогональным преобразованиям. Мы рекомендуем читателю работы Golub, Wilkinson (1976) и Kågström, Ruhe (1980a, 1980b), в которых описано, как выполнить этот этап редукции.

Рассмотренные выше вычисления на основе SVD в достаточной мере иллюстрируют, что трудное решение о величине ранга приходится принимать на каждом этапе и что окончательное вычисление блочной структуры существенно зависит от этих решений. К счастью, в практических приложениях можно почти всегда использовать вместо жорданова разложения устойчивое разложение Шура.

**Задачи**

**7.6.1.** Предложить полный алгоритм решения вещественной,  $n \times n$ , верхней квазитреугольной системы  $Tx = b$ .

**7.6.2.** Пусть  $U^{-1}AU = \text{diag } (\alpha_1, \dots, \alpha_m)$  и  $V^{-1}BV = \text{diag } (\beta_1, \dots, \beta_n)$ . Показать, что если  $\phi(X) = AX + XB$ , то  $\lambda(\phi) = \{\alpha_i + \beta_j : i = 1:m, j = 1:n\}$ . Какими будут соответствующие собственные векторы? Как использовать эти разложения для решения уравнения  $AX + BX = C$ ?

**7.6.3.** Показать, что если  $Y = \begin{bmatrix} I & Z \\ 0 & I \end{bmatrix}$ , то  $\varphi_2(Y) = [2 + \sigma^2 + \sqrt{4\sigma^2 + \sigma^4}]/2$ , где  $\sigma = \|Z\|_2$ .

**7.6.4.** Получить систему (7.6.5).

**7.6.5.** Предположим, что  $T \in \mathbb{R}^{n \times n}$  – блочная верхняя треугольная матрица, и она разбита на блоки следующим образом:

$$T = \begin{bmatrix} T_{11} & T_{12} & T_{13} \\ 0 & T_{22} & T_{23} \\ 0 & 0 & T_{33} \end{bmatrix},$$

Предположим, что диагональный блок  $T_{22}$  – матрица  $2 \times 2$  с комплексными собственными значениями, не принадлежащими  $\lambda(T_{11})$  и  $\lambda(T_{33})$ . Предложить алгоритм вычисления двумерного вещественного инвариантного подпространства, соответствующего собственным значениям матрицы  $T_{22}$ .

**Замечания и литература к § 7.6**

Большую часть материала, обсуждаемого в этом разделе, можно найти в обзорной статье Golub G. H. and Wilkinson J. H. (1976). “Ill-Conditioned Eigensystems and the Computation of the Jordan Canonical Form”, SIAM Review 18, 578–619.

Статьи, которые рассматривают метод обратных итераций для вычисления собственных векторов, включают

Peters G. and Wilkinson J. H. (1979). “Inverse Iteration, Ill-Conditioned Equations, and Newton’s Method”, SIAM Review 21, 339–360.

Varah J. (1968a). “The Calculation of the Eigenvectors of a General Complex Matrix by Inverse Iteration”, Math. Comp. 22, 785–791.

Varah J. (1968b). “Rigorous Machine Bounds for the Eigensystem of a General Complex Matrix”, Math. Comp. 22, 793–801.

Varah J. (1970). “Computing Invariant Subspace of a General Matrix When the Eigensystem is Poorly Determined”, Math. Comp. 24, 137–149.

Алгольная версия EISPACK подпрограммы обратной итерации приведена в

Peters G. and Wilkinson J. H. (1971). “The Calculation of Specified Eigenvectors by Inverse Iteration” in HACLA, pp. 418–439.

Проблема упорядочения собственных значений в вещественной форме Шуратема работ

Ruhe A. (1970a). “An Algorithm for Numerical Determination of the Structure of a General Matrix”, BIT 10, 196–216.

Stewart G. W. (1976a). “Algorithm 406: HQR3 and EXCHNG: Fortran Subroutines for Calculating and Ordering the Eigenvalues of a Real Upper Hessenberg Matrix”, ACM Trans. Math. Soft. 2, 275–280.

Фортранные программы для вычисления блочно диагональных и жордановых форм описаны в

Bailey C. and Stewart G. W. (1979). “An Algorithm for Computing Reducing Subspaces by Block Diagonalization”, SIAM J. Num. Anal. 16, 359–367.

Demmel J. W. (1983a). “A Numerical Analyst’s Jordan Canonical Form”, Ph. D. thesis, Berkely.

Kägström B. and Ruhe A. (1980a). “An Algorithm for Numerical Computation of the Jordan Normal Form of a Complex Matrix”, ACM Trans. Math. Soft. 6, 398–419.

Kägström B. and Ruhe A. (1980b). "Algorithm 560 JNF: An Algorithm of Numerical Computation of the Jordan Normal Form of a Complex Matrix", ACM Trans. Math. Soft. 6, 437–443.

Статьи, посвященные оцениванию ошибки округления при вычислении собственных значений и/или собственных векторов, включают

Chan S. P. and Parlett B. N. (1977). "Algorithm 517: A Program for Computing the Condition Numbers of matrix Eigenvalues Without Computing Eigenvectors", ACM Trans. Math. Soft. 3, 186–203.

Symm H.J. and Wilkinson J. H. (1980). "Realistic Error Bounds for a Simple Eigenvalue and Its Associated Eigenvector", Numer. Math. 35, 113–126.

Как мы видели, функция `sep (...)` является очень важной для оценки вычисленного инвариантного подпространства. Вопросы, связанные с этой величиной, обсуждаются в

Varah J. (1979). "On the Separation of Two Matrices", SIAM J. Num. Anal. 16, 212–222.

Многочисленные алгоритмы предложены для уравнения Сильвестра  $FX - XG = C$ , но описанные в

Bartels R. H. and Stewart G. W. (1972). "Solution of the Equation  $AX + XB = C$ ", Comm. ACM 15, 820–826.

Golub G. H., Nash S. and Van Loan C. (1979). "A Hessenberg–Schur Method for the Matrix Problem  $AX + XB = C$ ", IEEE Trans. Auto. Cont. AC–24, 909–913.

стоят в ряду наиболее надежных, так как они основаны на ортогональных преобразованиях. Уравнение Ляпунова  $FX + XF^T = -C$ , где  $C$  – неотрицательно определенная матрица, играет очень важную роль в теории управления. Смотри

Barnett S. and Storey C. (1968). "Some Applications of the Lyapunov Matrix Equation", J. Inst. Math. Applic. 4, 33–42.

Некоторые авторы рассматривали обобщенное уравнение Сильвестра, т. е.  $\Sigma F_i X G_i = C$ . В их числе

Lancaster P. (1970). "Explicit Solution of Linear Matrix Equations", SIAM Review 12, 544–566.

Vetter W. J. (1975). "Vector Structures and Solutions of Linear Matrix Equations", Lin. Alg. and Its Applic. 10, 181–188.

Wimmer H. and Ziebur A. D. (1972). "Solving the Matrix Equations  $f_p(A)g_p = C$ ", SIAM Review 14, 318–323.

Некоторые идеи усовершенствования вычислений собственных значений, собственных векторов и инвариантных подпространств можно найти в

Demmel J. W. (1987). "Three Methods for Refining Estimates of Invariant Subspaces", Computing 38, 43–57.

Dongarra J. J., Moler C. B. and Wilkinson J. H. (1983). "Improving the Accuracy of Computed Eigenvalues and Eigenvectors", SIAM J. Numer. Anal. 20, 23–46.

Для проблемы собственных значений были также предложены дешевые методы оценки обусловленности. Смотри

Byers R. (1984). "A Linpack–Style Condition Estimator for the Equation  $AX - XB^T = C$ ", IEEE Trans. Auto. Cont. AC–29, 926–928.

Van Loan C. (1987). "On Estimating the Condition of Eigenvalues and Eigenvectors", Lin. Alg. and Its Applic. 88/89, 715–732.

## 7.7. QZ-метод для $AX = \lambda Bx$

Пусть  $A$  и  $B$  – две матрицы размера  $n \times n$ . Множество всех матриц вида  $A - \lambda B$ :  $\lambda \in \mathbb{C}$  называют пучком. Собственные значения пучка – элементы множества  $\lambda(A, B)$ , определяемого как

$$\lambda(A, B) = \{z \in \mathbb{C} : \det(A - zB) = 0\}.$$

Если  $\lambda \in \lambda(A, B)$  и

$$Ax = \lambda Bx, \quad x \neq 0, \quad (7.7.1)$$

то вектор  $x$  называют собственным вектором пучка матриц  $A - \lambda B$ .

В этом разделе мы даем краткий обзор математических свойств обобщенной проблемы (7.7.1) и представляем устойчивый алгоритм ее решения. Важный случай, когда матрицы  $A$  и  $B$  – симметрические и матрица  $B$  – положительно определенная, рассматривается в § 8.7.2.

### 7.7.1. Основы теории

Первое, что следует сказать об обобщенной проблеме собственных значений, – это то, что  $n$  собственных значений имеются тогда и только тогда, когда  $\text{rank}(B) = n$ . Если  $B$  – матрица неполного ранга, то множество  $\lambda(A, B)$  может быть конечным, пустым или бесконечным:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow \lambda(A, B) = \{1\},$$

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \Rightarrow \lambda(A, B) = \emptyset,$$

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow \lambda(A, B) = \mathbb{C}.$$

Заметим, что если  $0 \neq \lambda \in \lambda(A, B)$ , то  $(1/\lambda) \in \lambda(B, A)$ . Кроме того, если матрица  $B$  невырожденная, то  $\lambda(A, B) = \lambda(B^{-1}A, I) = \lambda(B^{-1}A)$ .

Это последнее наблюдение подсказывает один метод решения  $A - \lambda B$  проблемы для случая, когда матрица  $B$  невырождена:

- Решить уравнение  $BC = A$  относительно  $C$ , используя, скажем, исключение Гаусса с выбором ведущего элемента.
- Используя QR-алгоритм, найти собственные значения матрицы  $C$ .

Заметим, что матрица  $C$  будет подвержена влиянию погрешностей округления порядка  $\|A\|_2 \|B^{-1}\|_2$ . Если матрица  $B$  плохо обусловлена, то это может сделать невозможным точное вычисление любого обобщенного собственного значения, даже тех собственных значений, которые можно считать хорошо обусловленными.

**Пример 7.7.1.** Если

$$A = \begin{bmatrix} 1.746 & 0.940 \\ 1.246 & 1.898 \end{bmatrix} \text{ и } B = \begin{bmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{bmatrix},$$

то  $\lambda(A, B) = \{2, 1.07 \times 10^6\}$ . В арифметике с плавающей запятой с 7 значащими цифрами, мы найдем  $\lambda(fI(AB^{-1})) = \{1.562539, 1.01 \times 10^6\}$ . Плохое качество младшего собственного значения вызвано тем, что  $\kappa_2(B) \approx 2 \times 10^6$ . С другой стороны, мы найдем, что

$$\lambda(I, fI(A^{-1}B)) \approx \{2.000001, 1.06 \times 10^6\}.$$

Точность младшего собственного значения улучшилась потому, что  $\kappa_2(A) \approx 4$ .

**Пример 7.7.1** наводит на мысль искать альтернативный подход к  $A - \lambda B$ -проблеме. Одна из идей – найти хорошо обусловленные матрицы  $Q$  и  $Z$ , такие, что каждая из

матриц

$$A_1 = Q^{-1}AZ, \quad B_1 = Q^{-1}BZ \quad (7.7.2)$$

имеет канонический вид. Заметим, что  $\lambda(A, B) = \lambda(A_1, B_1)$ , так как

$$Ax = \lambda Bx \Leftrightarrow A_1y = \lambda B_1y, \quad x = Zy.$$

Мы будем говорить, что пучки матриц  $A - \lambda B$  и  $A_1 - \lambda B_1$  – эквивалентные, если справедливы равенства (7.7.2) с невырожденными матрицами  $Q$  и  $Z$ .

### 7.7.2. Обобщенное разложение Шура

Как и в обычной проблеме собственных значений  $A - \lambda I$ , в обобщенной проблеме есть возможность выбора канонической формы. Аналогом жордановой формы является разложение Кронекера, в котором обе матрицы  $A$  и  $B$  – блочно-диагональные. Эти блоки похожи на жордановы блоки. Кронекерова каноническая форма приводит к тем же вычислительным проблемам, что и жорданова форма. Однако это разложение позволяет понять математические свойства пучка матриц  $A - \lambda B$ . Смотри Wilkinson (1978) и Demmel, Kågström (1987).

Более привлекательной формой с вычислительной точки зрения является следующее разложение, описанное Молером и Стюартом [Moler, Stewart (1973)].

**Теорема 7.7.1 (Обобщенное разложение Шура).** *Если  $A$  и  $B$  принадлежат  $\mathbb{C}^{n \times n}$ , то существуют унитарные матрицы  $Q$  и  $Z$ , такие, что матрицы  $Q^H AZ = T$  и  $Q^H BZ = S$  являются верхними треугольными. Если при некотором  $k$  элементы  $t_{kk}$  и  $s_{kk}$  оба равны нулю, то  $\lambda(A, B) = \mathbb{C}$ . В противном случае*

$$\lambda(A, B) = \{t_{ii}/s_{ii} : s_{ii} \neq 0\}.$$

**Доказательство.** Пусть  $\{B_k\}$  – последовательность невырожденных матриц, сходящаяся к матрице  $B$ . Для каждого  $k$  пусть  $Q_k^H (A B_k^{-1}) Q_k = R_k$  есть разложение Шура матрицы  $A B_k^{-1}$ . Пусть  $Z_k$  – унитарная матрица, такая, что матрица  $Z_k^H (B_k^{-1} Q_k) \equiv S_k^{-1}$  – верхняя треугольная. Тогда обе матрицы  $Q_k^H AZ_k = R_k S_k$  и  $Q_k^H B_k Z_k = S_k$  будут также верхние треугольные.

Из теоремы Больцано–Вейерштрасса мы знаем, что ограниченная последовательность  $\{(Q_k, Z_k)\}$  имеет сходящуюся подпоследовательность  $\lim(Q_{k_i}, Z_{k_i}) = (Q, Z)$ . Легко показать, что матрицы  $Q$  и  $Z$  – унитарные, а матрицы  $Q^H AZ$  и  $Q^H BZ$  – верхние треугольные. Доказываемые утверждения следуют из тождества

$$\det(A - \lambda B) = \det(QZ^H) \sum_{i=1}^n (t_{ii} - \lambda s_{ii}). \square$$

Если матрицы  $A$  и  $B$  – вещественные, то представляет интерес следующее разложение, которое соответствует вещественному разложению Шура (теорема 7.4.1).

**Теорема 7.7.2 (Обобщенное вещественное разложение Шура).** *Если  $A$  и  $B$  принадлежат  $\mathbb{R}^{n \times n}$ , то существуют ортогональные матрицы  $Q$  и  $Z$ , такие, что матрица  $Q^T AZ$  – верхняя квазитреугольная, а матрица  $Q^T BZ$  – верхняя треугольная.*

**Доказательство.** Смотри Stewart (1972).  $\square$

Оставшаяся часть данного раздела будет посвящена вычислению этого разложения и связанным с этим математическим вопросам.

### 7.7.3. Выводы о чувствительности

Обобщенное разложение Шура проливает свет на вопрос об устойчивости собственных значений  $A - \lambda B$ -проблемы. Ясно, что малые изменения в матрицах  $A$  и  $B$  могут вызвать большие изменения собственного значения  $\lambda_i = t_{ii}/s_{ii}$ , если элемент  $s_{ii}$  мал. Однако, как показал Стюарт [Stewart (1978)], было бы неправильно считать такое собственное значение «плохо обусловленным». Дело в том, что обратная величина  $\mu_i = s_{ii}/t_{ii}$  может быть собственным значением пучка матриц  $\mu A - B$ , ведущим себя очень хорошо. В анализе Стюарта матрицы  $A$  и  $B$  считались симметричными и собственные значения рассматривались скорее как упорядоченные пары, чем как частные. С этой точки зрения более удобно измерять возмущения собственных значений в метрике  $\text{chord}(a, b)$ , определяемой как

$$\text{chord}(a, b) = \frac{|a - b|}{\sqrt{1 + a^2} \sqrt{1 + b^2}}.$$

Стюарт показал, что если  $\lambda$  является отдаленным собственным значением пучка матриц  $A - \lambda B$  и  $\lambda_\epsilon$  – соответствующее собственное значение возмущенного пучка матриц  $\tilde{A} - \lambda \tilde{B}$  с  $\|A - \tilde{A}\|_2 \approx \|B - \tilde{B}\|_2 \approx \epsilon$ , то

$$\text{chord}(\lambda, \lambda_\epsilon) \leq \frac{\epsilon}{(y^H Ax)^2 + (y^H Bx)^2} + O(\epsilon^2),$$

где  $x$  и  $y$  имеют единичную вторую норму и удовлетворяют равенствам  $Ax = \lambda Bx$  и  $y^H = \lambda y^H B$ . Заметим, что в знаменатель этой верхней оценки матрицы  $A$  и  $B$  входят симметрично. «В самом деле плохо обусловленные» собственные значения это те, для которых этот знаменатель мал.

Экстремальный случай  $t_{kk} = s_{kk} = 0$  при некотором  $k$  был рассмотрен Уилкисоном [Wilkinson (1979)]. Он сделал интересное наблюдение, что когда это имеет место, остальные частные  $t_{ii}/s_{ii}$  могут принимать произвольные значения.

### 7.7.4. Хессенбергово-треугольная форма

Первый шаг вычисления обобщенного разложения Шура пары  $(A, B)$  состоит в преобразовании матрицы  $A$  к верхнему хессенбергову виду и матрицы  $B$  к верхнему треугольному виду посредством ортогональных преобразований. Сначала мы найдем ортогональную матрицу  $U$ , такую, что матрица  $U^T B$  – верхняя треугольная. Конечно, для того чтобы сохранить собственные значения, мы должны обновить матрицу  $A$  точно таким же образом. Давайте проследим, что произойдет в случае  $n = 5$ .

$$A = U^T A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}, B = U^T B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Затем мы преобразуем  $A$  к верхнему хессенбергову виду, сохраняя верхний треугольный вид матрицы  $B$ . Сначала определим вращение Гивенса  $Q_{45}$ , обнуляющее элемент  $a_{51}$ :

$$A = Q_{45}^T A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}, B = Q_{45}^T B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

Ненулевой элемент, появившийся в позиции (5,4) матрицы  $B$ , может быть обнулен последующим умножением на подходящую матрицу вращения Гивенса  $Z_{45}$ :

$$A = AZ_{45} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}, B = BZ_{45} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Точно так же вводятся нули в позиции (4,1) и (3,1) матрицы  $A$ :

$$A = Q_{34}^T A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}, B = Q_{34}^T B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

$$A = AZ_{34} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{bmatrix}, B = BZ_{34} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

$$A = Q_{23}^T A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{bmatrix}, B = Q_{23}^T B = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \end{bmatrix}$$

$$A = AZ_{23} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{bmatrix}, B = BZ_{23} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

Матрица  $A$  теперь верхняя хессенбергова в своем первом столбце. Преобразование завершает обнуление элементов  $a_{52}$ ,  $a_{42}$  и  $a_{32}$ . Как показано выше, требуется два

ортогональных преобразования для каждого обнуляемого элемента  $a_{ij}$  — одно для обнуления и другое для сохранения треугольности матрицы  $B$ . Можно использовать как вращения Гивенса, так и модифицированные преобразования Хаусхолдера размера  $2 \times 2$ . В целом имеем

**Алгоритм 7.7.1 (Хессенбергово-треугольное преобразование).** Для заданных матриц  $A$  и  $B$ , принадлежащих  $\mathbb{R}^{n \times n}$ , следующий алгоритм записывает в матрицу  $A$  верхнюю хессенбергову матрицу  $Q^T A Z$  и в матрицу  $B$  — верхнюю треугольную матрицу  $Q^T B Z$ , где обе матрицы  $Q$  и  $Z$  — ортогональные.

Используя алгоритм 5.2.1, записать в матрицу  $B$  матрицу  $Q^T B = R$ , где  $Q$  — ортогональная, а  $R$  верхняя треугольная матрицы.

```

 $A = Q^T A$ 
for  $j = 1:n - 2$ 
  for  $i = n:-1:j + 2$ 
    {Обнулить  $a_{ij}$ }
     $[c, s] = \text{givens}(A(i-1, j), A(i, j))$ 
     $A(i-1:i, j:n) = \text{row\_rot}(A(i-1:i, j:n), c, s)$ 
     $B(i-1:i, i-1:n) = \text{row\_rot}(B(i-1:i, i-1:n), c, s)$ 
     $[c, s] = \text{givens}(B(i, i-1), B(i, i))$ 
     $B(1:i, i-1:i) = \text{col\_rot}(B(1:i, i-1:i), c, s)$ 
     $A(1:n, i-1:i) = \text{col\_rot}(A(1:n, i-1:i), c, s)$ 
  end
end
```

Этот алгоритм требует примерно  $8n^3$  флоков. Накопление матриц  $Q$  и  $Z$  потребует примерно  $4n^3$  и  $3n^3$  флоков соответственно.

Преобразование пучка матриц  $A - \lambda B$  к хессенбергово-треугольной форме служит предварительным преобразованием для обобщенных QR-итераций, известных как QZ-итерации, которые мы опишем ниже.

**Пример 7.7.3.** Если

$$A = \begin{bmatrix} 10 & 1 & 2 \\ 1 & 2 & -1 \\ 1 & 1 & 2 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

и ортогональные матрицы  $Q$  и  $Z$  выбраны следующим образом:

$$Q = \begin{bmatrix} -0.1231 & -0.9917 & 0.0378 \\ -0.4924 & 0.0279 & -0.8699 \\ -0.8616 & 0.1257 & 0.4917 \end{bmatrix} \quad \text{и} \quad Z = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & -0.8944 & -0.4472 \\ 0.0000 & 0.4472 & -0.8944 \end{bmatrix},$$

то матрицы  $A_1 = Q^T A Z$  и  $B_1 = Q^T B Z$  имеют вид

$$A_1 = \begin{bmatrix} -2.5849 & 1.5413 & 2.4221 \\ -9.7631 & 0.0874 & 1.9239 \\ 0.0000 & 2.7233 & -0.7612 \end{bmatrix} \quad \text{и} \quad B_1 = \begin{bmatrix} -8.1240 & 3.6332 & 14.2024 \\ 0.0000 & 0.0000 & 1.8739 \\ 0.0000 & 0.0000 & 0.7612 \end{bmatrix}.$$

## 7.7.5. Исчерпывание

При описании QZ-итераций мы можем предполагать, не нарушая общности, что  $A$  — неприводимая верхняя хессенбергова матрица и что  $B$  — невырожденная верхняя

треугольная матрица. Первое из этих утверждений очевидно, так как если  $a_{k+1,k} = 0$ , то

$$A - \lambda B = \begin{bmatrix} A_{11} - \lambda B_{11} & A_{12} - \lambda B_{12} \\ 0 & A_{22} - \lambda B_{22} \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix}$$

и мы можем продолжать решать две проблемы меньшего размера  $A_{11} - \lambda B_{11}$  и  $A_{22} - \lambda B_{22}$ . С другой стороны, если  $b_{kk} = 0$  при некотором  $k$ , то можно ввести нулевой элемент в позицию  $(n, n-1)$  матрицы  $A$  и тем самым осуществить исчерпывание. Проиллюстрируем это следующим примером, где  $n = 5$  и  $k = 3$ :

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Используя вращения Гивенса, нуль на диагонали матрицы  $B$  можно «столкнуть» на позицию  $(5,5)$  следующим образом

$$A = Q_{34}^T A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad B = Q_{34}^T B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

$$A = AZ_{23} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad B = BZ_{23} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}$$

$$A = Q_{45}^T A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}, \quad B = Q_{45}^T B = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A = AZ_{34} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad B = BZ_{34} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A = AZ_{45} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}, \quad B = BZ_{45} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Этот метод перегонки нуля вполне общий и может быть использован для обнуления элемента  $a_{n,n-1}$  независимо от того, где на диагонали матрицы  $B$  появился нулевой элемент.

### 7.7.6. QZ-шаг

Теперь мы подготовлены к описанию QZ-шага. Основная идея заключается в том, чтобы обновить матрицы  $A$  и  $B$  следующим образом:

$$(\bar{A} - \lambda \bar{B}) = Q^T (A - \lambda B) \bar{Z},$$

где матрица  $\bar{A}$  – верхняя хессенбергова,  $\bar{B}$  – верхняя треугольная,  $\bar{Q}$  и  $\bar{Z}$  – ортогональные и матрица  $\bar{A}\bar{B}^{-1}$  в основном та матрица, которая получилась бы, если QR-шаг Фрэнсиса (алгоритм 7.5.2) применить к матрице  $AB^{-1}$ . Это можно сделать при помощи ловкой перегонки нуля и апелляции к теореме о неявном  $Q$ .

Пусть матрица  $M = AB^{-1}$  (верхняя хессенбергова) и пусть  $v$  – первый столбец матрицы  $(M - aI)(M - bI)$ , где  $a$  и  $b$  – собственные значения нижней подматрицы  $2 \times 2$  матрицы  $M$ . Заметим, что вектор  $v$  может быть вычислен за  $O(1)$  флоков. Если  $P_0$  – матрица Хаусхолдера, такая, что вектор  $P_0 v$  коллинеарен  $e_1$ , то

$$A = P_0 A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix},$$

$$B = P_0 B = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Теперь идея состоит в том, чтобы привести эти матрицы к хессенбергово-треугольной форме, перегоняя нежелательные ненулевые элементы в нижнюю часть диагонали.

С этой целью мы сначала найдем пару матриц Хаусхолдера  $Z_1$  и  $Z_2$ , обнуляющих элементы  $b_{31}$ ,  $b_{32}$  и  $b_{21}$ :

$$A = AZ_1 Z_2 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix},$$

$$B = BZ_1Z_2 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Затем используем матрицу Хаусхолдера  $P_1$  для обнуления  $a_{31}$  и  $a_{41}$ :

$$A = P_1 A = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix},$$

$$B = P_1 B = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Заметим, что после этого шага нежелательные ненулевые элементы оказались сдвинутыми вниз и вправо от их первоначального положения. Это иллюстрирует типичный шаг QZ-итераций. Заметим, что матрица  $Q = Q_0 Q_1 \dots Q_{n-2}$  имеет тот же самый первый столбец, что и матрица  $Q_0$ . Кстати, первая матрица Хаусхолдера определяется однозначно, мы можем применить теорему о неявном  $Q$  и утверждать, что матрица  $AB^{-1} = Q^T(AB^{-1})Q$  – действительно, в основном, та матрица, которую можно получить, применяя итерацию Фрэнсиса к матрице  $M = AB^{-1}$  непосредственно. В целом мы имеем

**Алгоритм 7.7.2 (QZ-шаг).** Для заданной неприводимой верхней хессенберговой матрицы  $A \in \mathbb{R}^{n \times n}$  и невырожденной верхней треугольной матрицы  $B \in \mathbb{R}^{n \times n}$  следующий алгоритм записывает в матрицу  $A$  верхнюю хессенбергову матрицу  $Q^T AZ$  и в  $B$  – верхнюю треугольную матрицу  $Q^T BZ$ , где матрицы  $Q$  и  $Z$  – ортогональные, и  $Q$  имеет тот же самый первый столбец, что и ортогональное преобразование подобия в алгоритме 7.5.1, если его применить к матрице  $AB^{-1}$ .

Положить  $M = AB^{-1}$  и вычислить  $(M - aI)(M - bI)e_1 = (x, y, z, 0, \dots, 0)^T$ , где  $a$  и  $b$  – собственные значения нижней подматрицы  $2 \times 2$  матрицы  $M$ .

for  $k = 1 : n - 2$

    Найти матрицу Хаусхолдера  $Q_k$ , такую, что  $Q_k [x \ y \ z]^T = [* \ 0 \ 0]^T$ .

$A = \text{diag}(I_{k-1}, Q_k, I_{n-k-2})A$

$B = \text{diag}(I_{k-1}, Q_k, I_{n-k-2})B$

Найти матрицу Хаусхолдера  $Z_{k1}$ , такую, что

$$[b_{k+2,k} \ b_{k+2,k+1} \ b_{k+2,k+2}] Z_{k1} = [0 \ 0 \ *]$$

$$A = A \operatorname{diag}(I_{k-1}, Z_{k1}, I_{n-k-2})$$

$$B = B \operatorname{diag}(I_{k-1}, Z_{k1}, I_{n-k-2})$$

Найти матрицу Хаусхолдера  $Z_{k2}$ , такую, что

$$[b_{k+1,k} \ b_{k+1,k+1}] Z_{k2} = [0 \ *].$$

$$A = A \operatorname{diag}(I_{k-1}, Z_{k2}, I_{n-k-1})$$

$$B = B \operatorname{diag}(I_{k-1}, Z_{k2}, I_{n-k-1})$$

$$x = a_{k+1,k}; y = a_{k+1,k}$$

if  $k < n - 2$

$$z = a_{k+3,k}$$

end

end

Найти матрицу Хаусхолдера  $Q_{n-1}$ , такую, что  $Q_{n-1} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$

$$A = \operatorname{diag}(I_{n-2}, Q_{n-1}) A$$

$$B = \operatorname{diag}(I_{n-2}, Q_{n-1}) B$$

Найти матрицу Хаусхолдера  $Z_{n-1}$ , такую, что

$$[b_{n,n-1} \ b_{nn}] Z_{n-1} = [0 \ *]$$

$$A = A \operatorname{diag}(I_{n-2}, Z_{n-1})$$

$$B = B \operatorname{diag}(I_{n-2}, Z_{n-1})$$

Этот алгоритм требует  $22n^2$  флоков. Матрицы  $Q$  и  $Z$  могут быть накоплены за еще  $8n^2$  флоков и  $13n^2$  флоков соответственно.

### 7.7.7. QZ-процесс в целом

Применяя последовательность QZ-шагов к хессенбергово-треугольному пучку матриц  $A - \lambda B$ , можно привести матрицу  $A$  к квазитреугольному виду. Делая это, необходимо контролировать поддиагональ матрицы  $A$  и диагональ матрицы  $B$ , для того чтобы осуществлять расщепление всякий раз, когда это возможно. Полный процесс, благодаря Молеру и Стюарту [Moler, Stewart (1973)] имеет следующий вид:

**Алгоритм 7.7.3.** Для заданных  $A \in \mathbb{R}^{n \times n}$  и  $B \in \mathbb{R}^{n \times n}$  следующий алгоритм вычисляет ортогональные матрицы  $Q$  и  $Z$ , такие, что матрица  $Q^T AZ = T$  — верхняя квазитреугольная, а матрица  $Q^T BZ = S$  — верхняя треугольная. Матрицу  $A$  замещает матрица  $T$ , а матрица  $B$  — матрица  $S$ .

Используя алгоритм 7.7.1, записать в  $A$  матрицу  $Q^T AZ$  (верхняя хессенбергова) и в матрицу  $B$  — матрицу  $Q^T BZ$  (нижняя треугольная).

until  $q = n$

Положить равными нулю все поддиагональные элементы матрицы  $A$ , удовлетворяющие неравенству

$$|a_{i,i-1}| \leq \varepsilon (|a_{i-1,i-1}| + |a_{ii}|)$$

Найти наибольшее неотрицательное  $q$  и наименьшее неотрицательное  $p$ , такие, что если

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \\ p & n-p-q & q \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$

то матрица  $A_{33}$  – верхняя квазитреугольная, а матрица  $A_{22}$  – неприводимая верхняя Хесенбергова.

В соответствии с этим разбить на блоки матрицу  $B$ :

$$B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ 0 & B_{22} & B_{23} \\ 0 & 0 & B_{33} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$

$$p \quad n-p-q \quad q$$

**if**  $q < n$

**if**  $B$  вырождена

Обнулить  $a_{n-q, n-q-1}$

**else**

Применить алгоритм 7.7.2 к матрицам  $A_{22}$  и  $B_{22}$

$$A = \text{diag}(I_p, Q, I_q)^T A \text{diag}(I_p, Q, I_q)$$

$$B = \text{diag}(I_p, Q, I_q)^T B \text{diag}(I_p, Q, I_q)$$

**end**

**end**

**end**

Этот алгоритм требует  $30n^3$  флопов. Если желательно иметь матрицу  $Q$ , то необходимо дополнительно  $16n^3$  флопов. Если требуется матрица  $Z$ , то нужно дополнительно  $20n^3$  флопов. Эти оценки вычислительных затрат основаны на наблюдении, что необходимо примерно два шага QZ-итераций на одно собственное значение. Таким образом, свойства сходимости QZ-алгоритма те же, что и у QR-алгоритма. На скорость сходимости QR-алгоритма не влияет дефект ранга матрицы  $B$ .

Можно показать, что вычисленные матрицы  $S$  и  $T$  удовлетворяют равенствам

$$Q_0^T(A + E)Z_0 = T, \quad Q_0^T(B + F)Z_0 = S,$$

где  $Q_0$  и  $Z_0$  – точно ортогональные матрицы, а  $\|E\|_2 \approx \mathbf{u}\|A\|_2$  и  $\|F\|_2 \approx \mathbf{u}\|B\|_2$ .

**Пример 7.7.5.** Если QZ-алгоритм применить к матрицам

$$A = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \\ 4 & 4 & 5 & 6 & 7 \\ 0 & 3 & 6 & 7 & 8 \\ 0 & 0 & 2 & 8 & 9 \\ 0 & 0 & 0 & 1 & 10 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 & -1 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

то поддиагональные элементы матрицы  $A$  будут сходиться следующим образом:

Итерация	$O( h_{21} )$	$O( h_{32} )$	$O( h_{43} )$	$O( h_{54} )$
1	$10^0$	$10^4$	$10^0$	$10^{-1}$
2	$10^0$	$10^0$	$10^0$	$10^{-1}$
3	$10^0$	$10^1$	$10^{-1}$	$10^{-3}$
4	$10^0$	$10^0$	$10^{-1}$	$10^{-8}$
5	$10^0$	$10^1$	$10^{-1}$	$10^{-16}$
6	$10^0$	$10^0$	$10^{-2}$	
7	$10^0$	$10^{-1}$	$10^{-4}$	
8	$10^1$	$10^{-1}$	$10^{-8}$	
9	$10^0$	$10^{-1}$	$10^{-19}$	
10	$10^0$	$10^{-2}$		
11	$10^{-1}$	$10^{-4}$		
12	$10^{-2}$	$10^{-11}$		
13	$10^{-3}$	$10^{-27}$		
14				

### 7.7.8. Вычисление обобщенных инвариантных подпространств

Многие алгоритмы вычисления инвариантных подпространств, обсуждавшиеся в § 7.6, переносятся на обобщенную проблему собственных значений. Например, приближенный собственный вектор можно найти посредством обратной итерации:

$q^{(0)} \in \mathbb{C}^{n \times n}$  задан.

for  $k = 1, 2, \dots$

Решить систему  $(A - \mu B)z^{(k)} = Bq^{(k-1)}$

Нормализовать:  $q^{(k)} = z^{(k)} / \|z^{(k)}\|_2$

$\lambda^{(k)} = [q^{(k)}]^H A q^{(k)} / [q^{(k)}]^H A q^{(k)}$

end

Когда матрица  $B$  невырожденная, это эквивалентно применению алгоритма (7.6.1) с матрицей  $B^{-1}A$ . Как правило, требуется только одна итерация, если  $\mu$  – приближенное собственное значение, найденное QZ-алгоритмом. Обратное итерирование с хессенбергово-треугольным пучком позволяет избежать дорогостоящего накопления Z-преобразований в течение QZ-итераций.

Подобно понятию инвариантного подпространства матрицы, существует понятие понижающего подпространства пучка матриц  $A - \lambda B$ . В частности, говорят, что  $k$ -мерное подпространство  $S \subseteq \mathbb{R}^n$  является понижающим для пучка  $A - \lambda B$ , если подпространство  $\{Ax + By : x, y \in S\}$  имеет размерность  $k$  или меньше. Заметим, что столбцы матрицы  $Z$  в обобщенном разложении Шура определяют семейство понижающих подпространств, так как если  $Q = [q_1, \dots, q_n]$  и  $Z = [z_1, \dots, z_n]$ , то  $\text{span}\{Az_1, \dots, Az_k\} \subseteq \text{span}\{q_1, \dots, q_k\}$  и  $\text{span}\{Bz_1, \dots, Bz_k\} \subseteq \text{span}\{q_1, \dots, q_k\}$ . Свойства понижающих подпространств и их поведение под воздействием возмущений описаны в Stewart (1972).

#### Задачи

7.7.1. Предположим, что  $A$  и  $B$  принадлежат  $\mathbb{R}^{n \times n}$  и что

$$U^T B V = \begin{bmatrix} D & 0 \\ 0 & 0 \\ r & n-r \end{bmatrix} \begin{matrix} r \\ n-r \\ r \\ n-r \end{matrix}, \quad U = [U_1 \quad U_2], \quad V = [V_1 \quad V_2]$$

есть SVD матрицы  $B$ , где  $D$  – матрица размера  $r \times r$ , а  $r = \text{rank}(B)$ . Показать, что если  $\lambda(A, B) = \mathbb{C}$ , то матрица  $U_2^T A V_2$  вырождена.

7.7.2. Определим функционал  $F: \mathbb{R}^n \rightarrow \mathbb{R}$  как

$$F(x) = \frac{1}{2} \|Ax - \frac{x^T B^T A x}{x^T B^T B x} Bx\|_2^2,$$

где  $A$  и  $B$  принадлежат  $\mathbb{R}^{n \times n}$ . Показать, что если  $\nabla F(x) = 0$ , то вектор  $Ax$  коллинеарен вектору  $Bx$ .

7.7.3. Пусть  $A$  и  $B$  принадлежат  $\mathbb{R}^{n \times n}$ . Предложить алгоритм вычисления ортогональных матриц  $Q$  и  $Z$ , таких, что матрица  $Q^T A Z$  верхняя хессенбергова, а матрица  $Z^T B Q$  верхняя треугольная.

7.7.4. Показать, что если  $\mu \notin \lambda(A, B)$ , то матрицы  $A_1 = (A - \mu B)^{-1} A$  и  $B_1 = (A - \mu B)^{-1}$  перестановочны.

7.7.5. Пусть  $A \in \mathbb{C}^{n \times n}$ ,  $B \in \mathbb{C}^{n \times n}$  и матрица  $A - \mu B$  невырожденная при некотором  $\mu \in \mathbb{C}$ . Предположим, что матрица  $B$  невырожденная. Показать, что  $\lambda \in \lambda(A, B)$  тогда и только тогда, когда  $\lambda \in \lambda(B, B(A - \mu B)^{-1} B)$ .

### Замечания и литература к § 7.7

Математические аспекты обобщенной проблемы собственных значений рассматриваются в работах

Erdeiyi I. (1967). "On the Matrix Equation  $Ax = \lambda Bx$ ", J. Math. Anal. and Applic. 17, 119–132.

Гантмахер Ф. Р. Теория матриц, т. 2, М.: Наука, 1967.

Turnbill H. W. and Aitken A. C. (1961). An Introduction to the Theory of Canonical Matrices, Dover, New York.

Хорошей книгой, охватывающей многие аспекты  $A - \lambda B$  проблемы, является

Kågström B. and Ruhe A. (1983). Matrix Pencils, Proc. Pite Havsbad, 1982, Lecture Notes in Mathematics 973, Springer-Verlag, New York and Berlin.

Стюарт рассматривал вопросы чувствительности собственных значений в

Stewart G. W. (1972). "On the Sensitivity of the Eigenvalue Problem  $Ax = \lambda Bx$ ", SIAM J. Num. Anal. 9, 669–686.

Stewart G. W. (1973b). "Error and Perturbation Bounds for Subspaces Associated with Certain Eigenvalue Problems", SIAM Review 15, 727–764.

Stewart G. W. (1975b). "Gershgorin Theory for the Generalized Eigenvalue Problem  $Ax = \lambda Bx$ ", Math. Comp. 29, 600–606.

Stewart G. W. (1978). "Perturbation Theory for the Generalized Eigenvalue Problem", in Recent Advance in Numerical Analysis, ed. C. de Boor and G. H. Golub, Academic Press, New York.

В некоторых прикладных задачах появляются обобщенные проблемы собственных значений с прямоугольными матрицами. Смотри

Thompson G. L. and Weil R. L. (1970). "Reducing the Rank of  $A - \lambda B$ ", Proc. Amer. Math. Sec. 26, 548–554.

Thompson G. L. and Weil R. L. (1972). "Roots of Matrix Pencils  $Ay = \lambda By$ : Existence, Calculations, and Relations to Game Theory", Lin. Alg. and Its Applic. 5, 207–226.

Неортогональные вычислительные процедуры для обобщенной проблемы собственных значений предложены в

Kublanovskaja V. N. and Faddeeva V. N. (1964). "Computational Methods for the Solution of a Generalized Eigenvalue Problem", Amer. Math. Soc. Transl. 2, 271–290.

Peters G. and Wilkinson J. H. (1970a). " $Ax = \lambda Bx$  and the Generalized Eigenproblem", SIAM J. Numer. Anal. 7, 479–492.

QZ алгоритм включен в EISPACK и был впервые описан в

Moler C. B. and Stewart G. W. (1973). "An Algorithm for Generalized Matrix Eigenvalue Problems", SIAM J. Num. Anal. 10, 241–256.

Улучшения этого первоначального алгоритма обсуждаются

Kaufman K. (1977). "Some Thoughts on the QZ Algorithm for Solving the Generalized Eigenvalue Problem", ACM Trans. Math. Soft. 3, 65–75.

Ward R. C. (1975). "The Combination Shift QZ Algorithm", SIAM J. Num. Anal. 12, 835–53.

Ward R. C. (1981). "Balancing the Generalized Eigenvalue Problem", SIAM J. Sci. and Stat. Comp. 2, 141–152.

Так же, как и хессенбергово разложение, хессенбергово-треугольное разложение, которое служит предварительным алгоритмом для QZ алгоритма, важно само по себе. Смотри

Enright W. and Serbin S. (1978). "A Note on the Efficient Solution of Matrix Pencil Systems", BIT 18, 276–281.

Метод, похожий на QZ-алгоритм, но основанный на преобразованиях Гаусса, предложен в

Kaufman K. (1974). "The LZ Algorithm to Solve the Generalized Eigenvalue Problem", SIAM J. Num. Anal. 11, 997–1024,

в то время как в

Van Loan C. F. (1975). "A General Matrix Eigenvalue Algorithm", SIAM J. Num. Anal. 12, 819–834.

рассматривается обобщение QZ-алгоритма, которое работает с проблемой  $A^T Cx = \lambda B^T Dx$  без обращения или формирования матриц  $A^T C$ ,  $B^T D$ .

Поведение QZ-алгоритма для пучков матриц  $A - \lambda B$ , которые всегда близки к вырожденным, обсуждается в

Wilkinson J. H. (1979). "Kronecker's Canonical Form and the QZ Algorithm", Lin. Alg. and Its Appl., 28, 285–303.

Другие подходы к этой проблеме включают

Jennings A. and Osborne M. B. (1977). "Generalized Eigenvalue Problems for Certain Unsymmetric Band Matrices", Lin. Alg. and Its Appl. 29, 139–150.

Kublanovskaya V. N. (1984). "AB Algorithm and Its Modifications for the Spectral Problem of Linear Pencils of Matrices", Numer. Math. 43, 329–343.

Rodrigue G. (1973). "A Gradient Method for the Matrix Eigenvalue Problem  $Ax = Bx$ ", Numer. Math. 22, 1–16.

Schwartz H. R. (1974). "The Method of Coordinate Relaxation for  $(A - \lambda B)x = 0$ ", Num. Math. 23, 135–152.

Обобщенная  $Ax = \lambda Bx$  проблема является центральной для многих важных приложений теории управления. Смотри

Arnold W. F. and Laub A. J. (1984). "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations", Proc. IEE 72, 1746–1754.

Demmel J. W. and Kågström B. (1988). "Accurate Solutions of Ill-Posed Problems in Control Theory", SIAM J. Matrix Anal. Appl. 126–145.

Van Dooren P. (1981). "A Generalized Eigenvalue Approach for Solving Riccati Equations", SIAM J. Sci. and Stat. Comp. 2, 121–135.

Van Dooren P. (1981). "The Generalized Eigenstructure Problem in Linear System Theory", IEEE Trans. Auto. Cont. AC-26, 111–128.

В самом деле, различные приложения теории управления заставили глубже взглянуть на Кронекерову структуру пучка матриц, что подтверждают следующие работы:

Demmel J. W. (1983b). "The Condition Number of Equivalence Transformations that Block Diagonalize Matrix Pencils", SIAM J. Numer. Anal. 20, 599–610.

Demmel J. W. and Kågström B. (1987). "Computing Stable Eigendecompositions of Matrix Pencils", Linear Alg. and Its Appl. 88/89, 139–186.

Kågström B. (1985). "The Generalized Singular Value Decomposition and the General  $A - \lambda B$  Problem", BIT 24, 568–583.

Kågström B. (1986). "RGSVD: An Algorithm for Computing the Kronecker Structure and Reducing Subspaces of Singular  $A - \lambda B$  Pencils", SIAM J. Sci. and Stat. Comp. 7, 185–211.

Kågström B. and Westin L. (1987). "GSYLV – Fortran Routines for the Generalized Schur Method with  $\text{dif}^{-1}$  estimators for Solving the Generalized Sylvester Equation", Report UMINF-132.86, Inst. of Inf. Proc., University of Umea, S-901 87 Umea, Sweden.

Van Dooren P. (1979). "The Computation of Kronecker's Canonical Form of a Singular Pencil", Lin. Alg. and Its Appl. 27, 103–140.

Van Dooren P. (1983). "Reducing Subspaces: definitions, properties and Algorithms", in Matrix Pencils, eds. B. Kågström and A. Ruhe, Springer-Verlag, New York, 1983, 58–73.

Wilkinson J. H. (1978). "Linear Differential Equations and Kronecker's Canonical Form", in Recent Advances in Numerical Analysis, ed. C. de Boor and G. H. Golub, Academic Press, New York, pp. 231–265.

Работы над параллельными алгоритмами для обобщенной проблемы собственных значений начата недавно. Смотри

Kågström B., Nyström L. and Poromaa P. (1987). "Parallel Algorithms for Solving the Triangular Sylvester Equation on a Hypercube Multiprocessor", Report UMINF-136.87, Inst. of Inf. Proc., University of Umea, S-901 87, Umea, Sweden.

Kågström B., Nyström L. and Poromaa P. (1988). "Parallel Shared Memory Algorithms for Solving the Triangular Sylvester Equation", Report UMINF-155.88, Inst. of Inf. Proc., University of Umea, S-901 87, Umea, Sweden.

### Добавления при переводе

Среди работ, опубликованных на русском языке, наиболее полно различные вопросы, связанные с обобщенной проблемой собственных значений, в том числе нелинейной, изложены в следующей серии:

Кублановская В. Н., Хазанов В. Б. Спектральные задачи для пучков матриц. Методы и алгоритмы. I—Препринт ЛОМИ, Р. 2-88. Ленинград, 1988, 57 с.

Кублановская В. Н., Хазанов В. Б. Спектральные задачи для пучков матриц. Методы и алгоритмы. II—Препринт ЛОМИ Р-33-88, Ленинград, 1988, 37 с.

Кублановская В. Н., Хазанов В. Б. Спектральные задачи для пучков матриц. Методы и алгоритмы. III—Препринт ЛОМИ Р-4-88, Ленинград, 1988, 54 с.

Построению алгоритмов, обеспечивающих гарантированную точность в условиях приближенных вычислений, посвящены работы

Годунов С. К. Задача о дихотомии спектра матрицы//Сиб. мат. журн.—1986, Т. 27, № 5.—С. 24—25.

Булгаков А. Я., Годунов С. К. Круговая дихотомия матричного спектра. Новосибирск, 1987.—34 с. (Препринт/АН СССР. Сиб. отд. Ин-т математики № 5.)

Малышев А. Н. Гарантиированная точность в спектральных задачах линейной алгебры//Вычислительные методы линейной алгебры.—Новосибирск: Наука. Сиб. отд., 1990, с. 19—104.

Гарантиированная точность решения систем линейных уравнений/Годунов С. К., Антонов А. Г., Кирилюк О. П., Костин В. И.—Новосибирск: Наука. Сиб. отд., 1988.

## Глава 8

# Симметричная проблема собственных значений

Теория возмущения и алгоритмические разработки из предыдущей главы значительно упрощаются, когда матрица  $A$  симметрична. Симметричная проблема собственных значений с ее богатой математической структурой поистине является одной из наиболее притягательных проблем вычислительной алгебры.

Мы начнем наше знакомство с короткого обсуждения математических результатов, лежащих в основе симметричной проблемы собственных значений. В § 8.2 мы специализируем алгоритмы из § 7.4 и 7.5 и получаем изящный симметричный QR-алгоритм. Вариант этой процедуры, позволяющий вычислять сингулярное разложение, подробно рассматривается в § 8.3. Поскольку собственные значения симметричной матрицы могут быть определены различными способами, существует множество QR-алгоритмов. Некоторые из этих методов описаны в § 8.4.

Одним из самых первых матричных алгоритмов, появившихся в литературе, является метод Якоби для симметричной проблемы собственных значений. Интерес к этой процедуре возобновился благодаря параллельным вычислениям и поэтому мы отвели этому методу значительное место в § 8.5. Другой высокопараллельный метод для проблемы собственных значений, вариант метода «разделяй и властвуй», рассмотрен в § 8.6. Его можно использовать для трехдиагональной проблемы, и он особенно интересен, так как успешно конкурирует с трехдиагональными QR-итерациями даже в условиях однопроцессорных вычислений.

В конце раздела мы рассматриваем  $A - \lambda B$  проблему для важного случая, когда матрица  $A$  – симметричная, а матрица  $B$  – симметричная положительно определенная. Хотя не существует подходящих аналогов QZ-алгоритма для этой обобщенной проблемы собственных значений со специальной структурой, существует несколько удачных методов, которые могут быть использованы. Также обсуждается обобщенная проблема сингулярных значений.

Большую часть материала этой главы можно найти в [Parlett, SEP].

## 8.1. Математические основы

Симметричная проблема собственных значений имеет изящную и богатую теорию. Наиболее важные аспекты этой теории рассматриваются в данном разделе.

### 8.1.1. Собственные значения симметричных матриц

Симметричность упрощает вещественную проблему собственных значений  $Ax = \lambda x$  в двух направлениях. Она подразумевает, что все собственные значения матрицы  $A$  вещественные и что существует ортогональный базис собственных векторов. Эти свойства вытекают из следующей теоремы.

**Теорема 8.1.1 (Симметричное вещественное разложение Шура).** Если  $A$  – вещественная симметричная матрица размера  $n \times n$ , то существует вещественная ортогональная матрица  $Q$ , такая, что

$$Q^T A Q = \text{diag}(\lambda_1, \dots, \lambda_n).$$

*Доказательство.* Пусть  $Q^T A Q = T$  есть вещественное разложение Шура (теорема 7.4.1) матрицы  $A$ . Поскольку матрица  $T$  к тому же симметричная, из сделанного предположения следует, что она должна быть прямой суммой матриц размера  $1 \times 1$  и  $2 \times 2$ . Однако легко проверить, что симметричная матрица размера  $2 \times 2$  не может иметь комплексных собственных значений. Следовательно, матрица  $T$  не может иметь на диагонали блоков размера  $2 \times 2$ .  $\square$

**Пример 8.1.1.** Если

$$A = \begin{bmatrix} 6.8 & 2.4 \\ 2.4 & 8.2 \end{bmatrix} \text{ и } Q = \begin{bmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{bmatrix},$$

то матрица  $Q$  ортогональная и  $Q^T A Q = \text{diag}(10,5)$ .

## 8.1.2. Теорема о минимаксе и некоторые следствия

Для  $A^T = A \in \mathbb{R}^{n \times n}$  пусть  $\lambda_i(A)$  означает  $i$ -е максимальное собственное значение  $A$ . Таким образом,

$$\lambda_n(A) \leq \lambda_{n-1}(A) \leq \dots \leq \lambda_2(A) \leq \lambda_1(A).$$

Легко показать, что каждое собственное значение симметричной матрицы является стационарной точкой отображения  $x \rightarrow x^T A x / x^T x$ , где  $x \neq 0$ . Более того, собственные значения допускают следующую минимаксную характеристизацию:

**Теорема 8.1.2 (Теорема Куранта – Фишера о минимаксе).** Если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, то

$$\lambda_k(A) = \max_{\dim(S)=k} \min_{0 \neq y \in S} \frac{y^T a y}{y^T y}$$

для  $k = 1:n$ .

*Доказательство.* [Wilkinson, AEP], с. 100–101.  $\square$

Используя этот результат, можно установить несколько очень полезных следствий.

**Следствие 8.1.3.** Если  $A$  и  $A + E$  – симметричные матрицы размера  $n \times n$ , то

$$\lambda_k(A) + \lambda_n(E) \leq \lambda_k(A + E) \leq \lambda_k(A) + \lambda_1(E), \quad k = 1:n.$$

*Доказательство* [Wilkinson, AEP], с. 101–102.  $\square$

**Пример 8.1.2.** Если

$$A = \begin{bmatrix} 6.8 & 2.4 \\ 2.4 & 8.2 \end{bmatrix} \text{ и } E = \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix},$$

то  $\lambda(A) = \{5, 10\}$ ,  $\lambda(E) = \{1, 2\}$  и  $\lambda(A + E) = \{6.5917, 11.4083\}$ .

**Следствие 8.1.4. (Свойство чередования).** Если  $A$ , означает верхнюю  $r \times r$  подматрицу симметричной матрицы  $A$  размера  $n \times n$ , то для  $r = 1:n - 1$  справедливо

следующее свойство чередования:

$$\lambda_{r+1}(A_{r+1}) \leq \lambda_r(A_r) \leq \lambda_r(A_{r+1}) \leq \dots \leq \lambda_2(A_{r+1}) \leq \lambda_1(A_r)$$

*Доказательство.* [Wilkinson, AEP], с. 103–104.  $\square$

**Пример 8.1.3.** Если

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix},$$

то  $\lambda(A_1) = \{1\}$ ,  $\lambda(A_2) = \{0.3820, 2.6180\}$ ,  $\lambda(A_3) = \{0.1270, 1.0000, 7.873\}$  и  $\lambda(A_4) = \{0.0380, 0.4538, 7.2034, 26.3047\}$ .

### 8.1.3. Дополнительные результаты

Если матрицу  $A$  подвергнуть одноранговому возмущению, то ее собственные значения сдвинутся некоторым очень регулярным образом.

**Теорема 8.1.5.** Пусть  $B = A + \tau cc^T$ , где  $A \in \mathbb{R}^{n \times n}$  – симметричная матрица,  $c \in \mathbb{R}^n$  – вектор, имеющий единичную вторую норму и  $\tau \in \mathbb{R}$ . Если  $\tau \geq 0$ , то

$$\lambda_i(B) \in [\lambda_i(A), \lambda_{i-1}(A)], \quad i = 2:n;$$

а если  $\tau \leq 0$ , то

$$\lambda_i(B) \in [\lambda_{i+1}(A), \lambda_i(A)], \quad i = 1:n-1.$$

В обоих случаях существуют неотрицательные числа  $m_1, \dots, m_n$ , такие, что

$$\lambda_i(B) = \lambda_i(A) + m_i \tau, \quad i = 1:n,$$

$$c m_1 + \dots + m_n = 1.$$

*Доказательство.* [Wilkinson, AEP], с. 94–97. См. также теорему 8.6.2.  $\square$

Воздействие однорангового возмущения на собственные значения симметричной матрицы интересно в различных контекстах, таких, как квазиньютоновские методы для безусловной оптимизации. См. также § 8.6, где теорема 8.1.5 лежит в основе решения трехдиагональной проблемы собственных значений методом «разделяй и властвуй».

Наш следующий результат, касающийся возмущений, является скорее другим вариантом предыдущего. Он устанавливает границы для разности  $\lambda(A)$  и  $\lambda(A + E)$  в терминах  $\|E\|_F$ .

**Теорема 8.1.6 (Виландта–Хоффмана).** Если  $A$  и  $A + E$  – симметрические матрицы размера  $n \times n$ , то

$$\sum_{i=1}^n (\lambda_i(A + E) - \lambda_i(A))^2 \leq \|E\|_F^2.$$

*Доказательство.* Доказательство можно найти в [Wilkinson, AEP], с. 104–108.

**Пример 8.1.4.** Если

$$A = \begin{bmatrix} 6.8 & 2.4 \\ 2.4 & 8.2 \end{bmatrix} \text{ и } E = \begin{bmatrix} 2.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix},$$

то

$$4.5168 = \sum_{i=1}^2 (\lambda_i(A+E) - \lambda_i(A))^2 \leq \|E\|_F^2 = 5.$$

См. пример 8.1.2.

### 8.1.4. Чувствительность инвариантных подпространств

Инвариантные подпространства симметричной матрицы не обязательно хорошо обусловлены. Отделенность собственных значений является критическим фактором, как и в несимметричном случае. Однако приведенная оценка в общей теореме о возмущении для инвариантных подпространств (теорема 7.2.4) допускает некоторое упрощение в симметричном случае.

**Теорема 8.1.7.** Пусть  $A$  и  $A+E$  – симметричные  $n \times n$  матрицы и

$$Q = \begin{bmatrix} Q_1 & Q_2 \\ k & n-k \end{bmatrix}$$

– ортогональная матрица, такая, что  $\text{range}(Q_1)$  является инвариантным подпространством матрицы  $A$ . Разобьем матрицы  $Q^T A Q$  и  $Q^T E Q$  на блоки следующим образом:

$$\begin{aligned} Q^T A Q &= \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \\ k & n-k \end{bmatrix} \quad \begin{matrix} k \\ n-k \end{matrix}, \\ Q^T E Q &= \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \\ k & n-k \end{bmatrix} \quad \begin{matrix} k \\ n-k \end{matrix}. \end{aligned}$$

Если

$$\delta = \min_{\substack{\lambda \in \lambda(A_{11}) \\ \mu \in \lambda(A_{22})}} |\lambda - \mu| - \|E_{11}\|_2 - \|E_{22}\|_2 > 0$$

и  $\|E_{12}\|_2 \leq \delta/2$ , то существует матрица  $P \in \mathbb{R}^{(n-k) \times k}$ , удовлетворяющая неравенству

$$\|P\|_2 \leq \frac{2}{\delta} \|E_{21}\|_2,$$

такая, что столбцы матрицы  $Q_1 = (Q_1 + Q_2 P)(I + P^T P)^{-1/2}$  образуют ортогональный базис в подпространстве, являющемся инвариантным для матрицы  $A+E$ .

**Доказательство.** См. [Stewart, 1973в].  $\square$

Эта теорема в сущности утверждает, что чувствительность инвариантного подпространства симметричной матрицы зависит исключительно от отделенности связанных с ним собственных значений.

**Пример 8.1.5.** Если  $A = \text{diag}(0.999, 1.001, 2.0)$  и

$$E = \begin{bmatrix} 0.00 & 0.01 & 0.01 \\ 0.01 & 0.00 & 0.01 \\ 0.01 & 0.01 & 0.00 \end{bmatrix},$$

то  $\hat{Q}^T(A + E)\hat{Q} = \text{diag}(0.9899, 1.0098, 2.0002)$ , где

$$\hat{Q} = \begin{bmatrix} -0.7418 & 0.6706 & 0.0101 \\ 0.6708 & 0.7417 & 0.0101 \\ 0.0007 & -0.0143 & 0.9999 \end{bmatrix}$$

является ортогональной. Пусть  $\hat{q}_i = \hat{Q}e_i$ ,  $i = 1, 2, 3$ , т.е. вектор  $\hat{q}_i$  есть возмущенный собственный вектор  $q_i = e_i$  матрицы  $A$ . Вычисления показывают, что

$$\text{dist}\{\text{span}\{q_1\}, \text{span}\{\hat{q}_1\}\} = \text{dist}\{\text{span}\{q_2\}, \text{span}\{\hat{q}_2\}\} = 0.67.$$

Таким образом, векторы  $q_1$  и  $q_2$  не могут быть вычислены точно, поскольку они связаны с близкими собственными значениями. С другой стороны, так как  $\lambda_1$  и  $\lambda_2$  хорошо отделены от  $\lambda_3$ , то они определяют двумерное подпространство не столь чувствительное:  $\text{dist}\{\text{span}\{a_1, a_2\}, \text{span}\{\hat{q}_1, \hat{q}_2\}\} = 0.01$ .

Наконец мы приведем ряд результатов, касающихся аппроксимации инвариантных подпространств. Эти результаты будут полезны в следующих разделах и помогут интерпретации различных алгоритмов.

**Теорема 8.1.8.** Пусть матрицы  $A \in \mathbf{R}^{n \times n}$  и  $S \in \mathbf{R}^{k \times k}$  симметричные и

$$AQ_1 - Q_1S = E_1,$$

где матрица  $Q_1 \in \mathbf{R}^{n \times k}$  удовлетворяет равенству  $Q_1^T Q_1 = T_k$ . Тогда существуют собственные значения  $\mu_1, \dots, \mu_k \in \lambda(A)$ , такие, что

$$|\mu_i - \lambda_i(S)| \leq \sqrt{2} \|E_1\|_2, \quad i = 1 : k.$$

**Доказательство.** Пусть  $Q_2 \in \mathbf{R}^{n \times (n-k)}$  — любая матрица, такая, что матрица  $Q = [Q_1, Q_2]$  ортогональная. Тогда

$$Q^T A Q = \begin{bmatrix} S & 0 \\ 0 & Q_2^T A Q_2 \end{bmatrix} + \begin{bmatrix} Q_1^T E_1 & E_1^T Q_2 \\ Q_2^T E_1 & 0 \end{bmatrix} \equiv B + E$$

и, используя следствие 8.1.3, мы получаем оценку  $|\lambda_i(A) - \lambda_i(B)| \leq \|E\|_2$  для  $i = 1 : n$ . Поскольку  $\lambda(S) \subseteq \lambda(B)$ , существуют  $\mu_1, \dots, \mu_k \in \lambda(A)$ , такие, что

$$|\mu_i - \lambda_i(S)| \leq \|E\|_2$$

для  $i = 1 : k$ . Доказываемое неравенство следует из того, что для любых  $x \in \mathbf{R}^k$  и  $y \in \mathbf{R}^{n-k}$  мы имеем

$$\left\| E \begin{bmatrix} x \\ y \end{bmatrix} \right\|_2 \leq \|E_1 x\|_2 + \|E_1^T Q_2 y\|_2 \leq \|E_1\|_2 \|x\|_2 + \|E_1\|_2 \|y\|_2,$$

откуда легко заключить, что  $\|E\|_2 \leq \sqrt{2} \|E_1\|_2$ .  $\square$

**Пример 8.1.6.** Если

$$A = \begin{bmatrix} 6.8 & 2.4 \\ 2.4 & 8.2 \end{bmatrix}, \quad Q_1 = \begin{bmatrix} 0.7994 \\ 0.6007 \end{bmatrix}, \quad S = (5.1) \in \mathbf{R},$$

то

$$AQ_1 - Q_1 S = \begin{bmatrix} -0.0828 \\ -0.0562 \end{bmatrix} = E_1.$$

Теорема предсказывает, что матрица  $A$  имеет собственное значение на расстоянии не более  $\sqrt{2} \|E_1\|_2 \approx 0.1415$  от числа 5.1. Это соответствует действительности, так как  $\lambda(A) = \{5, 10\}$ .

Оценка собственного значения в теореме 8.1.8 зависит от погрешности аппроксимации инвариантного подпространства, т. е.  $\|AQ_1 - Q_1S\|_2$ . Для заданных матриц  $A$  и  $Q_1$  следующая теорема показывает, как выбрать матрицу  $S$  так, чтобы эта величина была минимальной, если  $\|\cdot\| = \|\cdot\|_F$ .

**Теорема 8.1.9.** *Если матрица  $A \in \mathbb{R}^{n \times n}$  – симметричная, а матрица  $Q_1 \in \mathbb{R}^{n \times k}$  удовлетворяет уравнению  $Q_1^T Q_1 = I_k$ , то*

$$\min_{S \in \mathbb{R}^{k \times k}} \|AQ_1 - Q_1S\|_F = \|(I - Q_1Q_1^T)AQ_1\|_F.$$

*Доказательство.* Пусть матрица  $Q_2 \in \mathbb{R}^{n \times (n-k)}$  такова, что матрица  $Q = [Q_1, Q_2]$  – ортогональная. Для любой матрицы  $S \in \mathbb{R}^{k \times k}$  имеем

$$\|AQ_1 - Q_1S\|_F^2 = \|Q^T A Q_1 - Q^T Q_1 S\|_F^2 = \|Q_1^T A Q_1 - S\|_F^2 + \|Q_2^T A Q_1\|_F^2.$$

Ясно, что минимум по  $S$  дает  $S = Q_1^T A Q_1$ .  $\square$

Этот результат дает нам возможность связывать  $k$ -мерное подпространство  $\text{range}(Q_1)$  с некоторым набором  $k$  «оптимальных» приближенных пар собственные значения – собственные векторы.

**Теорема 8.1.10.** *Пусть матрица  $A \in \mathbb{R}^{n \times n}$  – симметричная, а матрица  $Q_1 \in \mathbb{R}^{n \times k}$  удовлетворяет уравнению  $Q_1^T Q_1 = I_k$ . Если*

$$Z^T(Q_1^T A Q_1)Z = \text{diag}(\theta_1, \dots, \theta_k) = D$$

– разложение Шура матрицы  $Q_1^T A Q_1$  и  $Q_1 Z = [y_1, \dots, y_k]$ , то  $\|Ay_i - \theta_i y_i\|_2 = \|(I - Q_1 Q_1^T)A Q_1 Z e_i\|_2 = \|(I - Q_1 Q_1^T)A Q_1\|_2$  для  $i = 1 : k$ . Величины  $\theta_i$  называются числами Ритца, векторы  $y_i$  называются векторами Ритца, а пары  $(\theta_i, y_i)$  – парами Ритца.

*Доказательство.*

$$Ay_i - \theta_i y_i = A Q_1 Z e_i = Q_1 Z D e_i = A Q_1 - Q_1 (Q_1^T A Q_1) Z e_i.$$

Беря норму, получаем доказываемое равенство.  $\square$

Полезность теоремы 8.1.8 увеличится, если мы ослабим предположение об ортогональности столбцов матрицы  $Q_1$ . Как и следует ожидать, оценки становятся хуже с потерей ортогональности.

**Теорема 8.1.11.** *Пусть  $A \in \mathbb{R}$  – симметричная матрица и*

$$AX_1 - X_1 S = F_1,$$

где  $X_1 \in \mathbb{R}^{n \times k}$  и  $S = X_1^T A X_1$ . Если  $\sigma_k(X_1) > 0$  и

$$\|X_1^T X_1 - I_k\|_2 = \tau,$$

то существуют собственные значения  $\mu_1, \dots, \mu_k \in \lambda(A)$ , такие, что

$$|\mu_i - \lambda_i(S)| \leq \sqrt{2} \left( \frac{\|F_1\|_2}{\sigma_k(X_1)} + \tau \|A\|_2 \right)$$

для  $k = 1 : n$ .

*Доказательство.* Пусть  $X_1 = U_1 \Sigma V^T$  есть SVD матрицы  $X_1$  с матрицами  $U_1 \in \mathbb{R}^{n \times k}$ ,  $V \in \mathbb{R}^{k \times k}$  и  $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_k) \in \mathbb{R}^{k \times k}$ . Подставляя его в уравнение  $AX_1 - X_1 S = F_1$ ,

мы получим равенство

$$AU_1\Sigma V^T - U_1\Sigma V^T(U_1\Sigma V^T)^TA(U_1\Sigma V^T) = F;$$

поэтому

$$AU_1 - U_1\Sigma^2(U_1^TAU_1) = FV\Sigma^{-1}.$$

Отсюда следует, что

$$AU_1 - U_1(U_1^TAU_1) = E_1, \quad (8.1.1)$$

где  $E_1 = FV\Sigma^{-1} + U_1(\Sigma^2 - I_k)U_1^TAU_1$ . Замечая, что

$$\|\Sigma^2 - I_k\|_2 = \|X_1^TX_1 - I_k\|_2 = \tau,$$

мы получаем оценку

$$\|E_1\|_2 \leq \|FV\Sigma^{-1}\|_2 + \|U_1(\Sigma^2 - I_k)U_1^TAU_1\|_2 \leq \frac{\|F\|_2}{\sigma_k(X_1)} + \tau \|A\|_2.$$

Применяя теорему 8.1.8 с матрицами  $Q = U_1$  и  $S = U_1^TAU_1$ , получаем доказываемое неравенство.  $\square$

### 8.1.5. Закон инерции

Мы завершим этот раздел установлением закона инерции Сильвестра. *Инерция* симметричной матрицы  $A$  – это тройка неотрицательных целых чисел  $(m, z, p)$ , где  $m$ ,  $z$  и  $p$  – число отрицательных, нулевых и положительных элементов множества  $\lambda(A)$ .

**Теорема 8.1.12 (Закон инерции Сильвестра).** *Если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, а матрица  $X \in \mathbb{R}^{n \times n}$  невырожденная, то матрицы  $A$  и  $X^TAX$  имеют одну и ту же инерцию.*

*Доказательство.* Предположим, что для некоторого  $r$   $\lambda_r(A) > 0$ , и определим подпространство  $S_0 \subseteq \mathbb{R}^n$  следующим образом:

$$S_0 = \text{span} \{X^{-1}q_1, \dots, X^{-1}q_r\}, \quad q_i \neq 0,$$

где  $Aq_i = \lambda_i(A)q_i$  и  $i = 1:r$ . Из минимаксной характеристики собственного значения  $\lambda_r(X^TAX)$  имеем:

$$\begin{aligned} \lambda_r(X^TAX) &= \max_{\dim(S)=r} \min_{y \in S} \frac{y^T(X^TAX)y}{y^Ty} \geq \\ &\geq \min_{y \in S_0} \frac{y^T(X^TAX)y}{y^Ty}. \end{aligned}$$

Далее, для любого  $y \in \mathbb{R}^n$  имеем

$$\frac{y^T(X^TAX)y}{y^Ty} \geq \sigma_n(X)^2,$$

в то время как для  $y \in S_0$  ясно, что

$$\frac{y^T(X^TAX)y}{y^Ty} \geq \lambda_r(A).$$

Следовательно,

$$\lambda_r(X^TAX) \geq \min_{y \in S_0} \left\{ \frac{y^T(X^TAX)y}{y^T(X^TX)y} \frac{y^T(X^TX)y}{y^Ty} \right\} \geq \lambda_r(A) \sigma_n(X)^2.$$

Если матрицы  $A$  и  $X^TAX$  поменять ролями, то аналогичные рассуждения покажут, что

$$\lambda_r(A) \geq \lambda_r(X^TAX) \sigma_n(X^{-1})^2 = \frac{\lambda_r(X^TAX)}{\sigma_1(X)^2}.$$

Таким образом, собственные числа  $\lambda_r(A)$  и  $\lambda_r(X^TAX)$  имеют один и тот же знак и, следовательно, мы показали, что матрицы  $A$  и  $X^TAX$  имеют одно и то же число положительных собственных значений. Если применить этот результат к матрице  $-A$ , то можно сделать вывод, что матрицы  $A$  и  $X^TAX$  имеют одно и то же число отрицательных собственных значений. Очевидно, что число нулевых собственных значений, которое имеет каждая из матриц, тоже будет одним и тем же.  $\square$

**Пример 8.1.7.** Если  $A = \text{diag}(3, 2, -1)$  и

$$X = \begin{bmatrix} 1 & 4 & 5 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix},$$

то

$$X^TAX = \begin{bmatrix} 4 & 12 & 15 \\ 12 & 50 & 64 \\ 15 & 64 & 82 \end{bmatrix}$$

и  $\lambda(X^TAX) = \{134.769, 0.3555, -0.1252\}$ .

### Задачи

**8.1.1.** Показать, что собственные значения симметричной матрицы размера  $2 \times 2$  должны быть вещественные.

**8.1.2.** Найти разложение Шура матрицы  $A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$ .

**8.1.3.** Показать, что собственные значения эрмитовой матрицы ( $A^H = A$ ) вещественные. Для каждой теоремы и следствия из этого раздела сформулировать и доказать соответствующий результат для эрмитовых матриц. Какие результаты имеют аналоги, если матрица  $A$  кососимметричная? (Подсказка: Если  $A^T = -A$ , то матрица  $iA$  эрмитова.)

### Замечания и литература к § 8.1

Теория возмущения симметричной проблемы собственных значений рассматривается в [Wilkinson, AEP], гл. 2 и [Parlett, SEP], гл. 10 и 11. Некоторые типичные работы в этой хорошо освещенной области включает следующий перечень:

Deift P., Nande T. and Tome C. (1983). "Ordinary Differential Equations and the Symmetric Eigenvalue Problem", SIAM J. Numer. Anal. 20, 1–22.

Kahan W. (1975). "Spectra of Nearly Hermitian Matrices", Proc. Amer. Math. Soc. 48, 11–17.

Kahan W. (1967). "Inclusion Theorems for Clusters of Eigenvalues of Hermitian Matrices", Computer Science Report, University of Toronto.

Paige C. C. (1974b). "Eigenvalues of Perturbed Hermitian Matrices", Lin. Alg. and Its Applic. 8, 1–10.

Ruhe A. (1975). "On the Closeness of Eigenvalues and Singular Values for Almost Normal Matrices", Lin. Alg. and Its Applic. 11, 87–94.

- Schonhage A. (1979). "Arbitrary Perturbations of Hermitian Matrices", Lin. Alg. and Its Applic. 24, 143–49.
- Scott D. S. (1985). "On the Accuracy of the Gershgorin Circle Theorem for Bounding the Spread of a Real Symmetric Matrix", Lin. Alg. and Its Applic. 65, 147–155.
- Stewart G. W. (1973b). "Error and Perturbation Bounds for Subspaces Associated with Certain Eigenvalue Problems", SIAM Review, 15, 727–64.

## 8.2. Симметрический QR-алгоритм

Посмотрим теперь, как практический QR-алгоритм, полученный в гл. 7, может быть специализирован, если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная. Три замечания можно сделать сразу:

- Если матрица  $U_0^T A U_0 = H$  верхняя хессенбергова, то  $H = T$  должна быть трехдиагональной.
- Симметричность и трехдиагональная ленточная структуры сохраняются после выполнения QR-шага с одинарным сдвигом.
- Нет необходимости рассматривать комплексные сдвиги, так как  $\lambda(A) \subseteq \mathbb{R}$ .

Эти упрощения вместе с приятными математическими свойствами симметричной проблемы собственных значений делают алгоритмы этой главы очень привлекательными.

### 8.2.1. Преобразование к трехдиагональному виду

Начнем с получения преобразования Хаусхолдера к трехдиагональному виду. Предположим, что матрицы Хаусхолдера  $P_1, \dots, P_{k-1}$  определены так, что подматрица  $B_{11}$  матрицы

$$A_{k-1} = (P_1 \dots P_{k-1})^T A (P_1 \dots P_{k-1}) = \begin{bmatrix} B_{11} & B_{12} & 0 & & & \\ B_{21} & B_{22} & B_{23} & & & \\ 0 & B_{32} & B_{33} & & & \\ & & & & & \\ & & & & & \\ k-1 & 1 & n-k & & & \end{bmatrix} \begin{matrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ 1 & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ n-k & & & & & \end{matrix}$$

трехдиагональная. Если  $\bar{P}_k$  – матрица Хаусхолдера порядка  $n - k$ , такая, что вектор  $\bar{P}_k B_{32}$  коллинеарен  $e_1^{(n-k)}$  и, если  $P_k = \text{diag}(I_k, \bar{P}_k)$ , то верхняя главная подматрица размера  $k \times k$  матрицы

$$A_k = P_k A_{k-1} P_k = \begin{bmatrix} B_{11} & B_{12} & 0 & & & \\ B_{21} & B_{22} & B_{23} \bar{P}_k & & & \\ 0 & \bar{P}_k B_{32} & \bar{P}_k B_{33} \bar{P}_k & & & \\ & & & & & \\ & & & & & \\ k-1 & 1 & n-k & & & \end{matrix} \begin{matrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ 1 & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ n-k & & & & & \end{matrix}$$

трехдиагональная<sup>1)</sup>. Ясно, что если  $U_0 = P_1 \dots P_{n-2}$ , то матрица  $U_0^T A U_0 = T$  трехдиагональная.

При вычислении матрицы  $A_k$  важно использовать симметричность в процессе формирования матрицы  $\bar{P}_k B_{33} \bar{P}_k$ . Для определенности предположим, что матрица

<sup>1)</sup> Перед  $k$ -м шагом трехдиагональной является верхняя главная подматрица порядка  $k + 1$ , а описанная процедура  $k$ -го шага делает трехдиагональной верхнюю главную подматрицу порядка  $k + 2$ . – Прим. перев.

$\bar{P}_k$  имеет вид

$$\bar{P}_k = I - \frac{2}{v^T v} vv^T, \quad 0 \neq v \in \mathbf{R}^{n-k}.$$

Заметим, что если

$$P = \frac{2}{v^T v} B_{33} v \text{ и } w = p - \frac{p^T v}{v^T v} v,$$

то

$$\bar{P}_k B_{33} \bar{P}_k = B_{33} - vw^T - wv^T.$$

Необходимо вычислить только верхнюю треугольную часть этой матрицы, поэтому переход от  $A_{k-1}$  к  $A_k$  можно выполнить всего за  $4(n-k)^2$  флопов.

В целом мы имеем следующую процедуру трехдиагонализации:

**Алгоритм 8.2.1 (Трехдиагонализация Хаусхолдера).** Для заданной симметричной матрицы  $A$  размера  $n \times n$  следующий алгоритм записывает в  $A$  матрицу  $T = U_0^T A U_0$ , где матрица  $T$  – трехдиагональная, а  $U_0 = H_1 \dots H_{n-2}$  – произведение преобразований Хаусхолдера.

```

for  $k = 1 : n - 2$ 
     $v = \text{house}(A(k+1:n, k))$ 
     $p = 2A(k+1:n, k+1:n)v/v^T v$ 
     $w = p - (p^T v)v/v^T v$ 
     $A(k+1:n, k+1:n) = A(k+1:n, k+1:n) - vw^T - wv^T$ 
end
```

Этот алгоритм требует  $4n^3/3$  флопов, если использовать симметричность при выполнении двухрангового обновления. Матрицу  $U_0$  можно хранить в факторизованном виде в поддиагональной части матрицы  $A$ . Если матрица  $U_0$  требуется в явном виде, то ее можно сформировать, выполнив дополнительно  $4n^3/3$  флопов.

**Пример 8.2.1.** Если

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 3 & 2 & 8 \\ 4 & 8 & 3 \end{bmatrix} \text{ и } U_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.6 & 0.8 \\ 0 & 0.8 & -0.6 \end{bmatrix},$$

то  $U_0^T U_0 = I$  и

$$U_0^T A U_0 = \begin{bmatrix} 1 & 5 & 0 \\ 5 & 10.32 & 1.76 \\ 0 & 1.76 & -5.32 \end{bmatrix}.$$

Пусть  $\hat{T}$  означает вариант матрицы  $T$ , вычисленный по алгоритму 8.2.1. Можно показать, что  $\hat{T} = Q^T (A + E) Q$ , где  $Q$  – точно ортогональная матрица, а  $E$  – симметричная матрица, удовлетворяющая неравенству  $\|E\|_F \leq c \|\mathbf{u}\| \|A\|_F$ , где  $c$  – некоторая небольшая константа. См. [Wilkinson, AEP], с. 297.

### 8.2.2. QR-итерации с явным одинарным сдвигом

Рассмотрим теперь QR-итерации с одинарным сдвигом для симметрических матриц:

```

 $T = U_0^T A U_0$  (трехдиагональная)
for  $k = 0, 1, \dots$ 
    Определить вещественный сдвиг  $\mu$ . (8.2.1)
     $T - \mu I = UR$  (QR-разложение)
     $T = RU + \mu I$ 
end

```

Эти итерации сохраняют хессенбергову структуру (ср. с алгоритмом 7.4.1) и симметричность, следовательно, они сохраняют трехдиагональную форму. Более того, QR-разложение трехдиагональной матрицы требует лишь  $O(n)$  флопов, так как матрица  $R$  имеет только две наддиагонали. Поэтому необходимо выполнять на порядок меньше вычислений, чем в несимметричном случае. Однако если накапливать ортогональные матрицы, потребуется  $O(n^2)$  флопов на итерацию.

Симметричность можно также использовать при вычислении сдвига  $\mu$ . Обозначим через  $T$  следующую матрицу

$$T = \begin{bmatrix} a_1 & b_1 & & \cdots & 0 \\ b_1 & a_2 & \cdots & & \vdots \\ \ddots & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & & b_{n-1} & a_n \end{bmatrix}.$$

Мы можем, очевидно, положить  $\mu = a_n$ . Однако эффективнее сдвигать на собственное значение подматрицы

$$T(n-1:n, n-1:n) = \begin{bmatrix} a_{n-1} & b_{n-1} \\ b_{n-1} & a_n \end{bmatrix}$$

охватывающей элемент  $a_n$ . Такой выбор сдвига известен как сдвиг Уилкинсона и его вычисляют по формуле

$$\mu = a_n + d - \text{sign}(d)\sqrt{d^2 + b_{n-1}^2}, \quad (8.2.2)$$

где  $d = (a_{n-1} - a_n)/2$ . Уилкинсон [Wilkinson, 1968] показал, что процесс (8.2.1) сходится кубически при любой стратегии сдвигов, но дал эвристическое объяснение, почему сдвиг (8.2.2) предпочтительнее.

### 8.2.3. Вариант с неявным сдвигом

Как и в случае несимметрических QR-итераций, в алгоритме (8.2.1) есть возможность делать неявный сдвиг. То есть мы можем выполнять переход от  $T$  к  $T_+ = RU + \mu I = U^T TU$  без явного формирования матрицы  $T - \mu I$ . Пусть вычислены  $c = \cos(\theta)$  и  $s = \sin(\theta)$ , такие, что

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_1 - \mu \\ b_1 \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}.$$

Если положить  $G_1 = G(1, 2, \theta)$ , то  $G_1 e_1 = U e_1$  и

$$T \leftarrow G_1^T T G_1 = \begin{bmatrix} \times & \times & + & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ + & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

На этом этапе применима теорема о неявном  $Q$  при условии, что мы можем вычислять вращения  $G_2, \dots, G_{n-1}$ , обладающие тем свойством, что если  $Z = G_1 G_2 \dots G_{n-1}$ , то  $Ze_1 = G_1 e_1 = Ue^1$  и матрица  $Z^T TZ$  – трехдиагональная.

Заметим, что первый столбец у матриц  $Z$  и  $U$  один и тот же при условии, что мы берем матрицы  $G_i$  вида  $G_i = G(i, i+1, \theta_i)$ ,  $i = 2:n-1$ . Но матрицу  $G_i$  такого вида можно использовать для выталкивания ненулевого элемента «+» из матрицы  $G_1^T TG_1$ :

$$\xrightarrow{G_2} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & + & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 \\ 0 & + & \times & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G_3} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & + & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & + & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}$$
  

$$\xrightarrow{G_4} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & + \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & + & \times & \times \end{bmatrix} \xrightarrow{G_5} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}$$

Из теоремы о неявном  $Q$  следует, что трехдиагональная матрица  $Z^T TZ$ , полученная таким выталкиванием нулевых элементов, в основном та же, что и трехдиагональная матрица  $T$ , полученная явным способом. (Мы можем предполагать, что все трехдиагональные матрицы по определению невырожденные, так как в противном случае проблема расщепляется.)

Заметим, что на любом этапе выталкивания ненулевых элементов с внешней стороны трехдиагональной ленты расположен лишь один ненулевой элемент. Следующее равенство показывает, как этот ненулевой элемент опускается в нижний угол матрицы в процессе обновления  $T \leftarrow G_k^T TG_k$ :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} a_k & b_k & z_k & 0 \\ b_k & a_p & b_p & 0 \\ z_k & b_p & a_q & b_q \\ 0 & 0 & b_q & a_r \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} a_k & b_k & 0 & 0 \\ b_k & a_p & b_p & z_p \\ 0 & b_p & a_q & b_q \\ 0 & z_p & b_q & a_r \end{bmatrix}.$$

Здесь  $(p, q, r) = (k + 1, k + 2, k + 3)$ . Это обновление может быть выполнено не более чем за 26 флоков, если определены числа  $c$  и  $s$  из уравнения  $b_k s + z_k c = 0$ . В целом получаем

**Алгоритм 8.2.2 (Неявный симметрический QR-шаг со сдвигом Уилкинсона).** Для заданной неприводимой симметрической трехдиагональной матрицы  $T \in \mathbb{R}^{n \times n}$ , следующий алгоритм записывает в матрицу  $T$  матрицу  $Z^T TZ$ , где  $Z = G_1 \dots G_{n-1}$  – произведение вращений Гивенса, такое, что матрица  $Z^T(T - \mu I)$  – верхняя треугольная, а  $\mu$  – собственное значение нижней главной подматрицы размера  $2 \times 2$ , охватывающей элемент  $t_{nn}$ .

```

 $d = (t_{n-1,n-1} - t_{nn})/2$ 
 $\mu = t_{nn} - t_{n,n-1}^2/(d + \text{sign}(d)\sqrt{d^2 + t_{n,n-1}})$ 
 $x = t_{11} - \mu$ 
 $z = t_{21}$ 
for  $k = 1:n - 1$ 
     $[c, s] = \text{givens}(x, z)$ 
     $T = G_k^T T G_k, \quad G_k = G(k, k + 1, \theta)$ 
    if  $k < n - 1$ 
         $x = t_{k+1,k}$ 
         $z = t_{k+2,k}$ 
    end
end

```

Этот алгоритм требует примерно  $30n$  флоков и  $n$  извлечений квадратного корня. Если в заданную ортогональную матрицу  $Q$  записывать  $QG_1 \dots G_{n-1}$ , то потребуется дополнительно  $6n^2$  флоков. Конечно, при любой практической реализации трехдиагональную матрицу  $T$  желательно хранить в паре векторов длины  $n$ , а не в массиве размера  $n \times n$ .

**Пример 8.2.2.** Если алгоритм 8.2.2 применить к матрице

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 3 & 0.01 \\ 0 & 0 & 0.01 & 4 \end{bmatrix},$$

то новая трехдиагональная матрица  $T$  будет иметь вид

$$T = \begin{bmatrix} 0.5000 & 0.5916 & 0 & 0 \\ 0.5916 & 1.785 & 0.1808 & 0 \\ 0 & 0.1808 & 3.7140 & 0.0000044 \\ 0 & 0 & 0.0000044 & 4.002497 \end{bmatrix}.$$

Алгоритм 8.2.2 образует основу симметрического QR-алгоритма – стандартного средства вычисления разложения Шура плотной симметрической матрицы.

**Алгоритм 8.2.3 (Симметрический QR-алгоритм).** Для заданной симметрической матрицы  $A$  размера  $n \times n$  и числа  $\epsilon$ , немного превосходящего машинную точность, следующий алгоритм записывает в матрицу  $A$  матрицу  $Q^T A Q = D + E$ , где  $Q$  – ортогональная матрица,  $D$  – диагональная, а матрица  $E$  удовлетворяет приближенному равенству  $\|E\|_2 \approx \|A\|_2$ .

Используя алгоритм 8.2.1, выполнить трехдиагонализацию

$$T = (P_1 \dots P_{n-2})^T A (P_1 \dots P_{n-2})$$

**until**  $q = n$

Для  $i = 1:n - 1$  положить  $a_{i+1,i}$  и  $a_{i,i+1}$  равными нулю, если

$$|a_{i+1,i}| = |a_{i,i+1}| \leq \varepsilon (|a_{ii}| + |a_{i+1,i+1}|)$$

Найти наибольшее  $q$  и наименьшее  $p$ , такие, что если

$$T = \begin{bmatrix} T_{11} & 0 & 0 \\ 0 & T_{22} & 0 \\ 0 & 0 & T_{33} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$

то  $T_{33}$  диагональная, а  $T_{22}$  неприводимая.

**if**  $q < n$

Применить алгоритм 8.2.2 к  $T_{22}$ :

$$T = \text{diag}(I_p, \bar{Z}, I_q)^T T \text{diag}(I_p, \bar{Z}, I_q)$$

**end**

**end**

Этот алгоритм требует примерно  $4n^3/3$  флопов, если не накапливается матрица  $Q$ , и примерно  $9n^3$  флопов, если  $Q$  накапливается.

**Пример 8.2.3.** Пусть алгоритм 8.2.3 применяется к трехдиагональной матрице

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 3 & 4 & 0 \\ 0 & 4 & 5 & 6 \\ 0 & 0 & 6 & 7 \end{bmatrix}.$$

Поддиагональные элементы меняются в процессе работы алгоритма 8.2.3 следующим образом:

Итерация	$a_{21}$	$a_{32}$	$a_{43}$
1	1.6817	3.2344	0.8649
2	1.6142	2.5755	0.0006
3	1.6245	1.6965	$10^{-13}$
4	1.6245	1.6965	сошелся
5	1.5117	0.0150	
6	1.1195	$10^{-9}$	
7	0.7071		
8			

По завершении вычислений мы найдем  $\lambda(A) = \{-2.4848, 0.7046, 4.9366, 12.831\}$ .

Как и в несимметричном QR-алгоритме, вычисленные собственные значения  $\hat{\lambda}_i$ , полученные при помощи алгоритма 8.2.3, являются точными собственными значениями некоторой матрицы, близкой к  $A$ , т. е.  $Q_0^T (A + E) Q_0 = \text{diag}(\lambda_i)$ , где  $Q_0^T Q_0 = I$  и  $\|E\|_2 \approx \|A\|_2$ . Однако в отличие от несимметричного случая мы можем утверждать (благодаря свойству 8.1.3), что абсолютная погрешность каждого  $\hat{\lambda}_i$  мала в том смысле, что  $|\hat{\lambda}_i - \lambda_i| \approx \|E\|_2$ . Если  $\hat{Q} = [\hat{q}_1, \dots, \hat{q}_n]$  – вычисленная матрица ортогональных собственных векторов, то точность  $\hat{q}_i$  зависит от удаленности  $\lambda_i$  от остального спектра. См. теорему 8.1.7.

Если требуются все собственные значения и несколько собственных векторов, то накапливать матрицу  $Q$  в алгоритме 8.2.3 нецелесообразно. Вместо этого необходимо

димые собственные векторы можно искать обратными итерациями с матрицей  $T$ . Если требуется только несколько собственных значений и собственных векторов, то лучше использовать некоторые специальные методы, описанные в § 8.4.

### Задачи

**8.2.1.** Пусть матрица  $A = \begin{bmatrix} w & x \\ x & z \end{bmatrix}$  вещественная, и пусть мы выполнили следующий QR-шаг со сдвигом:  $A - zI = UR$ ,  $\bar{A} = RU + zI$ . Показать, что если  $\bar{A} = \begin{bmatrix} \bar{w} & \bar{x} \\ \bar{x} & \bar{z} \end{bmatrix}$ , то

$$\begin{aligned}\bar{w} &= w + x^2(w - z)/[(w - z)^2 + x^2], \\ \bar{z} &= z - x^2(w - z)/[(w - z)^2 + x^2], \\ \bar{x} &= -x^3/[(w - z)^2 + x^2].\end{aligned}$$

**8.2.2.** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  — симметричная и положительно определенная, и рассмотрим следующие итерации:

```

 $A_0 = A$ 
for  $k = 1, 2, \dots$ 
   $A_{k-1} = G_k G_k^T$  (Холецкий)
   $A_k = G_k^T G_k$ 
end

```

(a) Показать, что эти итерации сходятся. (b) Показать, что если матрица  $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$  с  $a \geq c$  имеет собственные значения  $\lambda_1 \geq \lambda_2 \geq 0$ , то матрицы  $A_k$  сходятся к  $\text{diag}(\lambda_1, \lambda_2)$ .

**8.2.3.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  кососимметрическая ( $A^T = -A$ ). Показать, как конструировать матрицы Хаусхольдера  $P_1, \dots, P_{n-2}$ , так, чтобы матрица  $(P_1 \dots P_{n-2})^T A (P_1 \dots P_{n-2})$  была трехдиагональной. Сколько флопов требует ваш алгоритм?

**8.2.4.** Пусть матрица  $A \in \mathbb{C}^{n \times n}$  эрмитова. Показать, как конструировать унитарную матрицу  $Q$ , так, чтобы матрица  $Q^H A Q = T$  была вещественной, симметрической и трехдиагональной.

**8.2.5.** Показать, что если матрица  $A = B + iC$  эрмитова, то матрица  $M = \begin{bmatrix} B & -C \\ C & B \end{bmatrix}$  — симметричная. Сравнить собственные значения и собственные векторы матриц  $A$  и  $M$ .

**8.2.6.** Переписать алгоритм 8.2.2 для случая, когда матрица  $A$  хранится в двух векторах длины  $n$ . Обосновать приведенное число флопов.

### Замечания и литература к § 8.2

Трехдиагонализация симметрической матрицы обсуждается в работах

Gibbs N. E. and Poole W. G., Jr. (1974). "Tridiagonalization by Permutations", Comm. ACM 17, 20–24.

Martin R. S. and Wilkinson J. H. (1968a). "Householder's Tridiagonalization of a Symmetric Matrix", Numer. Math. 11, 181–95. See also HACLA, pp. 212–26.

Schwartz H. R. (1968). "Tridiagonalization of a Symmetric Band Matrix", Numer. Math. 12, 231–41. See also HACLA, pp. 273–83.

Первые две из них содержат алгольные программы. Алгольные процедуры для явного и неявного трехдиагонального QR-алгоритма даны в

Bowdler H., Martin R. S., Reinsch C., and Wilkinson J. H. (1968). "The QR and QL Algorithms for Symmetric Matrices", Numer. Math. 11, 293–306. See also HACLA, pp. 227–40.

Dubrulle A., Martin R. S., and Wilkinson J. H. (1968). "The Implicit QL-Algorithm", Numer. Math. 12, 377–83. See also HACLA, pp. 241–48.

QL-алгоритм идентичен QR-алгоритму за исключением того, что на каждом шаге матрица  $T - \lambda I$  раскладывается в произведение ортогональной матрицы и верхней треугольной матрицы. Другие статьи, касающиеся этих методов, включают следующий список:

Stewart G. W. (1970). "Incorporating Original Shifts into the QR Algorithm for Symmetric Tridiagonal Matrices", Comm. ACM 13, 365–67.

Dubrulle A. (1970). "A Short Note on the Implicit QL Algorithm for Symmetric Tridiagonal Matrices", Numer. Math. 15, 450.

Распространение этих алгоритмов на эрмитовы и косоэрмитовы матрицы обсуждается в

Mueller D. (1966). "Householder's Method for Complex Matrices and Hermitian Matrices" Numer. Math. 8, 72–92.

Ward R. C. and Gray L. J. (1978). "Eigensystem Computation for Skew-Symmetric and A Class of Symmetric Matrices", ACM Trans. Math. Soft. 4, 278–85.

Свойства сходимости алгоритма 8.2.3 детально рассматриваются в Lawson, Hanson (SLS), app. B, так же, как и в работах

Dekker T. J. and Traub J. F. (1971). "The Shifted QR Algorithm for Hermitian Matrices", Lin. Alg. and Its Applic. 4, 137–54.

Hoffman W. and Parlett B. N. (1978). "A New Proof of Global Convergence for the Tridiagonal QL Algorithm", SIAM J. Num. Anal. 15, 929–37.

Wilkinson J. H. (1968b). "Global Convergence of Tridiagonal QR Algorithm With Origin Shifts", Lin. Alg. and Its Applic. 1, 409–20.

Анализ этого метода в случае, когда его применяют для нормальных матриц, смотрите в

Huang C. P. (1981). "On the Convergence of the QR Algorithm with Origin Shifts for Normal Matrices", OMA J. Num. Anal. 1, 127–33.

В число интересных статей, касающихся сдвигов в трехдиагональном QR-алгоритме, входят

Bauer F. L. and Reinsch C. (1968). "Rational QR Transformations with Newton Shift for Symmetric Tridiagonal Matrices", Numer. Math. 11, 264–72. See also HACLA, pp. 257–65.

Stewart G. W. (1970). "Incorporating Origin Shifts into the QR Algorithm for Symmetric Tridiagonal Matrices", Comm. Assoc. Mach. 13, 365–67.

Симметричность не единственный пример структуры, которая может быть учтена в QR-алгоритме. Другие специализированные варианты QR-алгоритма включают

Byers R. (1983). "Hamiltonian and Symplectic Algorithms for the Algebraic Riccati Equation", PhD Thesis, Center for Applied Mathematics, Cornell University.

Byers R. (1986). "A Hamiltonian QR Algorithm", SIAM J. Sci. and Stat. Comp. 7, 212–229.

Gragg W. B. (1986). "The QR Algorithm for Unitary Hessenberg Matrices", J. Comp. Appl. Math. 16, 1–8.

Van Loan C. (1984). "A Symplectic Method for Approximating All the Eigenvalues of a Hamiltonian Matrix", Lin. Alg. and Its Applic. 61, 233–252.

Возможность параллельного вычисления по алгоритмам этого раздела обсуждается в

Lo S., Philippe B., and Sameh A. (1987). "A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem", SIAM J. Sci. and Stat. Comp. 8, s155–s165.

Chang H. Y. and Salama M. (1988). "A Parallel Householder Tridiagonalization Stratgem Using Scattered Square Decomposition", Parallel Computing 6, 297–312.

## 8.3. Вычисление SVD

Существуют важные связи между сингулярным разложением матрицы  $A$  и разложениями Шура симметричных матриц  $A^T A$ ,  $AA^T$  и  $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$ . В самом деле, если

$$U^T A V = \text{diag}(\sigma_1, \dots, \sigma_n)$$

есть SVD матрицы  $A \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ), то

$$V^T (A^T A) V = \text{diag}(\sigma_1^2, \dots, \sigma_n^2) \in \mathbb{R}^{n \times n} \quad (8.3.1)$$

и

$$U^T(AA^T)U = \text{diag}(\sigma_1^2, \dots, \sigma_n^2, \underbrace{0, \dots, 0}_{m-n}) \in \mathbb{R}^{m \times m}. \quad (8.3.2)$$

Кроме того, если

$$U = \begin{bmatrix} U_1 & U_2 \\ n & m-n \end{bmatrix}$$

и мы определим ортогональную матрицу  $Q \in \mathbb{R}^{(m+n) \times (m+n)}$  как

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} V & V & 0 \\ U_1 & -U_1 & \sqrt{2}U_2 \end{bmatrix},$$

то

$$Q^T \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} Q = \text{diag}(\sigma_1, \dots, \sigma_n, -\sigma_1, \dots, -\sigma_n, \underbrace{0, \dots, 0}_{m-n}).$$

Эти связи с симметрической проблемой собственных значений позволяют нам приспособить математические и алгоритмические разработки из двух предыдущих разделов к проблеме сингулярных значений.

### 8.3.1. Теория возмущения и свойства

Сначала мы установим некоторые результаты, касающиеся возмущения SVD на основе теорем из § 8.1. Напомним, что  $\sigma_i(A)$  означает  $i$ -е наибольшее сингулярное число матрицы  $A$ .

**Теорема 8.3.1.** Если  $A \in \mathbb{R}^{m \times n}$ , то для  $k = 1 : \min\{m, n\}$

$$\sigma_k(A) = \max_{\substack{\dim(S)=k \\ \dim(T)=k}} \min_{\substack{x \in S \\ y \in T}} \frac{y^T Ax}{\|x\|_2 \|y\|_2} = \max_{\substack{\dim(S)=k \\ \dim(T)=k}} \min_{x \in S} \frac{\|Ax\|_2}{\|x\|_2}.$$

Заметим, что в этом выражении  $S \subseteq \mathbb{R}^n$  и  $T \subseteq \mathbb{R}^m$  – подпространства.

**Доказательство.** Крайняя правая характеристика следует из теоремы 8.1.2, примененной к матрице  $A^T A$ . Остальную часть доказательства мы оставляем в качестве упражнения.  $\square$

**Следствие 8.3.2.** Если  $A$  и  $A + E$  – матрица из  $\mathbb{R}^{m \times n}$  с  $m \geq n$ , то при  $k = 1:n$  справедливы неравенства

$$|\sigma_k(A + E) - \sigma_k(A)| \leq \sigma_1(E) = \|E\|_2.$$

**Доказательство.** Применить следствие 8.1.3 к матрицам

$$\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \text{ и } \begin{bmatrix} 0 & (A + E)^T \\ A + E & 0 \end{bmatrix}. \quad \square$$

**Пример 8.3.1.** Если

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \text{и} \quad A + E = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6.01 \end{bmatrix},$$

то  $\sigma(A) = \{9.5080, 0.7729\}$  и  $\sigma(A + E) = \{9.5145, 0.7706\}$ . Ясно, что при  $i = 1:2$  справедливо неравенство  $|\sigma_i(A + E) - \sigma_i(A)| \leq \|E\|_2 = 0.01$ .

**Следствие 8.3.3.** Пусть  $A = [a_1, \dots, a_n] \in \mathbb{R}^{m \times n}$  есть разбиение матрицы на столбцы и  $m \geq n$ . Если  $A_r = [a_1, \dots, a_r]$ , то при  $r = 1:n-1$  имеем

$$\sigma_1(A_{r+1}) \geq \sigma_1(A_r) \geq \sigma_2(A_{r+1}) \geq \dots \geq \sigma_r(A_{r+1}) \geq \sigma_r(A_r) \geq \sigma_{r+1}(A_{r+1}).$$

*Доказательство.* Применить следствие 8.1.4 к матрице  $A^T A$ .  $\square$

Этот последний результат говорит о том, что при добавлении столбцов к матрице наибольшее сингулярное число возрастает, а наименьшее сингулярное число убывает.

**Пример 8.3.2.**

$$A = \begin{bmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 8 & 14 \\ 5 & 10 & 15 \end{bmatrix} \Rightarrow \begin{cases} \sigma(A_1) = \{7, 4162\} \\ \sigma(A_2) = \{19, 5377; 1, 8095\} \\ \sigma(A_3) = \{35, 1272; 2, 4654; 0, 0000\}, \end{cases}$$

что подтверждает следствие 8.3.3.

Следующий результат является теоремой Виландта – Хоффмана для сингулярных чисел:

**Теорема 8.3.4.** Если  $A$  и  $A + E$  принадлежат  $\mathbb{R}^{m \times n}$  и  $m \geq n$ , то

$$\sum_{k=1}^n (\sigma_k(A + E) - \sigma_k(A))^2 \leq \|E\|_F^2.$$

*Доказательство.* Применить теорему 8.1.5 к матрицам  $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$  и  $\begin{bmatrix} 0 & (A + E)^T \\ A + E & 0 \end{bmatrix}$ .  $\square$

**Пример 8.3.3.** Если

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \text{и} \quad A + E = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6.01 \end{bmatrix},$$

то

$$\sum_{k=1}^2 (\sigma_k(A + E) - \sigma_k(A))^2 = 0.472 \times 10^{-4} \leq 10^{-4} = \|E\|_F^2.$$

См. пример 8.3.1.

Мы будем говорить, что  $k$ -мерные подпространства  $S \subseteq \mathbb{R}^n$  и  $T \subseteq \mathbb{R}^m$  образуют пару сингулярных подпространств матрицы  $A \in \mathbb{R}^{m \times n}$ , если из  $x \in S$  и  $y \in T$  следует, что  $Ax \in T$  и  $A^T y \in S$ . Следующий результат касается возмущения пар сингулярных подпространств.

**Теорема 8.3.5.** Пусть даны матрицы  $A, E \in \mathbb{R}^{m \times n}$ , и пусть матрицы  $V \in \mathbb{R}^{n \times n}$  и  $U \in \mathbb{R}^{m \times m}$  ортогональные. Предположим, что

$$V = [V_1 \quad V_2] \quad U = [U_1 \quad U_2]$$

$$\begin{matrix} k & & & \\ & n-k & & \\ & & k & \\ & & & m-k \end{matrix} \quad \begin{matrix} & & & \\ & & k & \\ & & & m-k \end{matrix}$$

и  $\text{range}(V_1)$  и  $\text{range}(U_1)$  образуют пару сингулярных подпространств матрицы  $A$ . Пусть

$$U^H A V = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \\ k & n-k \end{bmatrix} \begin{matrix} & k \\ & m-k \end{matrix},$$

$$U^H E V = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \\ k & n-k \end{bmatrix} \begin{matrix} & k \\ & m-k \end{matrix}.$$

и  $\varepsilon = \| [E_{21} E_{12}^T] \|_F$ . Предположим, что

$$0 < \delta = \min_{\substack{\sigma \in \sigma(A_{11}) \\ \gamma \in \sigma(A_{22})}} |\sigma - \gamma| - \|E_{11}\|_2 - \|E_{22}\|_2.$$

Если  $\varepsilon/\delta \leq 1/2$ , то существуют матрицы  $P \in \mathbb{R}^{(n-k) \times k}$  и  $Q \in \mathbb{R}^{(m-k) \times k}$ , удовлетворяющие неравенству

$$\left\| \begin{bmatrix} Q \\ P \end{bmatrix} \right\|_F \leq 2 \frac{\varepsilon}{\delta},$$

такие, что  $\text{range}(V_1 + V_2 Q)$  и  $\text{range}(U_1 + U_2 P)$  являются парой сингулярных подпространств матрицы  $A + E$ .

*Доказательство.* См. [Stewart, 1973в], теорема 6.4.  $\square$

Грубо говоря, эта теорема утверждает, что изменения порядка  $O(\varepsilon)$  в матрице  $A$  могут вызвать изменения сингулярных подпространств порядка  $\varepsilon/\delta$ , где  $\delta$  – мера изолированности соответствующих сингулярных чисел.

**Пример 8.3.4.** Матрица  $A = \text{diag}(2.000, 1.001, 0.999) \in \mathbb{R}^{4 \times 3}$  имеет пары сингулярных подпространств  $(\text{span}\{v_i\}, \text{span}\{u_i\})$ ,  $i = 1, 2, 3$ , где  $v_i = e_i^{(3)}$  и  $u_i = e_i^{(4)}$ . Пусть

$$A + E = \begin{bmatrix} 2.000 & 0.010 & 0.010 \\ 0.010 & 1.001 & 0.010 \\ -0.010 & 0.010 & 0.999 \\ 0.010 & 0.010 & 0.010 \end{bmatrix}.$$

Соответствующие столбцы матриц

$$\hat{U} = [\hat{u}_1 \hat{u}_2 \hat{u}_3] = \begin{bmatrix} 0.9999 & -0.0144 & 0.0007 \\ 0.0101 & 0.7415 & 0.6708 \\ 0.0101 & 0.6707 & -0.7616 \\ 0.0051 & 0.0138 & -0.0007 \end{bmatrix},$$

$$\hat{V} = [\hat{v}_1 \hat{v}_2 \hat{v}_3] = \begin{bmatrix} 0.9999 & -0.0143 & 0.0007 \\ 0.0101 & 0.7416 & 0.6708 \\ 0.0101 & 0.6707 & -0.7416 \end{bmatrix}$$

определяют пары сингулярных подпространств матрицы  $A + E$ . Заметим, что пара  $(\text{span}\{\hat{v}_i\}, \text{span}\{\hat{u}_i\})$  близка к  $(\text{span}\{v_i\}, \text{span}\{u_i\})$  при  $i = 1$ , но не при  $i = 2$  и  $3$ . С другой стороны, пара сингулярных подпространств  $(\text{span}\{\hat{v}_2, \hat{v}_3\}, \text{span}\{\hat{u}_2, \hat{u}_3\})$  близка к  $(\text{span}\{v_2, v_3\}, \text{span}\{u_2, u_3\})$ .

### 8.3.2. SVD-алгоритм

Теперь мы покажем, как некоторый вариант QR-алгоритма можно использовать для вычисления SVD матрицы  $A \in \mathbb{R}^{m \times n}$  с  $m \geq n$ . На первый взгляд это кажется очевидным. Уравнение (8.3.1) наводит на мысль, что мы можем

- Сформировать матрицу  $C = A^T A$ .
- Использовать симметричный QR-алгоритм для вычисления  $V_1^T C V_1 = \text{diag}(\sigma_i^2)$ .
- Применить QR-разложение с выбором ведущего элемента по столбцу к матрице  $AV_1$ , получая  $U^T (AV_1)\Pi = R$ .

Поскольку матрица  $R$  имеет ортогональные столбцы, матрица  $U^T A (V_1 \Pi)$  диагональная. Однако, как мы видели в примере 5.3.1, формирование матрицы  $A^T A$  может привести к потере точности. В данном случае ситуация не совсем такая же плохая, поскольку для вычисления матрицы  $U$  используется исходная матрица  $A$ .

Более предпочтительный метод вычисления SVD описан в работе Голуба и Кахана [Golub, Kahan, 1965]. Их алгоритм находит матрицы  $U$  и  $V$  одновременно, неявно применяя симметричный QR-алгоритм к матрице  $A^T A$ . Первый шаг состоит в приведении матрицы  $A$  к верхней двухдиагональной форме посредством алгоритма 5.4.2:

$$U_B^T A V_B = \begin{bmatrix} B \\ 0 \end{bmatrix},$$

$$B = \begin{bmatrix} d_1 & f_1 & & \dots & 0 \\ 0 & d_2 & \ddots & & \vdots \\ \ddots & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & f_{n-1} \\ 0 & \dots & 0 & d_n \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

Таким образом, остается вычислить SVD матрицы  $B$ . С этой целью рассмотрим применение QR-шага с неявным сдвигом (алгоритм 8.2.2) к трехдиагональной матрице  $T = B^T B$ :

- Вычислить собственное значение  $\lambda$  подматрицы

$$T(m:n, m:n) = \begin{bmatrix} d_m^2 & d_m f_m \\ d_m f_m & f_m^2 + d_n^2 \end{bmatrix}, \quad m = n - 1,$$

охватывающей элемент  $d_n^2 + f_m^2$ .

- Вычислить величины  $c_1 = \cos(\theta_1)$  и  $s_1 = \sin(\theta_1)$ , такие, что

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} d_1^2 - \lambda \\ d_1 f_1 \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix},$$

и положить  $G_1 = G(1, 2, \theta_1)$ .

- Вычислить вращения Гивенса  $G_2, \dots, G_{n-1}$ , такие, что если  $Q = G_1 \dots G_{n-1}$ , то матрица  $Q^T T Q$  трехдиагональная и  $Qe_1 = G_1 e_1$ .

Заметим, что эти вычисления требуют явного формирования матрицы  $B^T B$ , которое, как мы видели, является нежелательным с вычислительной точки зрения.

Предположим, что вместо этого мы применяем указанное выше вращение Гивенса  $G_1$  непосредственно к матрице  $B$ . Применимально к случаю  $n = 6$  это дает

$$B \leftarrow BG_1 = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ + & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Затем мы можем определить вращения Гивенса  $U_1, V_2, U_2, \dots, V_{n-1}, U_{n-1}$ , перегоняющие нежелательный ненулевой элемент в нижнюю часть двухдиагонали:

$$B \leftarrow U_1^T B = \begin{bmatrix} \times & \times & + & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix},$$

$$B \leftarrow BV_2 = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & + & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix},$$

$$B \leftarrow U_2^T B = \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & + & 0 & 0 \\ 0 & 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix},$$

и так далее. Это процесс закончится появлением новой двухдиагональной матрицы  $\bar{B}$ , связанной с  $B$  следующим образом:

$$\bar{B} = (U_{n-1}^T \dots U_1^T) B (G_1 V_2 \dots V_{n-1}) = \bar{U}^T B \bar{V}.$$

Каждая матрица  $V_i$  имеет вид  $V_i = G(i, i+1, \theta_i)$ , где  $i = 2:n-1$ ; следовательно,  $V e_1 = Q e_1$ . В силу теоремы о неявном  $Q$  мы можем утверждать, что матрицы  $V$  и  $Q$  в основном равные. Таким образом, мы можем неявно выполнять переход от  $T$  к  $\bar{T} = \bar{B}^T \bar{B}$ , работая непосредственно с двухдиагональной матрицей  $B$ .

Конечно, для этого требуется, чтобы трехдиагональные матрицы были неприводимыми. Поскольку поддиагональные элементы матрицы  $B^T B$  формируют величины  $d_{i-1}$  и  $f_i$ , понятно, что мы должны убедиться в отсутствии нулей в двухдиагональной ленте. Если  $f_k = 0$  при некотором  $k$ , то

$$B = \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \\ k & n-k \end{bmatrix}$$

и исходная SVD-проблема расщепляется на две меньшие задачи с матрицами  $B_1$  и  $B_2$ . Если  $d_k = 0$  при некотором  $k$ , то умножением на последовательность преобразований Гивенса можно обнулить  $f_k$ . Например, если  $n = 6$  и  $k = 3$ , то вращением строк в плоскостях (3, 4), (3, 5) и (3, 6) мы можем обнулить элементы третьей строки:

$$\begin{aligned} B &= \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \\ &\xrightarrow{(3,5)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & + \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,6)} \begin{bmatrix} \times & \times & 0 & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \end{bmatrix} \end{aligned}$$

Таким образом, мы можем осуществлять расщепление всякий раз, когда  $f_1 \dots f_{n-1} = 0$  или  $d_1 \dots d_{n-1} = 0$ .

**Алгоритм 8.3.1 (SVD-шаг Голуба – Кахана).** Для заданной двухдиагональной матрицы  $B \in \mathbb{R}^{m \times n}$ , не имеющей нулей на главной диагонали или наддиагонали, следующий алгоритм записывает в матрицу  $B$  двухдиагональную матрицу  $\bar{B} = \bar{U}^T B \bar{V}$ , где  $\bar{U}$  и  $\bar{V}$  – ортогональные матрицы и матрица  $V$  в основном та ортогональная матрица, которая могла бы быть получена применением алгоритма 8.2.2 к матрице  $T = B^T B$ .

Пусть  $\mu$  – собственное значение нижней угловой  $2 \times 2$  подматрицы матрицы  $T = B^T B$ , охватывающей элемент  $t_{nn}$ .

$$y = t_{11} - \mu$$

$$z = t_{12}$$

**for**  $k = 1:n-1$

Определить  $c = \cos(\theta)$  и  $s = \sin(\theta)$ , такие, что

$$[yz] \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = [* \ 0]$$

$$B = BG(k, k+1, \theta)$$

$$y = b_{kk}; \quad z = b_{k+1,k}$$

Определить  $c = \cos(\theta)$  и  $s = \sin(\theta)$ , такие, что

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

$$B = G(k, k+1, \theta)^T B$$

**if**  $k < n-1$

$$y = b_{k,k+1}; \quad z = b_{k,k+2}$$

**end**

**end**

Эффективная реализация этого алгоритма хранит диагонали и наддиагонали матрицы  $B$  в векторах  $a(1:n)$  и  $f(1:n-1)$  и требует  $30n$  флопов и  $2n$  вычислений квадратных корней. Накопление матрицы  $U$  требует  $6n^2$  флопов. Накопление матрицы  $V$  требует  $6n^2$  флопов.

Обычно после нескольких вышеописанных SVD-итераций наддиагональный элемент  $f_{n-1}$  становится пренебрежимо малым. Критерий малости внутренних элементов ленты матрицы  $B$  обычно имеет вид

$$\begin{aligned} |f_i| &\leq \varepsilon (|d_i| + |d_{i+1}|), \\ |d_i| &\leq \varepsilon \|B\|, \end{aligned}$$

где  $\varepsilon$  – малая величина порядка машинной точности, а  $\|\cdot\|$  – некоторая удобная для вычислений норма.

Объединение алгоритма 5.4.2 (двуходиагонализация), алгоритма 8.3.1, и упомянутого способа расщепления дает

**Алгоритм 8.3.2 (SVD-алгоритм).** Для заданных матриц  $A \in \mathbb{R}^{m \times n}$  ( $m \geq n$ ) и  $\varepsilon$  – малой величины порядка машинной точности, следующий алгоритм записывает в матрицу  $A$  матрицу  $U^T A V = D + E$ , где  $U \in \mathbb{R}^{m \times n}$  ортогональная,  $V \in \mathbb{R}^{n \times n}$  ортогональная,  $D \in \mathbb{R}^{m \times n}$  диагональная матрицы, а матрица  $E$  удовлетворяет приближенному равенству  $\|E_2\|_2 \approx \varepsilon \|A\|_2$ .

Используя алгоритм 5.4.2, выполнить двухдиагонализацию

$$B \leftarrow (U_1 \dots U_n)^T A (V_1 \dots V_{n-2})$$

**until**  $q = n$

Положить  $a_{i,i+1}$  равным нулю, если  $|a_{i,i+1}| \leq \varepsilon (|a_{ii}| + |a_{i+1,i+1}|)$   
для всех  $i = 1:n-1$ .

Найти наибольшее  $q$  и наименьшее  $p$ , такие, что если

$$B = \begin{bmatrix} B_{11} & 0 & 0 \\ 0 & B_{22} & 0 \\ 0 & 0 & B_{33} \end{bmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix},$$

то подматрица  $B_{33}$  – диагональная, а подматрица  $B_{22}$  имеет ненулевую наддиагональ.

**if**  $q < n$

**if** некоторый диагональный элемент в  $B_{22}$  равен нулю, то обнулить наддиагональные элементы той же строки.

**else**

Применить алгоритм 8.3.1 к  $B_{22}$ ,  
 $B = \text{diag}(I_p, U, I_{q+m-n})^T B \text{diag}(I_p, V, I_q)$

**end**

**end**

**end**

Объем вычислений этого алгоритма и его вычислительные свойства обсуждались в § 5.4.5 и § 5.5.8.

**Пример 8.3.5.** Если алгоритм 8.3.2 применить к матрице

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix},$$

то наддиагональные элементы будут сходиться к нулю следующим образом:

Итерация	$O( a_{21} )$	$O( a_{32} )$	$O( a_{43} )$
1	$10^0$	$10^0$	$10^0$
2	$10^0$	$10^0$	$10^0$
3	$10^0$	$10^0$	$10^0$
4	$10^0$	$10^{-1}$	$10^{-2}$
5	$10^0$	$10^{-1}$	$10^{-8}$
6	$10^0$	$10^{-1}$	$10^{-27}$
7	$10^0$	$10^{-1}$	сошелся
8	$10^0$	$10^{-4}$	
9	$10^{-1}$	$10^{-14}$	
10	$10^{-1}$	сошелся	
11	$10^{-4}$		
12	$10^{-12}$		
13	сошелся		

Наблюдается кубическая сходимость.

### Задачи

**8.3.1.** Показать, что если  $B \in \mathbb{R}^{n \times n}$  – верхняя двухдиагональная матрица, имеющая кратное сингулярное значение, то  $B$  должна иметь нуль на главной диагонали или наддиагонали.

**8.3.2.** Получить формулу для собственных векторов матрицы  $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$  в терминах сингулярных векторов матрицы  $A \in \mathbb{R}^{m \times n}$ , где  $m \geq n$ .

**8.3.3.** Дать алгоритм приведения комплексной матрицы  $A$  к вещественной двухдиагональной форме, используя комплексные преобразования Хаусхолдера.

**8.3.4.** Связать сингулярные числа и векторы матрицы  $A = B + iC$  ( $B, C \in \mathbb{R}^{m \times n}$ ) с сингулярными числами и векторами матрицы  $\begin{bmatrix} B & -C \\ C & B \end{bmatrix}$ .

**8.3.5.** Закончить доказательство теоремы 8.3.1.

### Замечания и литература к § 8.3.1

SVD и его математические свойства обсуждаются в

Amir-Moez A. R. (1965). External Properties of Linear Transformations and Geometry of Unitary Spaces, Texas Tech University Mathematics Series, no. 243, Lubbock, Texas.

Stewart G. W. (1973b). "Error and Perturbation Bounds for Subspaces Associated with Certain Eigenvalue Problems", SIAM Review 15, 727–64.

Wedin P. A. (1972). "Perturbation Bounds in Connection with the Singular Value Decomposition", BIT 12, 99–111.

Две последние работы посвящены устойчивости SVD разложения. К другим работам, связанным с устойчивостью сингулярных чисел, относятся

- Ruhe A. (1975). "On the Closeness of Eigenvalues and Singular Values for Almost Normal Matrices", Lin. Alg. and Its Applic. 11, 87–94.
- Stewart G. W. (1979). "A Note on the Perturbation of Singular Values", Lin. Alg. and Its Applic. 28, 213–16.
- Stewart G. W. (1984). "A Second Order Perturbation Expansion for Small Singular Values", Lin. Alg. and Its Applic. 56, 231–236.

Идея приспособить симметрический QR-алгоритм к вычислению SVD впервые появилась в работе

- Golub G. H. and Kahan W. (1965). "Calculating the Singular Values and Pseudo-Inverse of a Matrix", SIAM J. Num. Anal. Ser. B 2, 205–224.

а первая работающая программа в

- Businger P. A. and Golub G. H. (1969). "Algorithm 358: Singular Value Decomposition of a Complex Matrix", Comm. Assoc. Comp. Mach. 12, 564–65.

Алгольная процедура из следующей статьи легла в основу SVD подпрограмм пакетов EISPACK 2 и LINPACK:

- Golub G. H. and Reinsch C. (1970). "Singular Value Decomposition and Least Squares Solutions", Numer. Math. 14, 403–20. See also HACLA, pp. 134–51.

Другие фортранные SVD программы приведены в [Lawson, Hanson, SLS] гл. 9 и в

- Forsythe G. E., Malcolm M. and Moler C. B. (1977). Computer Methods for Mathematical Computations, Prentice-Hall, Englewood Cliffs, NJ.

Интересные алгоритмические разработки, связанные с SVD, появились в

- Cuppen J. J. M. (1983). "The Singular Value Decomposition in Product Form", SIAM J. Sci. and Stat. Comp. 4, 216–222.

- Dongarra J. J. (1983). "Improving the Accuracy of Computed Singular Values", SIAM J. Sci. and Stat. Comp. 4, 712–719.

- Van Huffel S., Vandewalle J., and Haegemans A. (1987). "An Efficient and Reliable Algorithm for Computing the Singular Subspace of a Matrix Associated with its Smallest Singular Values", J. Comp. and Appl. Math. 19, 313–330.

SVD имеет много приложений в статистике. Смотри

- Hammarling S. J. (1985). "The Singular Value Decomposition in Multivariate Statistics", ACM SIGNUM Newsletter 20, 2–25.

Напомним об установленной в задаче 4.2.9 связи между полярным разложением  $A = QP$  и SVD матрицы  $A = U\Sigma V^T$ . Конечно, алгоритм 8.3.2 может быть использован для вычисления  $Q = UV^T$ . Однако часто предпочтительнее воспользоваться алгоритмом, описанным в

- Higham N. J. (1986a). "Computing Polar Decomposition – with Applications", SIAM J. Sci. and Stat. Comp. 7, 1160–1174.

Наконец, заметим, что были найдены некоторые специальные SVD-процедуры для матриц специальной структуры

- Bunse-Gerstner A. and Gragg W. B. (1988). "Singular Value Decompositions of Complex Symmetric Matrices", J. Comp. Appl. Math. 21, 41–54.

## 8.4. Некоторые специальные методы

Используя богатую математическую структуру симметрической проблемы собственных значений, можно придумать полезные альтернативы симметрическому QR-алгоритму. Многие из этих методов предпочтительнее, когда требуется найти только несколько собственных значений и/или собственных векторов. Три таких метода описаны в этом разделе: бисекция, итерации с отношением Рэлея, ортогональные итерации с ускорением Ритца.

### 8.4.1. Бисекция

Пусть  $T_r$  означает ведущую главную подматрицу размера  $r \times r$  матрицы

$$T = \begin{bmatrix} a_1 & b_1 & & \cdots & 0 \\ b_1 & a_2 & \ddots & & \vdots \\ \ddots & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & & b_{n-1} & a_n \end{bmatrix} \quad (8.4.1)$$

и определены многочлены  $p_r(x) = \det(T_r - xI)$ ,  $r = 1:n$ . Можно показать, используя обычную формулу для определителя, что

$$p_r(x) = (a_r - x)p_{r-1}(x) - b_{r-1}^2 p_{r-2}(x) \quad (8.4.2)$$

при  $r = 2:n$ , если положить  $p_0(x) = 1$ . Поскольку значение многочлена  $p_n(x)$  можно оценить за  $O(n)$  флопов, его корни имеет смысл искать, используя метод бисекции. Например, если  $p_n(y)p_n(z) < 0$  и  $y < z$ , то итерационный процесс

```

while |y - z| > ε(|y| + |z|)
    x = (y + z)/2
    if p_n(x)p_n(y) < 0
        z = x
    else
        y = x
    end
end

```

обязательно закончится с  $(y + z)/2$  равным приближенному корню многочлена  $p_n(x)$ , т. е. приближенному собственному значению матрицы  $T$ . Эти итерации сходятся линейно, причем погрешность уменьшается примерно в два раза на каждом шаге.

Иногда нужно вычислить  $k$ -е максимальное собственное значение матрицы  $T$  для некоторой заданной величины  $k$ . Это можно сделать эффективно, используя идею бисекции и следующий классический результат:

**Теорема 8.4.1 (Свойство последовательности Штурма).** *Если трехдиагональная матрица (8.4.1) неприводимая, то собственные значения подматрицы  $T_{r-1}$  строго разделяют собственные значения подматрицы  $T_r$ :*

$$\lambda_r(T_r) < \lambda_{r-1}(T_{r-1}) < \lambda_{r-1}(T_r) < \dots < \lambda_2(T_r) < \lambda_1(T_{r-1}) < \lambda_1(T_r).$$

Более того, если  $a(\lambda)$  означает число перемен знака в последовательности

$$\{p_0(\lambda), p_1(\lambda), \dots, p_n(\lambda)\},$$

то  $a(\lambda)$  равно числу тех собственных значений матрицы  $T$ , которые меньше чем  $\lambda$ . Здесь  $p_r(x)$ —многочлены, определенные равенством (8.5.2), и считается, что  $p_r(x)$  имеет отрицательный знак, если  $p_r(\lambda) = 0$ .

**Доказательство.** Из следствия 8.1.4 следует, что собственные значения подматрицы  $T_{r-1}$  нестрого разделяют собственные значения подматрицы  $T_r$ . Для того чтобы доказать, что разделение должно быть строгим, предположим, что  $p_r(\mu) = p_{r-1}(\mu) = \dots = 0$  при некоторых  $r$  и  $\mu$ . Тогда из (8.4.2) и предположения о неприводимости матрицы  $T$  следует, что  $p_0(\mu) = p_1(\mu) = \dots = p_r(\mu) = 0$ —противоречие. Следовательно, мы должны иметь строгое разделение.

Утверждение о величине  $a(\lambda)$  доказано в [Wilkinson, АЕР], с 300–301.  $\square$

**Пример 8.4.1.** Если

$$T = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix},$$

то  $\lambda(T) \approx \{0.254, 1.82, 3.18, 4.74\}$ . Последовательность

$$\{p_0(2), p_1(2), p_2(2), p_3(2)\} = \{1, -1, -1, 0, 1\}$$

подтверждает, что имеются два собственных значения, меньших, чем  $\lambda = 2$ .

Пусть нам нужно вычислить  $\lambda_k(T)$ . Из теоремы о кругах Гершгорина (теорема 7.2.1) следует, что  $\lambda_k(T) \in [y, z]$ , где

$$y = \min_{1 \leq i \leq n} a_i - |b_i| - |b_{i-1}|, \quad z = \max_{1 \leq i \leq n} a_i + |b_i| + |b_{i-1}|,$$

если определить  $b_0 = b_n = 0$ . Из свойства последовательности Штурма видно, что с этими начальными данными итерации

```

while |z - y| > u(|y| + |z|)
    b = (y + z)/2
    if a(x) ≥ k
        z = x
    else
        y = x
    end
end
(8.4.3)

```

породят последовательность подинтервалов, которые постоянно уменьшаются вдвое, но всегда содержат  $\lambda_k(T)$ .

**Пример 8.4.2.** Если (8.4.3) применить к матрице из примера 8.4.1 с  $k = 2$ , то будут вычисляться величины, показанные в табл. 8.4.1. Окончательный результат позволяет заключить, что  $\lambda_2(T) \in [1.7969, 1.9375]$ . Заметим, что  $\lambda_2(T) \approx 1.82$ .

$y$	$z$	$x$	$a(x)$
0.0000	5.0000	2.5000	2
0.0000	2.5000	1.2500	1
1.2500	2.5000	1.3750	1
1.3750	2.5000	1.9375	2
1.3750	1.9375	1.6563	1
1.6563	1.9375	1.7969	1

В процессе выполнения алгоритма (8.4.3) появляется информация о разложении других собственных значений. Систематически следя за этой информацией можно предложить эффективную схему вычисления «смежных» подмножеств спектра  $\lambda(T)$ , т. е.  $\lambda_k(T), \lambda_{k+1}(T), \dots, \lambda_{k+j}(T)$ . См. [Barth, Martin, Wilkinson, 1967].

Если требуются отдельные собственные значения симметрической матрицы  $A$  общего вида, то необходимо выполнить трехдиагонализацию  $T = U_0^T T U_0$ , прежде чем можно будет применять описанную выше бисекцию. Это можно сделать, используя алгоритм 8.2.1 или метод Ланцоша, рассматриваемый в следующей главе. В любом случае соответствующие собственные векторы могут быть легко найдены

посредством обратных итераций (см. § 7.6), так как трехдиагональную систему можно решить за  $O(n)$  флопов.

В тех приложениях, где исходная матрица  $A$  уже имеет трехдиагональный вид, бисекция вычисляет собственные значения с малой относительной погрешностью, независимо от их величины. Это отличает ее от трехдиагональных QR-итераций, где для вычисленных собственных значений  $\tilde{\lambda}_i$  можно гарантировать только наличие малой абсолютной погрешности  $|\tilde{\lambda}_i - \lambda_i(T)| \approx u \|T\|_2$ .

Наконец, можно вычислять отдельные собственные значения симметричной матрицы, используя  $LDL^T$ -разложение (см. § 4.2) и применяя теорему Сильвестра об инерции (теорема 8.1.12). Если

$$A - \mu I = LDL^T, \quad A = A^T \in \mathbb{R}^{n \times n},$$

есть  $LDL^T$ -разложение матрицы  $A - \mu I$  с матрицей  $D = \text{diag}(d_1, \dots, d_n)$ , то число отрицательных  $d_i$  равно числу собственных значений  $\lambda_i(A)$ , которые меньше  $\mu$ . Детали см. в [Parlett, SEP], с. 46.

#### 8.4.2. Итерации с отношением Рэлея

Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и  $x$  – заданный ненулевой вектор длины  $n$ . Простое дифференцирование показывает, что величина

$$\lambda = r(x) \equiv \frac{x^T Ax}{x^T x}$$

минимизирует  $\|(A - \lambda I)x\|_2$ . Скалярную величину  $r(x)$  называют отношением Рэлея вектора  $x$ . Ясно, что если  $x$  – приближенный собственный вектор, то  $r(x)$  является разумным выбором для соответствующего ему собственного значения. С другой стороны, если  $\lambda$  – приближенное собственное значение, то теория обратных итераций говорит нам, что решение уравнения  $(A - \lambda I)x = b$  будет почти всегда хорошей аппроксимацией собственного вектора.

Объединение этих двух идей естественным образом приводит к *итерациям с отношением Рэлея*:

```

 $x_0$  задан,  $\|x_0\|_2 = 1$ 
for  $k = 0, 1, \dots$ 
   $\mu_k = r(x_k)$  (8.4.4)
  Решить уравнение  $(A - \mu_k I)z_{k+1}$  относительно  $z_{k+1}$ 
   $x_{k+1} = z_{k+1}/\|z_{k+1}\|_2$ 
end
```

Заметим, что для любого  $k$  мы имеем равенство  $(A + E_k)z_{k+1} = \mu_k z_{k+1}$ , где матрица возмущения  $E_k$  есть  $E_k = -x_k z_{k+1}^T / \|z_{k+1}\|_2^2$ . Из теоремы 7.2.2 следует, что  $|\mu_k - \lambda| \leq 1/\|z_{k+1}\|_2$  для некоторого  $\lambda \in \lambda(A)$ .

**Пример 8.4.3.** Если (8.4.4) применить к матрице

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 3 & 6 & 10 & 15 & 21 \\ 1 & 4 & 10 & 20 & 35 & 56 \\ 1 & 5 & 15 & 35 & 70 & 126 \\ 1 & 6 & 21 & 56 & 126 & 252 \end{bmatrix}$$

с  $x_0 = (1, 1, 1, 1, 1, 1)^T/6$ , то

$k$	$\mu_k$
0	153.8333
1	120.0571
2	49.5011
3	13.8687
4	15.4959
5	15.5534

Эти итерации сходятся к собственному значению  $\lambda = 15.5534732737$ .

Парлетт [Parlett, 1974b] показал, что (8.4.4) сходится глобально и что сходимость почти всегда кубическая. Мы демонстрируем это для случая  $n = 2$ . Без потери общности можем считать, что  $A = \text{diag}(\lambda_1, \lambda_2)$  с  $\lambda_1 > \lambda_2$ . Обозначая компоненты вектора  $x_k$  следующим образом:

$$x_k = \begin{bmatrix} c_k \\ s_k \end{bmatrix}, \quad c_k^2 + s_k^2 = 1,$$

получим, что  $\mu_k = \lambda_1 c_k^2 + \lambda_2 s_k^2$  и

$$z_{k+1} = \frac{1}{\lambda_1 - \lambda_2} \begin{bmatrix} c_k/s_k^2 \\ -s_k/c_k^2 \end{bmatrix}.$$

Вычисление показывает, что

$$c_{k+1} = \frac{c_k^3}{\sqrt{c_k^6 + s_k^6}}, \quad s_{k+1} = \frac{-s_k^3}{\sqrt{c_k^6 + s_k^6}}.$$

Из этого уравнения видно, что вектор  $x_k$  сходится кубически либо к подпространству  $\text{span}\{e_1\}$ , либо к подпространству  $\text{span}\{e_2\}$  при условии, что  $|c_k| \neq |s_k|$ .

Детали, связанные с практическим применением итераций с отношением Рэлея, можно найти в [Parlett, 1974b].

Интересно заметить связь между итерациями с отношением Рэлея и симметричным QR-алгоритмом. Пусть мы применяем последний к трехдиагональной матрице  $T \in \mathbb{R}^{n \times n}$  со сдвигом  $\sigma = e_n^T T e_n = t_{nn}$ . Если  $T - \sigma I = QR$ , то мы получим матрицу  $T = RQ + \sigma I$ . Из уравнения  $(T - \sigma I)Q = R^T$  следует, что

$$(T - \sigma I)q_n = r_{nn}e_n,$$

где  $q_n$  – последний столбец ортогональной матрицы  $Q$ . Следовательно, если мы применим (8.4.4) с вектором  $x_0 = e_n$ , то  $x_1 = q_n$ .

### 8.4.3. Ортогональные итерации с ускорением Ритца

Напомним метод ортогональных итераций из § 7.3:

$Q_0 \in \mathbb{R}^{n \times p}$  задана с  $Q_0^T Q_0 = I_p$

for  $k = 1, 2, \dots$

$$Z_k = A Q_{k-1}$$

$$Z_k = Q_k R_k \text{ (QR-разложение)}$$

end

(8.4.5)

Пусть  $A$  – симметричная матрица с разложением Шура  $Q^T A Q = \text{diag}(\lambda_i)$ , где  $Q = [q_1, \dots, q_n]$  и  $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$ . Из теоремы 7.3.2 следует, что если  $d = \text{dist}\{D_p(A)\}$ ,

$\text{range}(Q_0) \} < 1$ , то

$$\text{dist}\{D_p(A), \text{range}(Q_k)\} \leq \frac{1}{\sqrt{1+d^2}} \left| \frac{\lambda_{p+1}}{\lambda_p} \right|^k,$$

где  $D_p(A) = \text{span}\{q_1, \dots, q_p\}$ . Более того, из анализа, выполненного в § 7.3, мы знаем, что если  $R_k = (r_{ij}^{(k)})$ , то

$$|r_{ii}^{(k)} - \lambda_i| = O\left(\left|\frac{\lambda_{i+1}}{\lambda_i}\right|^k\right), \quad i = 1:p.$$

Это может быть неприемлемо: низкая скорость сходимости, если собственные значения  $\lambda_i$  и  $\lambda_{i+1}$  имеют почти равные модули.

Эта трудность может быть практически преодолена заменой матрицы  $Q_k$  ее векторами Ритца на каждом шаге:

```

 $Q_0 \in \mathbf{R}^{n \times p}$  задана с  $Q_0^T Q_0 = I_p$ 
for  $k = 1, 2, \dots$ 
 $Z_k = A Q_{k-1}$ 
 $\tilde{Q}_k R_k = Z_k$  (QR-разложение)
 $S_k = \tilde{Q}_k^T A \tilde{Q}_k$ 
 $U_k^T S_k U_k = D_k$  (разложение Шура)
 $Q_k = \tilde{Q}_k U_k$ 
end

```

Можно показать, что если

$$D_k = \text{diag}(\theta_1^{(k)}, \dots, \theta_p^{(k)}) \quad |\theta_1^{(k)}| \geq \dots \geq |\theta_p^{(k)}|,$$

то

$$|\theta_i^{(k)} - \lambda_i(A)| = O\left(\left|\frac{\lambda_{p+1}}{\lambda_i}\right|^k\right), \quad i = 1:p.$$

Таким образом, числа Ритца  $\theta_i^{(k)}$  сходятся со значительно более предпочтительной скоростью, чем величины  $r_{ii}^{(k)}$  в (8.4.5). Детали см. в [Stewart, 1969].

**Пример 8.4.4.** Если мы применим (8.4.6) к матрицам

$$A = \begin{bmatrix} 100 & 1 & 1 & 1 \\ 1 & 99 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \text{и} \quad Q_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

то

$k$	$\text{dist}\{D_2(A), Q_k\}$
0	$0.2 \times 10^{-1}$
1	$0.5 \times 10^{-3}$
2	$0.1 \times 10^{-4}$
3	$0.3 \times 10^{-6}$
4	$0.8 \times 10^{-8}$

Ясно, что имеет место сходимость со скоростью  $(2/99)^k$ .

**Задачи**

**8.4.1.** Пусть  $\lambda$  есть собственное значение симметрической трехдиагональной матрицы  $T$ . Показать, что если  $\lambda$  имеет алгебраическую кратность  $k$ , то по меньшей мере  $k - 1$  поддиагональных элементов матрицы  $T$  равны нулю.

**8.4.2.** Пусть матрица  $A$  — симметричная и имеет ширину ленты  $p$ . Показать, что если мы выполним QR-шаг со сдвигом:  $A - \mu I = QR$ ,  $A = RQ + \mu I$ , то матрица  $A$  будет иметь ширину ленты  $p$ .

**8.4.3.** Пусть  $B \in \mathbb{R}^{n \times n}$  — верхняя двухдиагональная матрица с диагональными элементами  $d(1:n)$  и наддиагональными элементами  $f(1:n-1)$ . Предложить и доказать вариант теоремы 8.4.1 для сингулярных чисел.

**Замечания и литература к § 8.4**

Подпрограмма бисекций есть в EISPACK. Алгоритмическая программа, на которой она основана, описана в

Barth W., Martin R. S., and Wilkinson J. H. (1967). "Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection", Numer. Math. 9, 386–93. See also HACLA, pp. 249–56.

Еще один способ вычислять отдельные подмножества собственных значений дает вещественный QR-алгоритм. В этом методе сдвиг определяют, используя метод Ньютона. Это позволяет «направлять» итерацию к требуемым собственным значениям. Смотри

Reinsch C. and Bauer F. L. (1968). "Rational QR Transformation with Newton's Shift for Symmetric Tridiagonal Matrices", Numer. Math. 11, 264–72. See also HACLA, pp. 257–65.

К работам, касающимся симметричного QR-алгоритма для ленточных матриц, относятся

Martin R. S. and Wilkinson J. H. (1967). "Solution of Symmetric and Unsymmetric Band Equations and the Calculation of Eigenvectors of Band Matrices", Numer. Math. 9, 279–301. See also HACLA, pp. 70–92.

Martin R. S., Reinsch C., and Wilkinson J. H. (1970). "The QR Algorithm for band Symmetric Matrices", Numer. Math. 16, 85–92. See also HACLA, pp. 266–72.

Следующие работы посвящены методу одновременных итераций для симметрических матриц

Clint M. and Jennings A. (1970). "The Evaluation of Eigenvalues and Eigenvectors of Real Symmetric Matrices by Simultaneous Iteration", Comp. J. 13, 76–80.

Rutishauser H. (1970). "Simultaneous Iteration Method for Symmetric Matrices", Numer. Math. 16, 205–23. See also HACLA, pp. 284–302.

Stewart G. W. (1969). "Accelerating The Orthogonal Iteration for the Eigenvalues of a Hermitian Matrix", Numer. Math. 13, 362–76.

Литература по специальным методам для симметрической проблемы обширна. Некоторые типичные статьи даны в следующем списке:

Bathe K. J. and Wilson E. L. (1973). "Solution Methods for Eigenvalue Problems in Structural Mechanics", Int. J. Numer. Meth. Eng. 6, 213–26.

Bender C. F. and Shavitt I. (1970). "An Iterative Procedure for the Calculation of the Lowest Real Eigenvalue and Eigenvector of a Non-Symmetric Matrix", J. Comp. Physics 6, 146–49.

Gupta K. K. (1972). "Solution of Eigenvalue Problems by Sturm Sequence Method", Int. J. Numer. Math. Eng. 4, 379–404.

Jenson P. S. (1972). "The Solution of Large Symmetric Eigenproblems by Sectioning", SIAM J. Num. Anal. 9, 534–45.

McCormick S. F. (1972). "A General Approach to One-Step Iterative Methods with Application to Eigenvalue Problems", J. Comput. Sys. Sci. 6, 354–72.

Parlett B. N. (1974b). "The Rayleigh Quotient Iteration and Some Generalizations for Nonnormal Matrices", Math. Comp. 28, 679–93.

Ruhe A. (1974). "SOR Methods for the Eigenvalue Problem with Large Sparse Matrices", Math. Comp. 28, 695–710.

- Sameh A., Lermit J. and Noh K. (1975). "On the Intermediate Eigenvalues of Symmetric Sparse Matrices", BIT 12, 543–54.
- Scott D. S. (1984). "Computing a Few Eigenvalues and Eigenvectors of a Symmetric Band Matrix", SIAM J. Sci. and Stat. Comp. 5, 658–666.
- Stewart G. W. (1974). "The Numerical Treatment of Large Eigenvalue Problems", Proc. IFIP Congress 74, North-Holland, pp. 666–72.
- Vandergraft J. (1971). "Generalized Rayleigh Methods with Applications to Finding Eigenvalues of Large Matrices", Lin. Alg. and Its Applic. 4, 353–68.
- Проблемы собственных значений с плотными матрицами специальной структуры также допускают специальные методы. Смотри
- Cybenko G. and Van Loan C. (1986). "Computing the Minimum Eigenvalue of a Symmetric Positive Definite Toeplitz Matrix", SIAM J. Sci. and Stat. Comp. 7, 123–131.

## 8.5. Методы Якоби

Методы Якоби для симметричной проблемы собственных значений в настоящее время привлекают к себе внимание, потому что они являются существенно параллельными. Во время их работы выполняется последовательность ортогонально подобных обновлений  $A \leftarrow Q^T A Q$ , обладающих тем свойством, что каждая новая матрица  $A$ , даже плотная, является «более диагональной», чем ее предшественник. В конце концов, внедиагональные элементы становятся достаточно малыми для того, чтобы объявить их равными нулю.

После обзора основных идей, лежащих в основе метода Якоби, мы разработаем параллельную процедуру Якоби, которая годится для кольцевой многопроцессорной системы. Все процедуры Якоби для симметричной проблемы собственных значений из этого раздела имеют SVD-аналоги.

### 8.5.1. Идея метода Якоби

Идея, лежащая в основе метода Якоби, состоит в том, чтобы систематически уменьшать величину

$$\text{off}(A) = \sqrt{\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}^2},$$

т. е. «норму» внедиагональных элементов. Средством для осуществления этого являются вращения вида

$$J(p, q, \theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}_{p \quad q},$$

которые мы будем называть *вращениями Якоби*. Вращения Якоби не отличаются от вращений Гивенса, ср. с § 5.1.8. Мы пошли в этом разделе на изменение названия в честь автора метода.

Основной шаг в процедуре Якоби включает (1) выбор пары индексов  $(p, q)$ , удовлетворяющих неравенству  $1 \leq p < q \leq n$ , (2) вычисление пары косинус-синус  $(c, s)$  такой, что матрица

$$\begin{bmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \quad (8.5.1)$$

диагональная, и (3) запись матрицы  $B = J^T AJ$  в матрицу  $A$ , где  $J = J(p, q, \theta)$ . Заметим, что матрица  $B$  совпадает с матрицей  $A$ , кроме строк и столбцов с номерами  $p$  и  $q$ . Более того, так как норма Фробениуса сохраняется при ортогональных преобразованиях, мы находим, что

$$a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2 = b_{pp}^2 + b_{qq}^2 + 2b_{pq}^2 = b_{pp}^2 + b_{qq}^2,$$

и, следовательно,

$$\begin{aligned} \text{off}(B)^2 &= \|B\|_F^2 - \sum_{i=1}^n b_{ii}^2 \\ &= \|A\|_F^2 - \sum_{i=1}^n a_{ii}^2 + (a_{pp}^2 + a_{qq}^2 - b_{pp}^2 - b_{qq}^2) \\ &= \text{off}(A)^2 - 2a_{pq}^2. \end{aligned}$$

В этом смысле матрица  $A$  становится ближе к диагональной форме на каждом шаге метода Якоби.

Прежде чем обсуждать, как может быть выбрана пара индексов  $(p, q)$ , давайте рассмотрим фактические вычисления, связанные с  $(p, q)$ -подзадачей.

### 8.5.2. Симметричное разложение Шура размера $2 \times 2$

Сказать, что мы диагонализовали матрицу (8.5.1), это сказать, что

$$0 = b_{pq} = a_{pq}(c^2 - s^2) + (a_{pp} - a_{qq})cs. \quad (8.5.2)$$

Если  $a_{pq} = 0$ , то мы сразу берем  $(c, s) = (1, 0)$ . Иначе, полагаем

$$\tau = \frac{a_{qq} - a_{pp}}{2a_{pq}} \text{ и } t = s/c$$

и из равенства (8.5.2) делаем вывод, что  $t = \operatorname{tg}(\theta)$  является решением квадратного уравнения

$$t^2 + 2\tau t - 1 = 0.$$

Оказывается, важно выбрать наименьший из двух его корней

$$t = \frac{\operatorname{sign}(\tau)}{|\tau| + \sqrt{1 + \tau^2}}, \quad (8.5.3)$$

после чего величины  $c$  и  $s$  могут быть найдены по формулам

$$c = 1/\sqrt{1 + t^2}, \quad s = tc. \quad (8.5.4)$$

Выбор  $t$  равным наименьшему из двух корней гарантирует, что  $|\theta| \leq \pi/4$  и имеет место эффект минимизации разности матриц  $B$  и  $A$ , так как

$$\|B - A\|_F^2 = 4(1 - c) \sum_{\substack{i=1 \\ i \neq p, q}}^n (a_{ip}^2 + a_{iq}^2) + 2a_{pq}^2/c^2.$$

Мы подведем итог  $2 \times 2$  вычислениям следующим образом:

**Алгоритм 8.5.1.** Для заданной симметричной матрицы  $A$  размера  $n \times n$  и целых  $p$  и  $q$ , удовлетворяющих неравенствам  $1 \leq p < q \leq n$ , этот алгоритм вычисляет пару косинус – синус  $(c, s)$ , такую, что если  $B = J(p, q, \theta)^T AJ(p, q, \theta)$ , то  $b_{pq} = b_{qp} = 0$ .

```
function: (c, s) = sym.schur2(A, p, q)
    if A(p, q) ≠ 0
        τ = (A(p, q) - A(p, p))/(2A(p, q))
        t = sign(τ)/(|τ| + √(1 + τ²)); c = 1/√(1 + t²);
    else
        c = 1; s = 0
    end
end sym.schur2
```

### 8.5.3. Обновления в методе Якоби

Как было замечено выше, когда решают  $(p, q)$ -подзадачу, меняются только строки и столбцы с номерами  $p$  и  $q$ . После того, как процедура `sym.schur2` определила вращение размера  $2 \times 2$ , обновление  $A \leftarrow J(p, q, \theta)^T AJ(p, q, \theta)$  может быть выполнено за  $6n$  флопов, если использовать симметрию.

### 8.5.4. Классический алгоритм Якоби

Теперь мы обратим внимание на определение подзадач, т. е. выбор пары  $(p, q)$ . С точки зрения максимизации изменения величины  $\text{off}(A)$  в (8.5.2), имеет смысл выбирать пару  $(p, q)$  так, чтобы величина  $a_{pq}^2$  была максимальной. Это составляет основу классического алгоритма Якоби.

**Алгоритм 8.5.2 (Классический алгоритм Якоби).** Для заданной симметричной матрицы  $A \in \mathbb{R}^{n \times n}$  и погрешности  $tol > 0$  этот алгоритм записывает в матрицу  $A$  матрицу  $V^T AV$ , где матрица  $V$  ортогональная и  $\text{off}(V^T AV) \leq tol \|A\|_F$ .

```
V = I_n; eps = tol \|A\|_F
while off(A) > eps
    Выбрать индексы p и q, такие, что
    1 ≤ p < q ≤ n и |a_{pq}| = max_{i ≠ j} |a_{ij}|
    (c, s) = sym.schur2(A, p, q)
    A = J(p, q, θ)^T AJ(p, q, θ)
    V = VJ(p, q, θ)
end
```

Из того что  $|a_{pq}|$  является максимальным внедиагональным элементом, следует, что

$$\text{off}(A)^2 \leq N(a_{pq}^2 + a_{qp}^2),$$

где  $N = n(n - 1)/2$ . Поэтому из (8.5.1) мы получаем оценку

$$\text{off}(B)^2 \leq \left(1 - \frac{1}{N}\right) \text{off}(A)^2.$$

По индукции, если  $A^{(k)}$  означает матрицу  $A$  после  $k$  обновлений Якоби, то

$$\text{off}(A^{(k)})^2 \leq \left(1 - \frac{1}{N}\right)^k \text{off}(A^{(0)})^2.$$

Это означает, что классическая процедура Якоби имеет линейную скорость сходимости.

Однако асимптотическая скорость сходимости этого метода значительно выше линейной. Шонхаге [Schonhage, 1964] и Ван Кемпен [van Kempen, 1966] показали, что для достаточно большого  $k$  существует константа  $c$ , такая, что

$$\text{off}(A^{(k+N)}) \leq c \cdot \text{off}(A^{(k)})^2,$$

т. е. имеет место квадратичная сходимость. В более ранней работе Хенричи [Henrici, 1958] установлен тот же результат для специального случая, когда матрица  $A$  имеет различные собственные значения. В теории сходимости итераций Якоби существенно, что  $|\theta| \leq \pi/4$ . Помимо всего прочего, это предотвращает возможность «обмена» между близко сошедшимися диагональными элементами. Это следует из формул  $b_{pp} = a_{pp} - ta_{pq}$  и  $b_{qq} = a_{qq} + ta_{pq}$ , которые могут быть выведены из уравнений (8.5.1) и (8.5.3).

Принято называть  $N$  обновлений Якоби *чисткой*. Таким образом, после достаточного числа итераций квадратичная сходимость наблюдается, если проверять  $\text{off}(A)$  после каждой чистки.

Не существует строгой теории, дающей возможность предсказать число чисток, требующихся для достижения определенного уменьшения  $\text{off}(A)$ . Однако Брент и Люк [Brent, Luk, 1985] обосновали эвристически, что число чисток пропорционально  $\log(n)$ , и это, по-видимому, согласуется с практикой.

**Пример 8.5.1.** Применяя классическую итерацию Якоби к матрице

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix},$$

мы найдем, что

Чистка	$O(\text{off}(A))$
0	$10^2$
1	$10^1$
2	$10^{-2}$
3	$10^{-11}$
4	$10^{-17}$

### 8.5.5. Алгоритм циклический по строкам

Недостаток классического метода Якоби в том, что обновления требуют  $O(n)$  флопов, в то время как поиск оптимальной пары  $(p, q)$  требует  $O(n^2)$  флопов. Один из путей устранить этот дисбаланс – заранее фиксировать последовательность подзадач, которые будут решаться. Представляется разумным проходить все подзадачи строки за строкой. Например, при  $n = 4$  цикл имеет следующий вид

$$(p, q) = (1,2), (1,3), (1,4), (2,3), (2,4), (3,4), (1,2), \dots$$

Этот способ упорядочения называют циклом по строкам и его реализует следующая процедура:

**Алгоритм 8.5.3 (Циклический метод Якоби).** Для заданной симметричной матрицы  $A \in \mathbb{R}^{n \times n}$  и погрешности  $tol > 0$ , этот алгоритм записывает в матрицу  $A$  матрицу  $V^T A V$ , где  $V$  – ортогональная матрица и  $\text{off}(V^T A V) \leq tol \|A\|_F$ .

```

 $V = I_n$ 
 $eps = tol \| A \|_F$ 
while off( $A$ ) > eps
    for  $p = 1:n - 1$ 
        for  $q = p + 1:n$ 
            ( $c, s$ ) = sym.schur2( $A, p, q$ )
             $A = J(p, q, \theta)^T AJ(p, q, \theta)$ 
             $V = VJ(p, q, \theta)$ 
        end
    end
end

```

Циклический метод Якоби также сходится квадратично. (См. [Wilkinson, 1962] и [Van Kempen, 1966]). Однако поскольку он не требует внедиагонального поиска, он значительно быстрее, чем исходный алгоритм Якоби.

**Пример 8.5.2.** Если циклический метод Якоби применить к матрице из примера 8.5.1, то мы найдем, что

Чистка	$O(\text{off}(A))$
0	$10^2$
1	$10^x$
2	$10^{-1}$
3	$10^{-6}$
4	$10^{-16}$

### 8.5.6. Барьерный метод Якоби

Когда выполняется циклический метод Якоби, представляется разумным пропускать уничтожение элемента  $a_{pq}$ , если его модуль меньше, чем некоторый малый параметр, зависящий от чистки, так как фактическое уменьшение величины  $\text{off}(A)$  не будет оправдывать затрат. Это приводит к так называемому барьерному методу Якоби. Детали, связанные с этим вариантом алгоритма Якоби, можно найти в [Wilkinson, AEP], с. 277. Подходящий выбор барьеров может гарантировать сходимость.

### 8.5.7. Анализ ошибок округления

Используя предложенный Уилкинсоном анализ ошибок, можно показать, что если нужно выполнить  $r$  чисток по алгоритму 8.5.3, то вычисленная величина  $d_i$  удовлетворяет неравенству

$$\sum_{i=1}^n (d_i - \lambda_i)^2 \leq (\delta + k_r) \| A \|_F u$$

при некотором упорядочении собственных значений  $\lambda_i$  матрицы  $A$ . Параметр  $k_r$  зависит от  $r$  слабо.

### 8.5.8. Сравнение с симметричным QR-алгоритмом

Хотя циклический метод Якоби сходится квадратично, он, вообще говоря, не выдерживает конкуренции с симметричным QR-алгоритмом. Например, если подсчитывать флопы, то 2 чистки метода Якоби, примерно эквивалентны полной QR-редукции к диагональной форме с накоплением преобразований. Следователь-

но, алгоритм Якоби, вообще говоря, хуже. Однако, его программа очень простая по сравнению с QR-алгоритмом и, следовательно, при малых  $n$  он может быть предпочтительнее. Кроме того, если известна приближенная матрица собственных векторов  $V$ , то матрица  $V^T A V$  почти диагональная, — пример ситуации, которую может использовать метод Якоби, но не QR.

### 8.5.9. Параллельное упорядочение

Возможно наиболее интересное различие между алгоритмами QR и Якоби для симметрической проблемы собственных значений — богатый внутренний параллелизм последнего алгоритма. Для того чтобы пояснить это, положим  $n = 4$  и сгруппируем шесть подзадач в три множества вращений следующим образом

$$\begin{aligned} \text{rot. set}(1) &= \{(1, 2), (3, 4)\}, \\ \text{rot. set}(2) &= \{(1, 3), (2, 4)\}, \\ \text{rot. set}(3) &= \{(1, 4), (2, 3)\}. \end{aligned}$$

Заметим, что все вращения внутри каждого из трех множеств вращений являются «неконфликтующими». То есть подзадачи  $(1, 2)$  и  $(3, 4)$  могут быть выполнены параллельно. Подзадачи  $(1, 3)$  и  $(2, 4)$  также могут быть выполнены параллельно равно как и подзадачи  $(1, 4)$  и  $(2, 3)$ . В общем случае мы будем говорить, что

$$(i_1, j_1), (i_2, j_2), \dots, (i_N, j_N), N = (n - 1)n/2$$

есть параллельное упорядочение множества пар индексов  $\{(i, j) | 1 \leq i < j \leq n\}$ , если при  $s = 1 : n - 1$  множества вращений

$$\text{rot.set}(s) = \{(i_r, j_r) : r = 1 + n(s - 1)/2 : ns/2\}$$

состоят из неконфликтующих вращений. Для этого необходимо, чтобы  $n$  было четным, что мы и будем предполагать в этом разделе. (Случай нечетного  $n$  можно охватить, окаймляя матрицу  $A$  строкой и столбцом нулей и проявляя осторожность при решении подзадач, включающих эти добавочные нули.)

Хороший способ генерировать параллельное упорядочение — воспроизвести шахматный турнир с  $n$  игроками, в котором каждый должен играть с каждым ровно один раз. В случае  $n = 8$  это повлечет за собой 7 «кругов». В первом круге мы имеем следующие четыре игры

1	3	5	7
2	4	6	8

,

т. е. 1 играет с 2, 3 играет с 4 и т. д. Это соответствует множеству

$$\text{rot.set}(1) = \{(1, 2), (3, 4), (5, 6), (7, 8)\}.$$

Пройдя круги 2–7, игрок 1 остановится, а игроки 2–8 начнут новый тур:

1	2	3	5
4	6	8	7

$$\text{rot.set}(2) = \{(1, 4), (2, 6), (3, 8), (5, 7)\}$$

1	4	2	3
6	8	7	5

$$\text{rot.set}(3) = \{(1, 6), (4, 8), (2, 7), (3, 5)\}$$

1	6	4	2
8	7	5	3

$$\text{rot.set}(4) = \{(1, 8), (6, 7), (4, 5), (2, 3)\}$$

1	8	6	4
7	5	3	2

$$\text{rot.set}(5) = \{(1, 7), (5, 8), (3, 6), (2, 4)\}$$

1	7	8	6
5	3	2	4

$$\text{rot.set}(6) = \{(1, 5), (3, 7), (2, 8), (4, 6)\}$$

1	5	7	8
3	2	4	6

$$\text{rot.set}(7) = \{(1, 3), (2, 5), (4, 7), (6, 8)\}$$

Мы можем закодировать эти действия в паре целочисленных векторов  $\text{top}(1:n/2)$  и  $\text{bot}(1:n/2)$ . В течение заданного круга  $\text{top}(k)$  играет с  $\text{bot}(k)$ ,  $k = 1:n/2$ . Разбиение на пары для следующего круга достигается обновлением векторов  $\text{top}$  и  $\text{bot}$ :

```
function: [top, bot] = music(top, bot)
    m = length(top)
    for k = 1:m
        if k == 2
            new.top(k) = bot(1)
        elseif k > 2
            new.top(k) = top(k - 1)
        end
        if k == m
            new.bot(k) = top(k)
        else
            new.bot(k) = bot(k + 1)
        end
    end
    top = new.top; bot = new.bot
end music
```

Используя процедуру **music**, мы получаем следующую параллельную процедуру Якоби.

**Алгоритм 8.5.4 (Параллельный метод Якоби).** Для заданной симметричной матрицы  $A \in \mathbb{R}^{n \times n}$  и погрешности  $tol > 0$  этот алгоритм записывает в матрицу  $A$  матрицу  $V^T A V$ , где  $V$  – ортогональная матрица и  $\text{off}(V^T A V) \leq tol \|A\|_F$ . Предполагается, что  $n$  четное.

```
V = I_n
eps = tol * \|A\|_F
top = 1:2:n; bot = 2:2:n
while off(A) > eps
    for set = 1:n-1
        for k = 1:n/2
            p = min(top(k), bot(k))
            q = max(top(k), bot(k))
            (c, s) = sym.schur2(A, p, q)
            A = J(p, q, theta)^T A J(p, q, theta)
            V = VJ(p, q, theta)
        end
        [top, bot] = music(top, bot)
    end
end
```

Заметим, что цикл по  $k$  проходит через  $n/2$  независимых, неконфликтующих подзадач.

### 8.5.10. Кольцевая процедура

Теперь мы покажем, как выполнять алгоритм 8.4.4 на кольцевой  $p$ -процессорной вычислительной системе. Предположим на некоторое время, что  $p = n/2$ . В любой момент процессор Proc( $\mu$ ) располагает двумя столбцами матрицы  $A$  и соответствующими столбцами матрицы  $V$ . Например, если  $n = 8$ , то столбцы матрицы  $A$  распределяют от шага к шагу как показано ниже:

	Proc(1)	Proc(2)	Proc(3)	Proc(4)
Step 1:	[1 2]	[3 4]	[5 6]	[7 8]
Step 2:	[1 4]	[2 6]	[3 8]	[5 7]
Step 3:	[1 6]	[4 8]	[2 7]	[3 5]

и т. д.

Упорядоченные пары обозначают индексы расположенных в процессорах столбцов. Первый индекс означает *левый* столбец, а второй индекс – *правый* столбец. Таким образом, *левый* и *правый* столбцы в Proc(3) в течение шага 3 это 2 и 7, соответственно. Эта терминология используется ниже в параллельной процедуре.

Заметим, что между шагами столбцы сдвигают в соответствии с перестановкой, определяемой процедурой **music**, и что связи с ближайшими соседями сохраняются. На каждом шаге каждый процессор имеет дело с одной подзадачей. Она включает (a) вычисление ортогональной матрицы  $V_{small} \in \mathbb{R}^{2 \times 2}$ , решающей локальную проблему Шура размера  $2 \times 2$ , (b) использование матрицы  $2 \times 2 V_{small}$  для обновления двух столбцов матриц  $A$  и  $V$ , размещенных в этом процессоре, (c) пересылку  $2 \times 2 V_{small}$  во все другие процессоры, и (d) получение матриц  $V_{small}$  от других процессоров и локальное обновление матриц  $A$  и  $V$  соответственно. Поскольку матрица  $A$  хранится по столбцам, связь между процессорами необходима для того чтобы выполнять обновления матриц  $V_{small}$ , так как они зависят от столбцов матрицы  $A$ . Например, на втором шаге при  $n = 8$  Proc(2) должен получить матрицы вращения  $2 \times 2$ , связанные с подзадачами (1, 4), (3, 8) и (5, 7). Они поступают из Proc(1), Proc(3) и Proc(4) соответственно. В общем случае, распределение матриц вращения можно удобно осуществлять посредством циркуляции матриц  $2 \times 2 V_{small}$  в «карусельном» режиме по кольцу процессоров. Каждый процессор копирует проходящую через него матрицу  $2 \times 2 V_{small}$  в свою локальную память и затем обновляет размещенные в нем части матриц  $A$  и  $V$ .

Критерий остановки алгоритма 8.5.4 вызывает определенные проблемы в системе с распределенной памятью, состоящие в том, что оценка величин  $off(\cdot)$  и  $\|A\|_F$  требует доступа ко всей матрице  $A$ . Однако эти глобальные величины могут быть вычислены в течение одного цикла карусели. До того как начнется вращение, каждый процессор может вычислить свой вклад в  $\|A\|_F$  и  $off(\cdot)$ . Эти величины могут быть затем суммированы каждым процессором, если они расположены на карусели и считаются во время каждой остановки. После одного оборота каждый процессор будет иметь собственную копию величин  $\|A\|_F$  и  $off(\cdot)$ . Однако для простоты в нижеследующем алгоритме мы просто останавливаемся после определенного числа чисток  $M$ .

**Алгоритм 8.5.5.** Пусть  $A$  – симметричная матрица размера  $n \times n$ . Если каждый процессор  $p$ -процессорного кольца выполняет следующий набор программ, то по завершении процесса Proc( $\mu$ ) разместит  $B(:, 2\mu - 1 : 2\mu)$  в  $A_{loc}$  и  $V(:, 2\mu - 1 : 2\mu)$  в  $V_{loc}$ ,

где  $B = V^T A V$  – матрица, которая получится в результате  $M$  чисток по алгоритму 8.5.4. Предполагается, что  $p \geq 2$ .

```

local.  

init [ $p, \mu = my.id, left, right, n, A_{loc} = A(:, 2\mu - 1 : 2\mu), M]$   

 $V_{loc} = I_n(:, 2\mu - 1 : 2\mu)$   

 $top = 1 : 2 : n; bot = 2 : 2 : n$   

{Установить партнеров коммутационной сети для перетасовки столбцов  

    Lsend, Rsend = куда левый (правый) столбец передается  

    Lrecv, Rrecv = откуда новый левый (правый) столбец берется.}  

if  $\mu = 1$   

    Lsend = 1; Rsend = 2, Lrecv = 1; Rrecv = 2  

elseif  $\mu = p$   

    Lsend =  $p$ ; Rsend =  $p - 1$ ; Lrecv =  $p - 1$ ; Rrecv =  $p$   

else  

    Lsend =  $\mu + 1$ ; Rsend =  $\mu - 1$ ; Lrecv =  $\mu - 1$ ; Rrecv =  $\mu + 1$   

end  

for sweep = 1 :  $M$   

    for set = 1 :  $n - 1$   

         $p = \min(top(\mu), bot(\mu)); q = \max(top(\mu), bot(\mu))$   

        ( $c, s$ ) = sym.schur 2( $A_{loc}([p, q], 1 : 2), 1, 2$ );  $V_{small} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$   

         $A_{loc} = A_{loc} V_{small}; V_{loc} = V_{loc} V_{small}$   

        {Сдвинуть матрицы  $2 \times 2 V_{small}$  по кольцу.}  

         $\lambda = \mu$   

        for  $k = 1 : p$   

            send( $V_{small}, right$ ); recv( $V_{small}, left$ )  

            if  $\lambda = 1$   

                 $\lambda = p$   

            else  

                 $\lambda = \lambda - 1$   

            end  

            {Принять матрицу  $V_{small}$  от Proc( $\lambda$ )}  

             $p_0 = \min(top(\lambda), bot(\lambda)); q_0 = \max(top(\lambda), bot(\lambda))$   

             $A_{loc}([p_0, q_0], :) = V_{small}^T A_{loc}([p_0, q_0], :)$   

end  

{Тасовка столбцов для подготовки следующего множества вращений.}  

send( $A_{loc}(:, 1), Lsend$ ); send( $A_{loc}(:, 2), Rsend$ )  

recv( $A_{loc}(:, 1), Lrecv$ ); recv( $A_{loc}(:, 2), Rrecv$ )  

send( $V_{loc}(:, 1), Lsend$ ); send( $V_{loc}(:, 2), Rsend$ )  

recv( $V_{loc}(:, 1), Lrecv$ ); recv( $V_{loc}(:, 2), Rrecv$ )  

 $[top, bot] = music(top, bot)$   

end  

end
```

Порядок команд **send** и **recv**, связанных с тасовкой столбцов существен. Читатель, должно быть, заметил почему: левый и правый столбцы оба должны быть переданы после требуемой перестановки.

### 8.5.11. Блочная процедура Якоби

Обычно когда решают симметричную проблему собственных значений на  $p$ -процессорной системе,  $n \gg p$ . В этом случае может быть использована блочная

версия алгоритма Якоби. Блочная версия описанной выше процедуры строится непосредственно. Предположим, что  $n = rN$  и что мы разбили матрицу  $A$  размера  $n \times n$  на блоки следующим образом:

$$A = \begin{bmatrix} A_{11} & \dots & A_{1N} \\ \vdots & & \vdots \\ A_{N1} & \dots & A_{NN} \end{bmatrix}.$$

Здесь каждый блок  $A_{ij}$  — матрица размера  $r \times r$ . В блочном методе Якоби  $(p, q)$  подзадачи включают в себя вычисление разложения Шура размера  $2 \times 2$ :

$$\begin{bmatrix} V_{pp} & V_{pq} \\ V_{qp} & V_{qq} \end{bmatrix}^T \begin{bmatrix} A_{pp} & A_{pq} \\ A_{qp} & A_{qq} \end{bmatrix} \begin{bmatrix} V_{pp} & V_{pq} \\ V_{qp} & V_{qq} \end{bmatrix} = \begin{bmatrix} D_{pp} & 0 \\ 0 & D_{qq} \end{bmatrix}$$

и последующее применение к матрице  $A$  блочного вращения Якоби, осуществляющего совместно матрицами  $V_{ij}$ . Обозначим это блочное вращение через  $V$ . Легко показать, что

$$\text{off}(V^T AV)^2 = \text{off}(A)^2 - (2 \| A_{pq} \|_F^2 + \text{off}(A_{pp})^2 + \text{off}(A_{qq})^2).$$

Блочные процедуры Якоби имеют много интересных вычислительных аспектов. Например, существует много способов решать подзадачи и выбор оказывается существенен. См. [Bischof, 1987].

### 8.5.12. SVD-процедура Якоби

Наконец, мы заметим, что для каждой приведенной выше процедуры Якоби для симметрической проблемы собственных значений существует соответствующий аналогичный алгоритм SVD. Подзадачи в этом случае состоят в  $2 \times 2$  SVD-вычислениях, т. е.

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} d_p & 0 \\ 0 & d_q \end{bmatrix}.$$

Детали см. в задачах 8.5.1 и 8.5.2.

### Задачи

**8.5.1. (a)** Пусть матрица

$$C = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$$

вещественная. Предложить устойчивый алгоритм вычисления величин  $c$  и  $s$ , удовлетворяющих равенству  $c^2 + s^2 = 1$ , таких, что матрица

$$B = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} C$$

симметричная. **(b)** Комбинируя (a) с тригонометрическими вычислениями метода Якоби, описанными выше, получить устойчивый алгоритм вычисления SVD матрицы  $C$ . **(c)** Часть (b) можно использовать для разработки Якоби-подобного алгоритма для вычисления SVD матрицы  $A \in \mathbb{R}^{n \times n}$ . Для заданной пары  $(p, q)$ , где  $p < q$ , преобразования Якоби  $J(p, q, \theta_1)$  и  $J(p, q, \theta_2)$  определяют так, что если

$$B = J(p, q, \theta_1)^T A J(p, q, \theta_2),$$

то  $b_{pq} = b_{qp} = 0$ . Показать, что

$$\text{off}(B)^2 = \text{off}(A)^2 - b_{pq}^2 - b_{qp}^2.$$

Как лучше выбирать  $p$  и  $q$ ? Как можно модифицировать этот алгоритм для того, чтобы охватить случай, когда  $A \in \mathbb{R}^{m \times n}$  с  $m > n$ ?

**8.5.2.** Предположим, что векторы  $x$  и  $y$  принадлежат  $\mathbb{R}^n$ , и определим матрицу  $Q$  следующим образом:

$$Q = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}.$$

Предложить устойчивый алгоритм вычисления величин  $c$  и  $s$ , таких, что столбцы матрицы  $[x, y] Q$  взаимно ортогональны. Основная идея «односторонних» Якоби-подобных SVD-алгоритмов состоит в том, чтобы найти ортогональную матрицу  $V$ , такую, что столбцы матрицы  $B = AV$  – взаимно ортогональные. Матрица  $U$  в SVD-разложении может быть затем найдена путем нормализации столбцов.

**8.5.3.** Пусть задана скалярная величина  $\gamma$  вместе с матрицей

$$A = \begin{bmatrix} w & x \\ x & z \end{bmatrix}.$$

Необходимо вычислить ортогональную матрицу

$$F = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

такую, что элемент  $(1, 1)$  матрицы  $F^T AF$  равен  $\gamma$ . Показать, что эти требования приводят к уравнению

$$(w - \gamma)\tau^2 - 2x\tau + (z - \gamma) = 0,$$

где  $\tau = c/s$ . Проверить, что это квадратное уравнение имеет вещественные корни, если величина  $\gamma$  удовлетворяет неравенству  $\lambda_2 \leq \gamma \leq \lambda_1$ , где  $\lambda_1$  и  $\lambda_2$  – собственные значения матрицы  $A$ .

**8.5.4.** Пусть  $A \in \mathbb{R}^{n \times n}$  – симметричная матрица. Используя задачу 8.5.3, предложить алгоритм, который вычисляет разложение

$$Q^T AQ = \gamma I + F,$$

где  $Q$  – произведение вращений Якоби,  $\gamma = \text{trace}(A)/n$  и матрица  $F$  имеет нулевые диагональные элементы. Рассмотреть вопрос о единственности матрицы  $Q$ .

**8.5.5.** Сформулировать процедуру Якоби для (а) кососимметричной матрицы и (б) комплексной эрмитовой матрицы.

**8.5.6.** Разбить вещественную симметричную матрицу  $A$  размера  $n \times n$  на блоки следующим образом

$$A = \begin{bmatrix} a & v^T \\ v & A_1 \\ 1 & n-1 \end{bmatrix} \begin{matrix} 1 \\ \\ \end{matrix}.$$

Пусть  $Q$  означает матрицу Хаусхолдера, такую, что если  $B = Q^T AQ$ , то  $B(3:n, 1) = 0$ . Пусть матрица вращения  $J = J(1, 2, 0)$  определена так, что если  $C = J^T BJ$ , то  $c_{12} = 0$  и  $c_{11} \geq c_{22}$ . Показать, что  $c_{11} \geq a + \|v\|_2$ . Л. Будде [La Budde, 1964] сформулировал алгоритм для симметричной проблемы собственных значений, основанный на повторении этих хаусхолдер-якобиевых вычислений.

**8.5.7.** Организовать процедуру `music` так, чтобы она требовала как можно меньше рабочей памяти.

**8.5.8.** Модифицировать алгоритм 8.5.5 так, чтобы он останавливался, когда  $\text{off}(A) \leq tol ||A||_F$ .

**8.5.9.** Модифицировать алгоритм 8.5.5 так, чтобы он вычислял SVD.

### Замечания и литература к § 8.5

Основополагающая работа Якоби является одной из самых ранних работ, обнаруженных в литературе по численному анализу

Jacobi C. G. J. (1846). "Über ein Leichtes Verfahren Die in der Theorie der Sacularstrougen Vorkommenden Gleichungen Numerisch Aufzulösen", Crelle's J. 30, 51–94.

Предшествующий QR-алгоритму метод Якоби был стандартным методом решения плотной симметрической проблемы собственных значений. Ранние попытки улучшить его включают

La Budde C. D. (1964). "Two Classes of Algorithms for Finding the Eigenvalues and Eigenvectors of Real Symmetric Matrices", J. ACM 11, 53–58.

Lotkin M. (1956). "Characteristic Values of Arbitrary Matrices", Quart. Appl. Math. 14, 267–75.

Pope D. A. and Tompkins C. (1957). "Maximizing Functions of Rotations: Experiments Concerning Speed of Diagonalization of Symmetric Matrices Using Jacobi's Method", J. ACM 4, 459–66.

Вычислительные аспекты метода Якоби описаны в [Wilkinson, AEP], с. 265, а алгольную процедуру можно найти в

Rutishauser H. (1966). "The Jacobi Method for Real Symmetric Matrices", Numer. Math. 9, 1–10. See also HACLA, pp. 202–11.

Методы Якоби также представляют интерес при использовании миникомпьютеров, где компактность делает их привлекательными. Смотри

Nash J. C. (1975). "A One-Sided Transformation Method for the Singular Value Decomposition and Algebraic Eigenproblem", Comp. J. 18, 74–76.

Они также полезны, когда нужно диагонализовать матрицу, близкую к диагональной. Смотри

Wilkinson J. H. (1968). "Almost Diagonal Matrices with Multiple or Close Eigenvalues", Lin. Alg. and Its Applic. 1, 1–12.

Установление квадратичной сходимости классической и циклической итераций Якоби увлекло нескольких авторов

Brodie K. W. and Powell M. J. D. (1975). "On the Convergence of Cyclic Jacobi Methods", J. Inst. Math. Applic. 15, 279–87.

Hansen E. R. (1962). "On Quasicyclic Jacobi Methods", ACM J. 9, 118–35.

Hansen E. R. (1963). "On Cyclic Jacobi Methods", SIAM J. Appl. Math. 11, 448–59.

Henrici P. (1958). "On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing the Eigenvalues of Hermitian Matrices", SIAM J. Appl. Math. 6, 144–62.

Henrici P. and Zimmermann K. (1968). "An Estimate for the Norms of Certain Cyclic Jacobi Operators", Lin. Alg. and Its Applic. 1, 489–501.

Schonhage A. (1964). "On the Quadratic Convergence of the Jacobi Process", Numer. Math. 6, 410–12.

van Kempen H. P. M. (1966). "On Quadratic Convergence of the Special Cyclic Jacobi Method", Numer. Math. 9, 19–22.

Wilkinson J. H. (1962). "Note on the Quadratic Convergence of the Cyclic Jacobi Process", Numer. Math. 6, 296–300.

Делались попытки распространить итерации Якоби на другие классы матриц и добиться соответствующих результатов в сходимости. Случай нормальных матриц рассматривается в

Goldstine H. H. and Horowitz L. P. (1959). "A Procedure for the Diagonalization of Normal Matrices", J. Assoc. Comp. Mach. 6, 176–95.

Loizou G. (1972). "On the Quadratic Convergence of the Jacobi Method for Normal Matrices", Comp. J. 15, 274–76.

Ruhe A. (1972). "On the Quadratic Convergence of the Jacobi Method for Normal Matrices", BIT 7, 305–13.

Смотри также

Paarderkooper M. H. C. (1971). "An Eigenvalue Algorithm for Skew Symmetric Matrices", Numer. Math. 17, 189–202.

Существенно, что анализ и алгоритмические разработки, приведенные в этой работе, охватывают случай нормальных матриц без значительной модификации. Это замечание, вообще говоря, относится и к Якоби-подобным методам для SVD. Смотри

Bai Z. (1988). "Note on the Quadratic Convergence of Kogbetliantz's Algorithm for Computing the Singular Value Decomposition", Lin. Alg. and Its Applic. 104, 131–140.

Charlier J. P., Vanbegin M., Van Dooren P. (1988). "On Efficient Implementation of Kogbetliantz's Algorithm for Computing the Singular Value Decomposition", Numer. Math. 52, 279–300.

Charlier J. P. and Van Dooren P. (1987). "On Kogbetliantz's SVD Algorithm in the Presence of Clusters", Lin. Alg. and Its Applic. 95, 135–160.

Forsythe G. E. and Henrici P. (1960). "The Cyclic Jacobi Method for Computing the Principal Values of a Complex Matrix", Trans. Amer. Math. Soc. 94, 1–23.

Hansen P. C. (1988). "Reducing the Number of Sweeps in Hestenes Method", in Singular Value Decomposition and Signal Processing, ed. E. F. Deprettere, North Holland.

Kogbetliantz E. G. (1955). "Solution of linear Equations by Diagonalization of Coefficient Matrix", Quart. Appl. Math. 13, 123–132.

Paige C. C. (1986). "On the Quadratic Convergence of Kogbetliantz's Algorithm for Computing the Singular Value Decomposition", Lin. Alg. and Its Applic. 77, 301–314.

Van Dooren P. and Paige C. C. (1986). "On the Quadratic Convergence of Kogbetliantz's Algorithm for Computing the Singular Value Decomposition", Lin. Alg. and Its Applic. 77, 301–313.

Алгоритм Когбетлиантца – это в основном метод Якоби для SVD, набросок которого дан в § 8.5.12. Для матриц, не являющихся нормальными, ситуация значительно более сложная. См.

Boothroyd J. and Eberlein P. J. (1969). "Solution to the Eigenproblem by a Norm-Reducing Jacobi-Type Method (Handbook)", Numer. Math. 11, 1–12. See also HACLA, pp. 327–38.

Eberlein P. J. (1970). "Solution to the Complex Eigenproblem by a Norm-Reducing Jacobi-Type Method", Numer. Math. 14, 232–45. See also HACLA, pp. 404–17.

Froberg C. E. (1965). "On Triangularization of Complex Matrices by Two Dimensional Unitary Transformations", BIT 5, 230–34.

Greenstadt J. (1955). "A Method for Finding Roots of Arbitrary Matrices", Math. Tables and Other Aids to Comp. 9, 47–52.

Hari V. (1982). "On the Global Convergence of the Eberlein Method for Real Matrices", Numer. Math. 39, 361–370.

Huang C. P. (1975). "A Jacobi-Type Method for Triangularizing an Arbitrary Matrix", SIAM J. Num. Anal. 12, 566–70.

Ruhe A. (1968). On the Quadratic Convergence of a Generalization of the Jacobi Method to Arbitrary Matrices", BIT 8, 210–31.

Ruhe A. (1969). "The Norm of a Matrix After a Similarity Transformation", BIT 9, 53–58.

Stewart G. W. (1985). "A Jacobi-like Algorithm for Computing the Schur Decomposition of a Nonhermitian Matrix", SIAM J. Sci. and Stat. Comp. 6, 853–862.

Метод Якоби для комплексных симметрических матриц также может быть разработан. См.

Anderson P. and Loizou G. (1973). "On the Quadratic Convergence of an Algorithm Which Diagonalizes a Complex Symmetric Matrix", J. Inst. Applic. 12, 261–71.

Anderson P. and Loizou G. (1976). "A Jacobi-Type Method for Complex Symmetric Matrices (Handbook)", Numer. Math. 25, 347–63.

Eberlein P. J. (1971). "On the Diagonalization of Complex Symmetric Matrices", J. Inst. Math. Applic. 7, 377–83.

Seaton J. J. (1969). "Diagonalization of Complex Symmetric Matrices Using a Modified Jacobi Method", Comp. J. 12, 156–57.

Хотя симметричный OR-алгоритм, вообще говоря, значительно быстрее, чем метод Якоби, существуют особые ситуации, в которых последний метод представляет интерес. Как мы показали, на параллельных компьютерах можно выполнять несколько вращений совместно, посредством этого ускоряя уменьшение внедиагональных элементов. Смотри

Berry M. and Sameg A. (1986). "Multiprocessor Jacobi Algorithms for Dense Symmetric Eigenvalue

- and Singular Value Decompositions”, in Proc. International Conference on Parallel Processing, 433–440.
- Bischof C. H. (1987). “The Two-Sided Block Jacobi Method on Hypercube Architectures”, in Hypercube Multiprocessors, ed. M. T. Heath; SIAM Press, Philadelphia.
- Bischof C. H. (1988). “Computing the Singular Value Decomposition on a Distributed System of Vector Processors”, Cornell Computer Science Report 87-869, Ithaca, N.Y.
- Bischof C. H. and Van Loan C. (1986). “Computing the SVD on a Ring of Array Processors”, in Large Scale Problems, eds. J. Cullum and R. Willoughby, North Holland, 51–66.
- Brent R. P. and Luk F. T. (1985). “The Solution of Singular Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays”, SIAM J. Sci. and Stat. Comp. 6, 69–84.
- Brent R. P., Luk F. T., and Van Loan C. (1985). “Computation of the Singular Value Decomposition Using Mesh Connected Processors”, J. VLSI Computer Systems 1, 242–270.
- Eberlein P. J. (1987). “On Using the Jacobi Method on a Hypercube”, in Hypercube Multiprocessors, ed. M. T. Heath, SIAM Publications, Philadelphia.
- Luk F. T. (1980). “Computing the Singular Value Decomposition in the ILLIAC IV”, ACM Trans. Math. Soft. 6, 524–39.
- Luk F. T. (1986). “A triangular Processor Array for Computing Singular Values”, Lin. Alg. and Its Appl. 77, 259–274.
- Modi J. J. and Pryce J. D. (1985). “Efficient Implementation of Jacobi’s Diagonalization Method on the DAP”, Numer. Math. 45, 443–454.
- Sameh A. (1971). “On Jacobi and Jacobi-like Algorithms for a Parallel Computer”, Math. Comp. 25, 579–90.
- Schreiber R. (1986). “Solving Eigenvalue and Singular Value Problems on an Undersized Systolic Array”, SIAM J. Sci. and Stat. Comp. 7, 441–451.
- Scott D. S., Heath M. T., and Ward R. C. (1986). “Parallel Block Jacobi Eigenvalue Algorithms Using Systolic Arrays”, Lin. Alg. and Its Appl. 77, 345–356.
- Shroff G. and Scheiber R. (1987). “Convergence of Block Jacobi Methods”, Report 87-25, Comp. Sci. Dept., RPI, Troy, NY.

## 8.6. Метод «разделяй и властвуй»

В этом разделе мы опишем алгоритм «разделяй и властвуй» для вычисления собственных значений и собственных векторов симметричной трехдиагональной матрицы

$$T = \begin{bmatrix} a_1 & b_1 & & \cdots & 0 \\ b_1 & a_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & & b_{n-1} & a_n \end{bmatrix}.$$

Он вычисляет разложение Шура

$$Q^T T Q = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n), Q^T Q = I, \quad (8.6.1)$$

склеивая вместе разложения Шура двух специальных трехдиагональных проблем вдвое меньшего размера. Каждая из этих проблем может быть определена в свою очередь парой разложений Шура вчетверо меньшего размера и т. д. Полная процедура, благодаря Донгарре и Соренсену [Dongarra, Sorensen, 1987], удобна для параллельных вычислений.

### 8.6.1. Расщепление

Сначала мы покажем, как матрица  $T$  может быть «расщеплена» на две части при помощи одноранговой модификации. Для простоты пусть  $n = 2m$ . Определим вектор  $v \in \mathbb{R}^n$  следующим образом:

$$v = \begin{bmatrix} e_m^{(m)} \\ \theta e_1^{(m)} \end{bmatrix}. \quad (8.6.2)$$

Заметим, что для всех  $\rho \in \mathbb{R}$  матрица  $\tilde{T} = T - \rho vv^T$  идентична матрице  $T$ , кроме «четырех средних» элементов:

$$\tilde{T}(m:m+1, m:m+1) = \begin{bmatrix} a_m - \rho & b_m - \rho\theta \\ b_m - \rho\theta & a_{m+1} - \rho\theta^2 \end{bmatrix}.$$

Если мы положим  $\rho\theta = b_m$ , то

$$T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \rho vv^T,$$

где

$$T_1 = \begin{bmatrix} a_1 & b_1 & \cdots & 0 \\ b_1 & a_2 & \ddots & \vdots \\ \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & b_{m-1} \\ 0 & \cdots & b_{m-1} & \tilde{a}_m \end{bmatrix},$$

$$T_2 = \begin{bmatrix} \tilde{a}_{m+1} & b_{m+1} & \cdots & 0 \\ b_{m+1} & a_{m+2} & \ddots & \vdots \\ \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & b_{n-1} & a_n \end{bmatrix},$$

а  $\tilde{a}_m = a_m - \rho$  и  $\tilde{a}_{m+1} = a_{m+1} - \rho\theta^2$ .

### 8.6.2. Объединение разложений Шура

Пусть у нас есть ортогональные матрицы  $Q_1$  и  $Q_2$  размера  $m \times m$ , такие, что каждая из матриц  $Q_1^T T_1 Q_1 = D_1$  и  $Q_2^T T_2 Q_2 = D_2$  – диагональная. Если положить

$$U = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix},$$

то

$$U^T T U = U^T \left( \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \rho vv^T \right) U = D + \rho z z^T, \quad (8.6.3)$$

где матрица

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

диагональная и

$$z = U^T v = \begin{bmatrix} Q_1^T e_m \\ \theta Q_2^T e_1 \end{bmatrix}. \quad (8.6.4)$$

Сравнивая (8.6.1) и (8.6.3), мы видим, что для эффективного синтеза этих двух разложений Шура вдвое меньшего размера требуется быстрое и устойчивое вычисление ортогональной матрицы  $V$ , такой, что

$$V^T (D + \rho z z^T) V = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n). \quad (8.6.5)$$

После того как матрица  $V$  вычислена, мы полагаем  $Q = UV$ , чтобы получить (8.6.1).

### 8.6.3. Собственная система матрицы $D + \rho z z^T$

К счастью, матрицу вида диагональная + матрица ранга один можно диагонализовать очень быстро. Метод опирается на некоторые базовые теоретические результаты, которые мы сейчас установим.

**Лемма 8.6.1.** Пусть матрица  $D = \text{diag}(d_1, \dots, d_n) \in \mathbb{R}^{n \times n}$  обладает свойством  $d_1 > \dots > d_n$ . Предположим, что  $\rho \neq 0$  и что вектор  $z \in \mathbb{R}^n$  не имеет нулевых компонент. Если

$$(D + \rho z z^T) v = \lambda v, \quad v \neq 0,$$

то  $z^T v \neq 0$  и матрица  $D - \lambda I$  невырожденная.

*Доказательство.* Если  $\lambda \in \lambda(D)$ , то  $\lambda = d_i$  при некотором  $i$  и, следовательно,

$$0 = e_i^T [(D - \lambda I)v + \rho(z^T v)z] = \rho(z^T v)z_i.$$

Поскольку  $\rho$  и  $z_i$  не равны нулю, мы должны иметь равенство  $0 = z^T v$  и, таким образом,  $Dv = \lambda v$ . Однако матрица  $D$  имеет различные собственные значения и, следовательно,  $v \in \text{span}\{e_i\}$ . Но из этого следует, что  $0 = z^T v = z_i$  — противоречие. Таким образом, матрицы  $D$  и  $D + \rho z z^T$  не могут иметь общих собственных значений и  $z^T v \neq 0$ .  $\square$

**Теорема 8.6.2.** Пусть матрица  $D = \text{diag}(d_1, \dots, d_n) \in \mathbb{R}^{n \times n}$  и ее диагональные элементы удовлетворяют условию  $d_1 > \dots > d_n$ . Предположим, что  $\rho \neq 0$  и что вектор  $z \in \mathbb{R}^n$  не имеет нулевых компонент. Если  $V \in \mathbb{R}^{n \times n}$  — ортогональная матрица, такая, что

$$V^T (D + \rho z z^T) V = \text{diag}(\lambda_1, \dots, \lambda_n)$$

с  $\lambda_1 \geq \dots \geq \lambda_n$  и  $V = [v_1, \dots, v_n]$ , то

(a) Величины  $\lambda_i$  являются нулями функции  $f(\lambda) = 1 + \rho z^T (D - \lambda I)^{-1} z$ .

(b) Если  $\rho > 0$ , то  $\lambda_1 > d_1 > \lambda_2 > \dots > \lambda_n > d_n$ .

Если  $\rho < 0$ , то  $d_1 > \lambda_1 > d_2 > \dots > d_n > \lambda_n$ .

(c) Собственный вектор  $v_i$  соправлен с  $(D - \lambda_i I)^{-1} z$ .

*Доказательство.* Если  $(D + \rho z z^T) v = \lambda v$ , то

$$(D - \lambda I)v + \rho(z^T v)z = 0. \quad (8.6.6)$$

Мы знаем из леммы 8.6.1, что матрица  $D - \lambda I$  невырожденная. Следовательно,

$$v \in \text{span}\{(D - \lambda I)^{-1} z\},$$

что устанавливает (c). Кроме того, если мы умножим матрицу  $z^T(D - \lambda I)^{-1}$  на обе части уравнения (8.6.6), то получим равенство

$$z^T v(1 + \rho z^T(D - \lambda I)^{-1} z) = 0.$$

Согласно лемме 8.6.1,  $z^T v \neq 0$ , и, следовательно, это равенство показывает, что если  $\lambda \in \lambda(D + \rho z z^T)$ , то  $f(\lambda) = 0$ . Мы должны показать, что все нули функции  $f$  являются собственными значениями матрицы  $D + \rho z z^T$  и что имеет место чередование (b).

Для того чтобы сделать это, посмотрим более внимательно на уравнения

$$\begin{aligned} f(\lambda) &= 1 + \rho \left( \frac{z_1^2}{d_1 - \lambda} + \dots + \frac{z_n^2}{d_n - \lambda} \right), \\ f'(\lambda) &= -\rho \left( \frac{z_1^2}{(d_1 - \lambda)^2} + \dots + \frac{z_n^2}{(d_n - \lambda)^2} \right). \end{aligned}$$

Заметим, что функция  $f$  монотонна между полюсами. Это позволяет нам заключить, что если  $\rho > 0$ , то  $f$  имеет точно  $n$  корней, по одному в каждом из интервалов

$$(d_n, d_{n-1}), \dots, (d_2, d_1), (d_1, \infty).$$

Если  $\rho < 0$ , то функция  $f$  также имеет точно  $n$  корней, по одному на каждом из интервалов

$$(-\infty, d_n), (d_n, d_{n-1}), \dots, (d_2, d_1).$$

В обоих случаях из этого следует, что нули функции  $f$  – это в точности собственные значения матрицы  $D + \rho v v^T$ .  $\square$

Эта теорема наводит на мысль для вычисления матрицы  $V$  (a) найти корни  $\lambda_1, \dots, \lambda_n$  функции  $f$  и затем (b) вычислить столбцы матрицы  $V$ , нормализуя векторы  $(D - \lambda_i I)^{-1} z$  при  $i = 1:n$ . До того как мы начнем рассматривать практические аспекты этих вычислений, мы покажем, что точно такому же плану можно следовать, даже если встречаются кратные  $d_i$  и нулевые  $z_i$ .

**Теорема 8.6.3.** *Если матрица  $D = \text{diag}(d_1, \dots, d_n)$  и вектор  $z \in \mathbb{R}^n$ , то существует ортогональная матрица  $V_1$ , такая, что если  $V_1^T D V_1 = \text{diag}(\mu_1, \dots, \mu_n)$  и  $w = V_1^T z$ , то*

$$\mu_1 > \mu_2 > \dots > \mu_r \geq \mu_{r+1} \geq \dots \geq \mu_n,$$

$$w_i \neq 0 \text{ при } i = 1:r \text{ и } w_i = 0 \text{ при } i = r+1:n.$$

*Доказательство.* Мы дадим конструктивное доказательство, основанное на двух элементарных операциях. (a) Пусть  $d_i = d_j$  при некотором  $i < j$ . Пусть  $J(i, j, \theta)$  есть вращение Якоби в плоскости  $(i, j)$ , обладающее тем свойством, что  $j$ -я компонента матрицы  $J(i, j, \theta)^T$  – нулевая. Нетрудно показать, что  $J(i, j, \theta)^T D J(i, j, \theta) = D$ . Следовательно, мы можем обнулить компоненту вектора  $z$ , если существует кратное  $d_i$ . (b) Если  $z_i = 0, z_j \neq 0$ , и  $i < j$ , то пусть  $P$  есть единичная матрица с переставленными столбцами  $i$  и  $j$ . Тогда матрица  $P^T D P$  – диагональная,  $(P^T z)_i \neq 0$ , и  $(P^T z)_j = 0$ . Следовательно, мы можем переставить все нулевые компоненты  $z_i$  «вниз». Понятно, что повторением операций (a) и (b) можно получить требуемую каноническую структуру. Матрица  $V_1$  является произведением матриц вращения.  $\square$

#### 8.6.4. Практический синтез

Практическое определение разложения (8.6.5) состоит в разумном применении теорем 8.6.2 и 8.6.3. Матрица  $V$  находится после этого как произведение  $V = V_1 V_2$ . Ортогональную матрицу  $V$  можно получить, выполняя редукцию, указанную в

теореме 8.6.3. Конечно, на практике нам нужен критерий для принятия решения, когда два диагональных элемента различные и когда элемент  $z$  можно рассматривать как нулевой. Пусть  $tol$  – небольшая величина порядка машинной точности, например,  $tol = \mathbf{u}(\|D\|_2 + |\rho| \|z\|_2)$ . Определим ортогональную матрицу  $V_1$  и целое число  $\hat{r}(1 \leq \hat{r} \leq n)$ , такие, что  $V_1^T D V_1 = \text{diag}(\mu_1, \dots, \mu_n)$  и  $w = V_1^T z$  с (a)  $\mu_i - \mu_{i+1} > tol$  при  $i = 1 : \hat{r} - 1$ , (b)  $|w_i| \geq tol$  при  $i = 1 : \hat{r}$  и (c)  $|w_i| < tol$  при  $i = \hat{r} + 1 : n$ . Выполняя замещение  $D \leftarrow V_1^T D V_1$  и  $z \leftarrow V_1^T z$ , мы придем к задаче нахождения матрицы  $\tilde{V}_2 \in \mathbb{R}^{\hat{r} \times \hat{r}}$ , такой, что

$$\tilde{V}_2^T (D(1 : \hat{r}, 1 : \hat{r}) + \rho z(1 : \hat{r}) z(1 : \hat{r})^T) \tilde{V}_2 = \text{diag}(\lambda_1, \dots, \lambda_{\hat{r}}).$$

Замечая, что теорема 8.6.2 применима к этой подзадаче, поставим в качестве первой задачи нахождение  $\hat{r}$  нулей функции

$$f(\lambda) = 1 + \rho \sum_{i=1}^{\hat{r}} \frac{z_i^2}{d_i - \lambda}.$$

Пусть  $\rho > 0$ . (Если это не так, заменим  $\rho$  на  $-\rho$  и  $D$  на  $-D$ .) Пусть  $\lambda_i$  – единственный нуль функции  $f$  в открытом интервале  $(d_i, d_{i-1})$ , где мы полагаем  $d_0 = \infty$ . Нелинейное уравнение  $f(\lambda) = 0$  в интервале  $(d_i, d_{i-1})$  может быть решено с помощью очень быстрых ньютоно-подобных итераций, основанных на рациональном приближении функции  $f$  на каждой текущей итерации. Секулярное уравнение прекрасно устроено, и есть возможность начать ньютоновский процесс с хорошего начального приближения. Если  $\hat{\lambda}_i$  – вычисленный корень секулярного уравнения, то  $i$ -й столбец матрицы  $\tilde{V}_2$  находится посредством нормализации вектора  $(D(1 : \hat{r}, 1 : \hat{r}) - \lambda_i I_{\hat{r}})^{-1} z(1 : \hat{r})$ . Заметим, что  $V = V_1 V_2$ , где  $V_2 = \text{diag}(\tilde{V}_2, I_{n-\hat{r}})$ . Детали можно найти в работе [Dongarra, Sorensen, 1987] вместе с анализом ошибок округления, показывающего ортонормальность вычисляемой матрицы  $V$ .

### 8.6.5. Полный процесс с параллелизмом

Рассмотрев операции расщепления и синтеза, мы готовы к тому, чтобы пояснить весь процесс и то, как он может быть реализован на многопроцессорной системе. Для ясности предположим, что  $n = 8N$  при некотором положительном целом  $N$  и, что выполняются три уровня расщепления. Мы можем изобразить это при помощи двоичного дерева следующим образом:

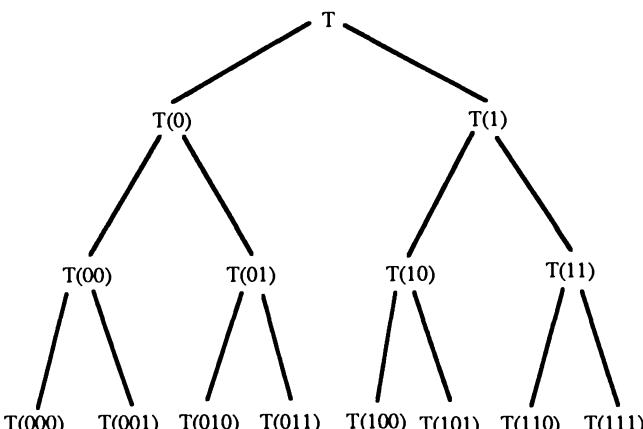


Рис. 8.6.1. Вычислительное дерево. Здесь для индексов использованы двоичные коды и схемы.

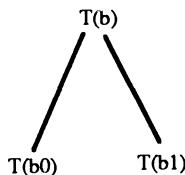


Рис. 8.6.2. Вершина.

Рис. 8.6.2 означает, что собственная система трехдиагональной матрицы  $T(b)$  получается из собственных систем трехдиагональных матриц  $T(b0)$  и  $T(b1)$ . Например, собственные системы  $N \times N$  матриц  $T(110)$  и  $T(111)$  в комбинации дают собственную систему трехдиагональной матрицы  $T(11)$  размера  $2N \times 2N$ .

Для алгоритмов, имеющих древовидную структуру, всегда существует опасность, что параллелизм теряется при подъеме к корню, но в нашей задаче это не так. Для того чтобы увидеть это, предположим, что у нас есть 8 процессоров и что первая задача процессора  $\text{Proc}(b)$  состоит в вычислении разложения Шура матрицы  $T(b)$ , где  $b = 000, 001, 010, 011, 100, 101, 110, 111$ . Эта часть вычислений вполне сбалансирована и не требует межпроцессорных связей. (Мы игнорируем исчерпывания теоремы 8.3, которые вызывают значительный дисбаланс загруженности.) На следующем уровне нужно выполнить четыре операции склейки:  $T(00)$ ,  $T(01)$ ,  $T(10)$ ,  $T(11)$ . Однако каждое из этих вычислений изящно разделяется на части и мы можем использовать два процессора для решения каждой задачи. Например, поскольку секулярное уравнение, которое лежит в основе синтеза  $T(00)$ , известно обоим процессорам  $T(000)$  и  $T(001)$ , то каждый из них может взять на себя получение половины всех собственных значений и соответствующих собственных векторов. Точно так же четверки процессоров могут быть использованы для решения задач  $T(0)$  и  $T(1)$ . Все 8 процессоров могут разделиться при вычислении собственной системы матрицы  $T$ . Таким образом на любом уровне можно достичь полного параллелизма, поскольку вычисления собственных значений/собственных векторов не зависят друг от друга.

### Задачи

**8.6.1.** Предложить алгоритм определения величин  $\rho$  и  $\theta$  в (8.6.3), обладающих свойством  $\theta \in \{-1, 1\}$  и  $\min\{|a_r - \rho|, |a_{r+1} - \rho|\}$  - максимальен.

**8.6.2.** Показать, что если  $D = \text{diag}(d_1, \dots, d_n)$ ,  $z_i^2 + z_j^2 \neq 0$ ,  $i < j$ , и  $(J(i, j, \theta)^T z)_j = 0$ , то внедиагональная часть матрицы  $\bar{J}(i, j, \theta)^T D \bar{J}(i, j, \theta)$  имеет норму Фробениуса, равную

$$|d_i - d_j| |z_i z_j| / (z_i^2 + z_j^2).$$

**8.6.3.** Какие связи необходимы между процессорами для части  $T_b$ ? Можно ли распределить между процессорами работу, связанную с кратными  $d_i$  и нулевыми  $z_i$ ?

**8.6.4.** Если матрица  $T$  положительно определена, следует ли из этого, что матрицы  $T_1$  и  $T_2$  из § 8.6.1 положительно определены?

### Замечания и литература к § 8.6

Алгоритм, обсуждавшийся в этом разделе, детально рассмотрен в работе

Dongarra J. J. and Sorensen D. C. (1987). “A Fully Parallel Algorithm for the Symmetric Eigenvalue Problem”, SIAM J. Sci. and Stat. Comp. 8, S139–S154.

Критические аспекты этого алгоритма берут начало из следующих работ

Bunch J. R., Nielsen C. P., and Sorensen D. C. (1978). “Rank – One Modification of the Symmetric Eigenproblem”, Numer. Math. 31, 31–48.

Cuppen J. J. M. (1981). “A Divide and Conquer Method for the Symmetric Eigenproblem”, Numer. Math. 36, 177–95.

Golub G. H. (1973). "Some Modified Matrix Eigenvalue Problems", SIAM Review 15, 318–44.

SVD-версии и способ перехода к общему ленточному случаю обсуждаются в

Arbenz P., Gander W., and Golub G. H. (1988). "Restricted Rank Modification of the Symmetric Eigenvalue Problem: Theoretical Considerations", Lin. Alg. and Its Applic. 104, 75–95.

Arbenz P. and Golub G. H. (1988). "On the Spectral Decomposition of Hermitian Matrices Subject to Indefinite Low Rank Perturbations with Applications", SIAM J. Matrix Anal. Appl. 9, 40–58.

Jessup E. R. and Sorensen D.C. (1988). "A Parallel Algorithm for Computing the Singular Value Decomposition", Report 102, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

## 8.7. Более общие проблемы собственных значений

Часто в обобщенной проблеме собственных значений  $Ax = \lambda Bx$  матрица  $A$  – симметричная, а матрица  $B$  – симметричная и положительно определенная. Пучки такого рода называют симметрично определенными пучками. Это свойство сохраняется при преобразовании конгруэнтности:

$$\begin{array}{ccc} A - \lambda B & & (X^T AX) - \lambda (X^T BX) \\ \Leftrightarrow & & \\ \text{симметрично-определен-} & & \text{симметрично-определен-} \\ \text{ный} & & \text{ный} \end{array}$$

В этом разделе мы приводим различные сохраняющие структуру процедуры, решающие такую проблему собственных значений. Обсуждается также связанная с ней обобщенная проблема сингулярных значений.

### 8.7.1. Математические основы

Хотя QZ-алгоритм из § 7.7 можно использовать для решения симметрично определенной проблемы, он разрушает как симметричность, так и положительную определенность. То, что мы ищем, это устойчивый, эффективный алгоритм, вычисляющий матрицу  $X$ , такую, что обе матрицы  $X^T AX$  и  $X^T BX$  имеют «каноническую форму». Очевидной формой, отвечающей этой цели, является диагональная форма

**Теорема 8.7.1.** Предположим, что  $A$  и  $B$  – симметричные матрицы размера  $n \times n$ , и определим  $C(\mu)$  как

$$C(\mu) = \mu A + (1 - \mu) B, \quad \mu \in \mathbb{R}. \quad (8.7.1)$$

Если существует некоторое  $\mu \in [0, 1]$ , такое, что матрица  $C(\mu)$  неотрицательно определенная и

$$\text{null}(C(\mu)) = \text{null}(A) \cap \text{null}(B),$$

то существует невырожденная матрица  $X$ , такая, что обе матрицы  $X^T AX$  и  $X^T BX$  диагональные.

**Доказательство.** Пусть величина  $\mu \in [0, 1]$  выбрана так, что матрица  $C(\mu)$  неотрицательно определенная и обладает свойством  $\text{null}(C(\mu)) = \text{null}(A) \cap \text{null}(B)$ . Пусть

$$Q_1^T C(\mu) Q_1 = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}, \quad D = \text{diag}(d_1, \dots, d_k), \quad d_i > 0,$$

– разложение Шура матрицы  $C(\mu)$  и  $X_1 = Q_1 \text{diag}(D^{-1/2}, I_{n-k})$ . Если  $A_1 = X_1^T AX_1$ ,

$B = X_1^T BX_1$  и  $C_1 = X_1^T C(\mu) X_1$ , то

$$C_1 = \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix} = \mu A_1 + (1 - \mu) B_1.$$

Так как  $\text{span}\{e_{k+1}, \dots, e_n\} = \text{null}(C_1) = \text{null}(A_1) \cap \text{null}(B_1)$ , то из этого следует, что матрицы  $A_1$  и  $B_1$  имеют следующую блочную структуру:

$$A_1 = \begin{bmatrix} A_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix}, \quad B_1 = \begin{bmatrix} B_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix}.$$

Более того,  $I_k = \mu A_{11} + (1 - \mu) B_{11}$ .

Пусть  $\mu \neq 0$ . Тогда, если  $Z^T B_{11} Z = \text{diag}(b_1, \dots, b_k)$  разложение Шура матрицы  $B_{11}$  и мы положим  $X = X_1 \text{diag}(Z, I_{n-k})$ , то

$$X^T BX = \text{diag}(b_1, \dots, b_k, 0, \dots, 0) \equiv D_B$$

и

$$\begin{aligned} X^T AX &= \frac{1}{\mu} X^T (C(\mu) - (1 - \mu) B) X = \\ &= \frac{1}{\mu} \left( \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix} - (1 - \mu) D_B \right) \equiv D_A. \end{aligned}$$

С другой стороны, если  $\mu = 0$ , то пусть  $Z^T A_{11} Z = \text{diag}(a_{11}, \dots, a_{kk})$  – разложение Шура матрицы  $A_{11}$  и  $X = X_1 \text{diag}(Z, I_{n-k})$ . Несложно проверить, что в этом случае также обе матрицы  $X^T AX$  и  $X^T BX$  диагональные.  $\square$

Часто условия теоремы 8.7.1 выполняются из-за того, что или матрица  $A$ , или матрица  $B$  положительно определенная.

**Следствие 8.7.2.** Если  $A - \lambda B \in \mathbb{R}^{n \times n}$  симметрично-определенный пучок матриц, то существует невырожденная матрица  $X = [x_1, \dots, x_n]$ , такая, что

$$X^T AX = \text{diag}(a_1, \dots, a_n), \quad X^T BX = \text{diag}(b_1, \dots, b_n).$$

Кроме того,  $Ax_i = \lambda_i Bx_i$  при  $i = 1:n$ , где  $\lambda_i = a_i/b_i$ .

**Доказательство.** Полагая  $\mu = 0$  в теореме 8.7.1, мы видим, что симметрично-определенный пучок может быть одновременно диагонализован. Остальная часть следствия легко проверяется.  $\square$

**Пример 8.7.1.** Если

$$A = \begin{bmatrix} 229 & 163 \\ 163 & 116 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 81 & 59 \\ 59 & 43 \end{bmatrix},$$

то пучок  $A - \lambda B$  симметрично-определен и  $\lambda(A, B) = \{5, -1/2\}$ . Если

$$X = \begin{bmatrix} 3 & -5 \\ -4 & 7 \end{bmatrix},$$

то  $X^T AX = \text{diag}(5, -1)$  и  $X^T BX = \text{diag}(1, 2)$ .

Стюарт [Stewart, 1979b] разработал теорию возмущения для симметричного пучка  $A - \lambda B$ , удовлетворяющего условию

$$c(A, B) = \min_{\|x\|_2=1} (x^T Ax)^2 + (x^T Bx)^2 > 0. \quad (8.7.2)$$

Эту скалярную величину называют *числом Крауфорда* пучка  $A - \lambda B$ .

**Теорема 8.7.3.** Пусть  $A - \lambda B$  – симметрично-определенный пучок размера  $n \times n$  с собственными значениями

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n.$$

Пусть  $E_A$  и  $E_B$  – симметричные  $n \times n$  матрицы, удовлетворяющие неравенству

$$\varepsilon^2 = \|E_A\|_2^2 + \|E_B\|_2^2 < c(A, B).$$

Тогда  $(A + E_A) - \lambda(B + E_B)$  является симметрично-определенным пучком с собственными значениями  $\mu_1 \geq \dots \geq \mu_n$ , удовлетворяющими неравенству

$$|\operatorname{arctg}(\lambda_i) - \operatorname{arctg}(\mu_i)| \leq \operatorname{arctg}(\varepsilon/c(A, B))$$

при  $i = 1:n$ .

*Доказательство.* См. [Stewart, 1979b].  $\square$

## 8.7.2. Методы для симметрично-определенной проблемы

Возвращаясь к алгоритмическим вопросам, мы прежде всего рассмотрим метод решения симметрично-определенной проблемы, который использует разложение Холецкого и симметричный QR-алгоритм.

**Алгоритм 8.7.1.** Для заданных матриц  $A = A^T \in \mathbb{R}^{n \times n}$  и  $B = B^T \in \mathbb{R}^{n \times n}$ , где  $B$  – положительно определенная матрица, следующий алгоритм вычисляет невырожденную матрицу  $X$ , такую, что  $X^T BX = I_n$  и  $X^T AX = \operatorname{diag}(a_1, \dots, a_n)$ .

Вычислить разложение Холецкого  $B = GG^T$ ,

используя алгоритм 4.2.2.

Вычислить матрицу  $C = G^{-1}AG^{-T}$ .

Использовать симметричный OR-алгоритм для вычисления разложения Шура  $Q^T C Q = \operatorname{diag}(a_1, \dots, a_n)$ .

Положить  $X = G^{-T}Q$ .

Этот алгоритм требует примерно  $14n^3$  флоков. В практической реализации, матрицу  $A$  можно замещать матрицей  $C$ . Детали смотри в [Martin, Wilkinson, 1968c]. Заметим, что

$$\lambda(A, B) = \lambda(A, GG^T) = \lambda(G^{-1}AG^{-T}, I) = \lambda(C) = \{a_1, \dots, a_n\}.$$

Если  $\hat{a}_i$  – вычисленное собственное значение, полученное по алгоритму 8.7.1, то можно показать, что  $\hat{a}_i \in \lambda(G^{-1}AG^{-T} + E_i)$ , где  $\|E_i\|_2 \approx \|\mathbf{u}\|_2 \|A\|_2 \|B^{-1}\|_2$ . Следовательно, если матрица  $B$  плохо обусловлена, то  $\hat{a}_i$  может быть несколько испорчено погрешностью округления, даже если  $a_i$  – хорошо обусловленное обобщенное собственное значение. Проблема, конечно, в этом случае в том, что матрица  $C = G^{-1}AG^{-T}$  может иметь очень большие элементы, если матрица  $B$  и, следовательно,  $G$ , плохо обусловлена. Эту трудность можно иногда обойти, заменив матрицу  $G$  в алгоритме 8.7.1 на матрицу  $V D^{-1/2}$ , где  $V^T B V = D$  – разложение Шура матрицы  $B$ . Если диагональные элементы матрицы  $D$  упорядочены в порядке возрастания, то большие элементы в матрице  $C$  будут сконцентрированы в верхнем

левом углу. Малые собственные значения матрицы  $C$  могут быть вычислены без чрезмерной погрешности округления (или таковы эвристические выводы). Дальнейшее смотри в [Wilkinson, AEP], с. 337–38.

**Пример 8.7.2.** Если

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}, \quad G = \begin{bmatrix} 0.001 & 0 & 0 \\ 1 & 0.001 & 0 \\ 2 & 1 & 0.001 \end{bmatrix}$$

и  $B = GG^T$ , то два наименьших собственных значения пучка  $A - \lambda B$  есть

$$a_1 = -0.619402940600584, \quad a_2 = 1.627440079051887.$$

Если использовать 17-разрядную арифметику с плавающей запятой, то эти собственные значения будут вычислены с полной машинной точностью, если симметричный QR-алгоритм применять к матрице  $f(I(D^{-1/2}V^T A V D^{-1/2}))$ , где  $B = V D V^T$  разложение Шура матрицы  $B$ . С другой стороны, если применяется алгоритм 8.7.1, то

$$\hat{a}_1 = -0.619373517376444, \quad \hat{a}_2 = 1.627516601905228.$$

Причина получения только четырех правильных значащих цифр в том, что  $k_2(B) \approx 10^{18}$ .

Обусловленность матрицы  $X$  в алгоритме 8.7.1 иногда можно улучшить, заменяя матрицу  $B$  подходящей выпуклой комбинацией матриц  $A$  и  $B$ . Связь между собственными значениями этого модифицированного пучка и собственными значениями исходного пучка подробно рассматривается в доказательстве теоремы 8.7.1.

Другая проблема, касающаяся алгоритма 8.7.1, связана с тем, что матрица  $G^{-1}AG^{-T}$ , вообще говоря, плотная, даже если матрицы  $A$  и  $B$  разреженные. Это серьезная проблема, так как многие симметрично определенные задачи, возникающие на практике, большие и разреженные.

Крауфорд [Crawford, 1973] показал, как эффективно применять алгоритм 8.7.1, когда матрицы  $A$  и  $B$  – ленточные. Однако за исключением этого случая подход, связанный с одновременной диагонализацией не практичен для больших разреженных симметрично-определеных проблем.

Альтернативная идея состоит в том, чтобы обобщить итерации с отношением Рэлея (8.4.4) следующим образом

Задан вектор  $x_0$  с  $\|x_0\|_2 = 1$ .

**for**  $k = 0, 1, \dots$

$$\mu_k = x_k^T A x_k / x_k^T B x_k$$

Решить уравнение  $(A - \mu_k B) z_{k+1} = B x_k$

относительно  $z_{k+1}$ .

$$x_{k+1} = z_{k+1} / \|z_{k+1}\|_2$$

**end**

Математической основой этих итераций является то, что

$$\lambda = \frac{x^T A x}{x^T B x} \tag{8.7.4}$$

минимизирует

$$f(\lambda) = \|Ax - \lambda Bx\|_B, \tag{8.7.5}$$

где  $\|\cdot\|_B$  определена как  $\|z\|_B^2 = z^T B^{-1} z$ . Математические свойства итераций (8.7.3) подобны свойствам итераций (8.4.4). Их применимость зависит от того, может ли быть легко решена система вида  $(A - \mu B)z = x$ . Подобное замечание относится

и к следующим обобщенным ортогональным итерациям

Задана матрица  $Q_0 \in \mathbb{R}^{n \times p}$  с  $Q_0^T Q_0 = I_p$ .

**for**  $k = 1, 2, \dots$

Решить уравнение  $BZ_k = AQ_{k-1}$  относительно  $Z_k$ . (8.7.6)

$Z_k = Q_k R_k$  (QR-разложение)

**end**

Этот алгоритм математически эквивалентен алгоритму (7.3.4) с матрицей  $A$ , замененной на  $B^{-1}A$ . Его практичность зависит от того, насколько легко решать линейные системы вида  $Bz = y$ .

Иногда матрицы  $A$  и  $B$  такие большие, что ни (8.7.3), ни (8.7.6) не могут помочь. В этой ситуации можно прибегнуть к любому из градиентных или релаксационных алгоритмов. См. [Stewart, 1976b], где приведен обширный список литературы.

### 8.7.3. Обобщенная проблема сингулярных значений

Мы закончим некоторыми замечаниями о симметрических пучках, которые имеют вид  $A^T A - \lambda B^T B$ , где  $A \in \mathbb{R}^{m \times n}$  и  $B \in \mathbb{R}^{p \times n}$ . Этот пучок допускает *обобщенное сингулярное разложение* (GSVD-разложение), которое полезно для особым образом устроенных методов наименьших квадратов (§ 12.1). Заметим, что по теореме 8.7.1 существует невырожденная матрица  $X \in \mathbb{R}^{n \times n}$ , такая, что обе матрицы  $X^T(A^T A)X$  и  $X^T(B^T B)X$  – диагональные. Значение GSVD в том, что эти диагонализации можно получить без формирования матриц  $A^T A$  и  $B^T B$ .

**Теорема 8.7.4 (Обобщенное сингулярное разложение).** Если  $A \in \mathbb{R}^{m \times n}$  с  $m \geq n$  и  $B \in \mathbb{R}^{p \times n}$ , то существуют ортогональные матрицы  $U \in \mathbb{R}^{m \times m}$  и  $V \in \mathbb{R}^{p \times p}$  и обратимая матрица  $X \in \mathbb{R}^{n \times n}$ , такие, что

$$U^T A X = C = \text{diag}(c_1, \dots, c_n), \quad c_i \geq 0,$$

и

$$V^T B X = S = \text{diag}(s_1, \dots, s_q), \quad s_i \geq 0,$$

где  $q = \min(p, n)$ .

*Доказательство.* Доказательство этого разложения появилось в работе Van Loan [1976]. Мы приведем более конструктивное доказательство в духе Paige, Saunders [1981]. Для ясности предположим, что  $\text{null}(A) \cap \text{null}(B) = \{0\}$  и  $p \geq n$ . Мы оставим читателю обобщить доказательство так, чтобы оно охватывало все случаи.

Пусть

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R \quad (8.7.6)$$

является QR-факторизацией с матрицами  $Q_1 \in \mathbb{R}^{m \times n}$ ,  $Q_2 \in \mathbb{R}^{p \times n}$  и  $R \in \mathbb{R}^{n \times n}$ . Пейдж и Саундерс показали, что SVD матриц  $Q_1$  и  $Q_2$  связаны в том смысле, что

$$Q_1 = UCW^T, \quad Q_2 = VSW^T. \quad (8.7.7)$$

Здесь  $U$ ,  $V$  и  $W$  – ортогональные матрицы,  $C = \text{diag}(c_i)$  с  $0 \leq c_1 \leq \dots \leq c_n$ ,  $S = \text{diag}(s_i)$  с  $s_1 \geq \dots \geq s_n$  и  $C^T C + S^T S = I_n$ . Разложение (8.7.7) – вариант CS разложения из § 2.6, поэтому  $A = Q_1 R = UC(W^T R)$  и  $B = Q_2 R = VS(W^T R)$ . Теорему доказывает подстановка  $X = (W^T R)^{-1}$ ,  $D_A = C$  и  $D_B = S$ . Обратимость матрицы  $R$  следует из нашего предположения, что  $\text{null}(A) \cap \text{null}(B) = \{0\}$ . Мы оставляем читателю детально рассмотреть случай, когда матрицы  $A$  и  $B$  имеют общие нуль-векторы.  $\square$

Элементы множества  $\sigma(A, B) \equiv \{c_1/s_1, \dots, c_n/s_n\}$  называют *обобщенными сингулярными числами* матриц  $A$  и  $B$ . Заметим, что  $\sigma \in \sigma(A, B)$  означает, что  $\sigma^2 \in \lambda(A^T A, B^T B)$ . Эта теорема является обобщением SVD, так как если  $B = I_n$ , то  $\sigma(A, B) = \sigma(A)$ .

Наше доказательство GSVD имеет практическую ценность, так как Стюарт (Stewart, 1982) и Ван Лоан (Van Loan, 1985) показали, как вычислять CS-разложение устойчиво. Единственной хитрой частью является обращение матрицы  $W^T R$  для получения матрицы  $X$ . Заметим, что столбцы матрицы  $X = [x_1, \dots, x_n]$  удовлетворяют равенству

$$s_i^2 A^T A x_i = c_i^2 B^T B x_i, \quad i = 1:n,$$

и, следовательно, если  $s_i \neq 0$ , то  $A^T A x_i = \sigma_i^2 B^T B x_i$ , где  $\sigma_i = c_i/s_i$ . Поэтому вектор  $x_i$  естественно назвать обобщенным сингулярным вектором пары  $(A, B)$ .

В некоторых приложениях требуется ортогональный базис для построения подпространства обобщенных сингулярных векторов  $\text{span}\{x_{i_1}, \dots, x_{i_k}\}$ . Мы покажем, как это можно сделать без каких бы то ни было матричных обращений или перекрестных умножений:

- Вычислить QR-разложение

$$\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R.$$

- Вычислить CS-разложение

$$Q_1 = U C W^T, \quad Q_2 = V S W^T$$

и упорядочить диагональные элементы матриц  $C$  и  $S$  так, чтобы

$$\{c_1/s_1, \dots, c_k/s_k\} = \{c_{i_1}/s_{i_1}, \dots, c_{i_k}/s_{i_k}\}.$$

- Вычислить ортогональную матрицу  $Z$  и верхнюю треугольную матрицу  $T$ , такие, что  $TZ = W^T R$ . (См. задачу 8.7.5.) Заметим, что если  $X^{-1} = W^T R = TZ$ , то  $X = Z^T T^{-1}$  и, следовательно, первые  $k$  столбцов матрицы  $Z$  являются ортогональным базисом в подпространстве  $\text{span}\{x_1, \dots, x_k\}$ .

### Задачи

**8.7.1.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, а  $G \in \mathbb{R}^{n \times n}$  нижняя треугольная и невырожденная. Предложить эффективный алгоритм вычисления  $C = G^{-1} A G^{-T}$ .

**8.7.2.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, а матрица  $B \in \mathbb{R}^{n \times n}$  симметричная положительно определенная. Предложить алгоритм вычисления собственных значений матрицы  $AB$ , использующий разложение Холецкого и симметричный QR-алгоритм.

**8.7.3.** Показать, что если матрица  $C$  вещественная и диагонализуемая, то существуют симметричные матрицы  $A$  и  $B$  ( $B$  невырожденна), такие, что  $C = AB^{-1}$ . Это показывает, что симметричный пучок  $A - \lambda B$  – достаточно общий случай.

**8.7.4.** Показать, как преобразовать проблему  $Ax = \lambda Bx$  в обобщенную проблему сингулярных значений, если матрицы  $A$  и  $B$  – обе симметричные и неотрицательно определенные.

**8.7.5.** Для заданной матрицы  $Y \in \mathbb{R}^{n \times n}$  показать, как вычислить матрицы Хаусхолдера  $H_1, \dots, H_n$ , такие, что матрица  $YH_n \dots H_2$  верхняя треугольная. Подсказка: матрица  $H_k$  обнуляет  $k$ -ю строку.

**8.7.6.** Пусть

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \lambda \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix},$$

где  $A \in \mathbf{R}^{m \times n}$ ,  $B_1 \in \mathbf{R}^{m \times m}$  и  $B_2 \in \mathbf{R}^{n \times n}$ . Предположим, что матрицы  $B_1$  и  $B_2$  положительно определенные с треугольными матрицами разложения Холецкого  $G_1$  и  $G_2$  соответственно. Связать обобщенные собственные значения этой проблемы с сингулярными числами матрицы  $G_1^{-1} A G_2^{-T}$ .

### Замечания и литература к § 8.7

Превосходный обзор вычислительных методов для симметрично-определеных пучков дан в Stewart G. W. (1976b). "A Bibliographical Tour of the Large Sparse Generalized Eigenvalue Problem", in Sparse Matrix Computations, ed., J. R. Bunch and D. J. Rose, Academic Press, New York.

Следующие статьи представляют особый интерес:

- Crawford C. R. (1973). "Reduction of a Band Symmetric Generalized Eigenvalue Problem", Comm. ACM 16, 41–44.  
 Crawford C. R. (1976). "A Stable Generalized Eigenvalue Problem", SIAM J. Num. Anal. 13, 854–60.  
 Fix G. and Heiberger R. (1972). "An Algorithm for the ILL–Conditioned Generalized Eigenvalue Problem", SIAM J. Num. Anal. 9, 78–88.  
 Martin R. S. and Wilkinson J. H. (1968c). "Reduction of a Symmetric Eigenproblem  $Ax = \lambda Bx$  and Related Problems to Standard Form", Numer. Math. 11, 99–110.  
 Peters G. and Wilkinson J. H. (1969). "Eigenvalue of  $Ax = \lambda Bx$  with Band Symmetric A and B", Comp. J. 12, 398–404.  
 Ruhe A. (1974). "SOR Methods for the Eigenvalue Problem with Large Sparse Matrices", Math. Comp. 28, 695–710.

См. также

- Bunse-Gerstner (1984). "An Algorithm for the Symmetric Generalized Eigenvalue Problem", Lin. Alg. and Its Appl. 58, 43–68.  
 Crawford C. R. (1986). "Algorithm 646 PDFIND: A Routine to Find a Positive Definite Linear Combination of Two Real Symmetric Matrices", ACM Trans. Math. Soft. 12, 278–282.  
 Crawford C. R. and Moon Y. S. (1983). "Finding a Positive Linear Combination of Two Matrices", SIAM J. Sci. and Stat. Comp. 7, 1147–1159.  
 Heath M. T., Laub A. J., Paige C. C., and Ward R. C. (1986). "Computing the SVD of a Product of Two Matrices", SIAM J. Sci. and Stat. Comp. 7, 1147–1159.

Одновременное преобразование двух симметрических матриц к диагональному виду обсуждается в

- Berman A. and Ben-Israel A. (1971). "A Note on Pencils of Hermitian or Symmetric Matrices", SIAM J. Appl. Math. 2, 51–54.  
 Mahindar K. N. (1979). "Linear Combinations of Hermitian and Real Symmetric Matrices", Lin. Alg. and Its Appl. 25, 95–105.  
 Uhlig F. (1973). "Simultaneous Block Diagonalization of Two Real Symmetric Matrices" Lin. Alg. and Its Appl. 7, 281–89.  
 Uhlig F. (1976). "A Canonical Form for a Pair of Real Symmetric Matrices That Generate a Nonsingular Pencil", Lin. Alg. and Its Appl. 14, 189–210.

Теория возмущения, которую мы привели для симметрично-определенной проблемы, взята из работы

- Stewart G. W. (1979b). "Perturbation Bounds for the Definite Generalized Eigenvalue Problem", Lin. Alg. and Its Appl. 23, 69–86.

См. также

- Elsner L. and Guang Sun J. (1982). "Perturbation Theorems for the Generalized Eigenvalue Problem", Lin. Alg. and Its Appl. 48, 341–357.  
 Paige C. C. (1984). "A Note on a Result of Sun J.-Guang: Sensitivity of the CS and GSV Decompositions", SIAM J. Numer. Anal. 21, 186–191.  
 Guang Sun J. (1982). "A Note on Stewart's Theorem for Definite Matrix Pairs", Lin. Alg. and Its Appl. 48, 331–339.  
 Guang Sun J. (1983). "Perturbation Analysis for the Generalized Singular Value Problem", SIAM J. Numer. Anal. 20, 611–625.

- Обобщенное SVD и некоторые его приложения обсуждаются в
- Kågström (1985). “The Generalized Singular Value Decomposition and the General  $A - \lambda B$  Problem”, BIT 24, 568–583.
- Van Loan C. F. (1976). “Generalizing the Singular Value Decomposition”, SIAM J. Num. Anal. 13, 76–83.
- Paige C. C. and Saunders M. (1981). “Towards A Generalized Singular Value Decomposition”, SIAM J. Num. Anal. 18, 398–405.
- Устойчивые методы вычисления CS и обобщенного сингулярного разложений описаны в
- Paige C. C. (1986). “Computing the Generalized Singular Value Decomposition”, SIAM J. Si. and Stat. Comp. 7, 1126–1146.
- Stewart G. W. (1983). “A Method for Computing the Generalized Singular Value Decomposition”, in Matrix Pencils, ed. B. Kagström and A. Ruhe, Springer-Verlag, New York, pp. 207–20.
- Van Loan C. (1985). “Computing the CS and Generalized Singular Value Decomposition”, Numer. Math. 46, 479–492.

## Глава 9

# Методы Ланцоша

В этой главе мы изучаем метод Ланцоша – метод решения некоторого класса больших разреженных симметричных спектральных задач  $Ax = \lambda x$ . В методе используются частичные трехдиагонализации заданной матрицы  $A$ . Однако в отличие от подхода Хаусхолдера здесь не строятся какие-либо промежуточные полные матрицы. В равной степени важно то, что информация об экстремальных собственных значениях для  $A$  обычно появляется задолго до построения полной трехдиагонализации. Это делает метод Ланцоша особенно полезным в тех случаях, когда для  $A$  требуется найти небольшое число наибольших или наименьших собственных значений.

В § 9.1 представлены вывод и свойства метода в точной арифметике. Детально рассмотрены ключевые аспекты теории Каниэля–Пейджа. Эта теория объясняет необычные свойства сходимости процесса Ланцоша.

К несчастью, практическое использование метода Ланцоша несколько затрудняется ошибками округления. Центральная проблема – это потеря ортогональности получаемых итерационно векторов Ланцоша. С этим можно справиться несколькими способами – они обсуждаются в § 9.2.

В заключительном разделе мы показываем, как «идея Ланцоша» может быть применена к различным задачам, связанным с сингулярными числами, наименьшими квадратами и линейными уравнениями. Обсуждается также и несимметричный процесс Ланцоша.

В § 9.3 особый интерес представляет разработка метода сопряженных градиентов для симметричных положительно определенных линейных систем. Изучение связи метода Ланцоша и метода сопряженных градиентов будет продолжено в следующей главе.

## 9.1. Вывод и свойства сходимости

Пусть имеется большая разреженная симметричная матрица  $A \in \mathbb{R}^{n \times n}$  и нас интересует небольшое число ее наибольших и (или) наименьших собственных значений. Эта задача может быть решена с помощью метода, приписываемого Ланцошу (1950). Метод строит последовательность трехдиагональных матриц  $T_j$  с тем свойством, что экстремальные собственные значения для  $T_j \in \mathbb{R}^{j \times j}$  с ростом  $j$  дают все более точные оценки экстремальных собственных значений для  $A$ . В этом разделе мы выводим метод и исследуем его свойства в точной арифметике.

### 9.1.1. Подпространства Крылова

К выводу алгоритма Ланцоша можно подойти с различных сторон. Для того чтобы замечательные свойства сходимости метода не оказались полным сюрпри-

зом, мы предпочтаем начать с рассмотрения оптимизации отношения Рэлея

$$r(x) = \frac{x^T Ax}{x^T x}, \quad x \neq 0.$$

Напомним, что согласно теореме 8.1.2 максимальное и минимальное значения  $r(x)$  суть  $\lambda_1(A)$  и  $\lambda_n(A)$  соответственно. Предположим, что  $\{q_i\} \subseteq \mathbb{R}^n$  есть последовательность ортонормированных векторов. Определим скаляры  $M_j$  и  $m_j$  следующим образом:

$$M_j = \lambda_1(Q_j^T A Q_j) = \max_{y \neq 0} \frac{y^T (Q_j^T A Q_j) y}{y^T y} = \max_{\|y\|_2 = 1} r(Q_j y) \leq \lambda_1(A),$$

$$m_j = \lambda_n(Q_j^T A Q_j) = \min_{y \neq 0} \frac{y^T (Q_j^T A Q_j) y}{y^T y} = \min_{\|y\|_2 = 1} r(Q_j y) \geq \lambda_n(A),$$

где  $Q_j = [q_1, \dots, q_j]$ . Алгоритм Ланцоша можно получить, рассматривая вопрос о таком выборе векторов  $q_j$ , чтобы  $M_j$  и  $m_j$  были все более и более точными оценками для  $\lambda_1(A)$  и  $\lambda_n(A)$ .

Пусть вектор  $u_j \in \text{span}\{q_1, \dots, q_j\}$  таков, что  $M_j = r(u_j)$ . Поскольку  $r(x)$  возрастает наиболее быстро в направлении градиента

$$\nabla r(x) = \frac{2}{x^T x} (Ax - r(x)x),$$

мы можем гарантировать, что  $M_{j+1} > M_j$ , если  $q_{j+1}$  определяется так, что

$$\nabla r(u_j) \in \text{span}\{q_1, \dots, q_{j+1}\}. \quad (9.1.1)$$

(Предполагается, что  $\nabla r(u_j) \neq 0$ .) Аналогично, если вектор  $v_j \in \text{span}\{q_1, \dots, q_j\}$  удовлетворяет условию  $r(v_j) = m_j$ , то есть смысл потребовать, чтобы

$$\nabla r(v_j) \in \text{span}\{q_1, \dots, q_{j+1}\}, \quad (9.1.2)$$

так как  $r(x)$  убывает наиболее быстро в направлении  $-\nabla r(x)$ .

На первый взгляд, задача отыскания одного вектора  $q_{j+1}$ , удовлетворяющего этим двум требованиям, кажется неразрешимой. Однако, поскольку  $\nabla r(x) \in \text{span}\{x, Ax\}$ , ясно, что (9.1.1) и (9.1.2) можно удовлетворить одновременно при условии, что

$$\text{span}\{q_1, \dots, q_j\} = \text{span}\{q_1, Aq_1, \dots, A^{j-1}q_1\} \equiv \mathcal{K}(A, q_1, j),$$

и вектор  $q_{j+1}$  выбирается так, что  $\mathcal{K}(A, q_1, j+1) = \text{span}\{q_1, \dots, q_{j+1}\}$ . Таким образом, мы пришли к задаче вычисления ортонормированных базисов в подпространствах Крылова  $\mathcal{K}(A, q_1, j)$ .

## 9.1.2. Трехдиагонализация

Для эффективного вычисления базиса мы используем связь между трехдиагонализацией матрицы  $A$  и  $QR$ -разложением матрицы Крылова

$$K(A, q_1, n) = [q_1, Aq_1, A^2q_1, \dots, A^{n-1}q_1].$$

(См. § 7.4.4.) Именно, если матрица  $Q^T A Q = T$  трехдиагональная и  $Qe_1 = q_1$ , то соотношение

$$K(A, q_1, n) = Q[e_1, Te_1, T^2e_1, \dots, T^{n-1}e_1]$$

является  $QR$ -разложением для  $K(A, q_1, n)$ . Таким образом, векторы  $q_j$  можно

строить эффективно, выполняя трехдиагонализацию матрицы  $A$  с помощью некоторой ортонормированной матрицы, в которой первый столбец есть  $q_1$ .

Для этой цели можно адаптировать трехдиагонализацию Хаусхолдера, обсуждавшуюся в § 8.2.1. Однако такой подход непрактичен в случае большой и разреженной матрицы  $A$ , потому что подобные преобразования Хаусхолдера имеют тенденцию к разрушению разреженности. В результате этого в ходе процесса будут возникать неприемлемо большие плотные матрицы.

Иногда потерей разреженности можно управлять, заменяя преобразования Хаусхолдера на преобразования Гивенса (см. Duff and Reid (1976)). Однако любой метод, вычисляющий матрицу  $T$  последовательно с помощью модификации матрицы  $A$ , бесполезен для большинства разреженных матриц  $A$ .

Это наводит на мысль о том, чтобы попытаться вычислять элементы матрицы  $T = Q^T A Q$  непосредственно. Полагая  $Q = [q_1, \dots, q_n]$ ,

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ \ddots & \ddots & \ddots & & \ddots \\ \vdots & \ddots & \ddots & & \beta_{n-1} \\ 0 & \cdots & \beta_{n-1} & \alpha_n \end{bmatrix},$$

и приравнивая столбцы в равенстве  $AQ = QT$ , для  $j = 1:n - 1$  находим

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}; \quad \beta_0 q_0 \equiv 0.$$

Ортонормированность векторов  $q_i$  влечет за собой  $\alpha_j = q_j^T A q_j$ . Более того, если вектор  $r_j = (A - \alpha_j I)q_j - \beta_{j-1}q_{j-1}$  ненулевой, то  $q_{j+1} = r_j/\beta_j$ , где  $\beta_j = \pm \|r_j\|_2$ . Если  $r_j = 0$ , то итерации прерываются, но (как мы увидим) остается ценная информация об инвариантном подпространстве. Упорядочивая подходящим образом приведенные выше формулы, мы получаем *итерации Ланцоша*:

```

 $r_0 = q_1; \beta_0 = 1; q_0 = 0; j = 0$ 
while ( $\beta_j \neq 0$ )
     $q_{j+1} = r_j/\beta_j; j = j + 1; \alpha_j = q_j^T A q_j;$ 
     $r_j = (A - \alpha_j I)q_j - \beta_{j-1}q_{j-1}; \beta_j = \|r_j\|_2$ 
end

```

(9.1.3)

Без потери общности можно выбирать  $\beta_j$  положительными. Векторы  $q_j$  называются *векторами Ланцоша*.

### 9.1.3. Окончание и оценки погрешностей

Итерации обрываются, прежде чем будет получена полная трехдиагонализация в том случае, когда  $q_1$  принадлежит подходящему собственному подпространству. Это одно из нескольких математических свойств метода, охватываемых следующей теоремой.

**Теорема 9.1.1.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметрична, и предположим, что вектор  $q_1 \in \mathbb{R}^n$  имеет единичную 2-норму. Тогда итерации Ланцоша (9.1.3) обрываются при  $j = m$ , где  $m = \text{rank}(K(A, q_1, n))$ . Более того, при  $j = 1:m$  мы имеем

$$AQ_j = Q_j T_j + r_j e_j^T, \quad (9.1.4)$$

зде

$$T_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{j-1} \\ 0 & \cdots & & \beta_{j-1} & \alpha_j \end{bmatrix}$$

и  $Q_j = [q_1, \dots, q_j]$  имеет ортонормированные столбцы, являющиеся базисом в  $\mathcal{K}(A, q_1, j)$ .

*Доказательство.* Проведем индукцию по  $j$ . Предположим, что итерации дали матрицу  $Q_j = [q_1, \dots, q_j]$ , такую, что  $\text{range}(Q_j) = \mathcal{K}(A, q_1, j)$  и  $Q_j^T Q_j = I_j$ . Легко видеть, что (9.1.3) влечет за собой (9.1.4). Итак,  $Q_j^T A Q_j^j = T_j + Q_j^T r_j e_j^T$ . Вследствие того что  $a_i = q_j^T A q_i$  при  $i = 1:j$

$$q_{i+1}^T A q_i = q_{i+1}^T (A q_i - a_i q_i - \beta_{i-1} q_{i-1}) = q_{i+1}^T (\beta_i q_{i+1}) = \beta_i$$

при  $i = 1:j-1$ , мы имеем  $Q_j^T A Q_j = T_j$ . Следовательно,  $Q_j^T r_j = 0$ .

Если  $r_j \neq 0$ , то вектор  $q_{j+1} = r_j / \|r_j\|_2$  ортогонален к  $q_1, \dots, q_j$  и

$$q_{j+1} \in \text{span}\{A q_j, q_j, q_{j-1}\} \subseteq \mathcal{K}(A, q_1, j+1).$$

Таким образом,  $Q_{j+1}^T Q_{j+1} = I_{j+1}$  и  $\text{range}(Q_{j+1}) = \mathcal{K}(A, q_1, j+1)$ . С другой стороны, если  $r_j = 0$ , то  $A Q_j = Q_j T_j$ . Это говорит об инвариантности  $\text{range}(Q_j) = \mathcal{K}(A, q_1, j)$ . Отсюда получаем, что  $j = m = \text{rank}(K(A, q_1, n))$ .  $\square$

Получение нулевого  $\beta_j$  в процессе Ланцоша – событие, желательное в том смысле, что оно свидетельствует о том, что найдено некоторое подпространство, являющееся в точности инвариантным. Однако на практике точный нуль или даже малое  $\beta_j$  встречаются редко. Тем не менее крайние собственные значения для  $T_j$  оказываются удивительно хорошими аппроксимациями к крайним собственным значениям для  $A$ . Поэтому для сходимости собственных значений нужно искать иное объяснение. Следующий результат представляет собой некоторый шаг в этом направлении.

**Теорема 9.1.2.** Предположим, что выполнены  $j$  шагов алгоритма Ланцоша и  $S_j^T T_j S_j = \text{diag}(\theta_1, \dots, \theta_j)$  есть разложение Шура трехдиагональной матрицы  $T_j$ . Если  $Y_j = [y_1, \dots, y_j] = Q_j S_j \in \mathbb{R}^{j \times j}$ , то для  $i = 1:j$  имеем  $\|A y_i - \theta_i y_i\|_2 = |\beta_i| |s_{ji}|$ , где  $S_j = (s_{pq})$ .

*Доказательство.* Умножая (9.1.4) справа на  $S_j$ , получаем

$$AY_j = Y_j \text{diag}(\theta_1, \dots, \theta_j) + r_j e_j^T S_j,$$

так что  $A y_i = \theta_i y_i + r_j (e_j^T S_j)$ . Чтобы завершить доказательство, остается перейти к нормам и вспомнить, что  $\|r_j\|_2 = |\beta_j|$ .  $\square$

Теорема позволяет вычислить оценки погрешностей для собственных значений матрицы  $T_j$ :

$$\min_{\mu \in \lambda(A)} |\theta_i - \mu| \leq |\beta_j| |s_{ji}|, \quad i = 1:j.$$

Заметим, что в терминологии теоремы 8.1.10  $(\theta_i, y_i)$  суть пары Ритца для подпространства  $\text{range}(Q_j)$ .

Другой подход к использованию  $T_j$  для получения оценок собственных значений матрицы  $A$  описан в [Golub (1974)]. Предлагается специальным образом построить некоторую одноранговую матрицу  $E$ , такую, что подпространство  $\text{range}(Q_j)$  инвариантно относительно  $A + E$ . Более точно, если в соответствии с методом Ланцоша  $AQ_j = Q_j T_j + r_j e_j^T$ , то, положив  $E = \tau w w^T$ , где  $\tau = \pm 1$  и  $w = aq_j + br_j$ , можно показать, что

$$(A + E)Q_j = Q_j(T_j + \tau a^2 e_j e_j^T) + (1 + \tau ab)r_j e_j^T.$$

Если  $0 = 1 + \tau ab$ , то собственные значения матрицы  $T_j = T_j + \tau a^2 e_j e_j^T$  (трехдиагональной) являются также собственными значениями для  $A + E$ . Тогда из теоремы 8.1.5 мы можем вывести, что каждый из интервалов<sup>1)</sup>  $[\lambda_i(\tilde{T}_j), \lambda_{i-1}(\tilde{T}_j)]$ ,  $i = 2:n$ , содержит какое-то собственное значение матрицы  $A$ .

Эти интервалы локализации зависят от выбора  $\tau a^2$ . Предположим, что у нас имеется какое-то приближение к собственному значению  $\lambda$  матрицы  $A$ . Одна из возможностей – выбирать  $\tau a^2$  таким образом, чтобы  $\det(T_j - \lambda T_j) = (a_j + \tau a^2 - \lambda)p_{j-1}(\lambda) - \beta_{j-1}^2 p_{j-2}(\lambda) = 0$ , где значения полиномов  $p_i(x) = \det(T_i - x T_i)$  в точке  $\lambda$  можно оценить, используя трехчленные рекуррентные соотношения (8.5.2). Здесь предполагается, что  $p_{j-1}(\lambda) \neq 0$ . Оценивание собственных значений в таком духе обсуждается в Lehmann (1963); Householder (1968).

#### 9.1.4. Теория сходимости Каниэля–Пейджа

Предыдущее обсуждение показывает, каким образом с помощью алгоритма Ланцоша можно получать оценки собственных значений, но оно не дает никакой информации о скорости сходимости. Результаты такого рода составляют то, что называется теорией Каниэля–Пейджа. Ниже приводится некоторое извлечение из нее.

**Теорема 9.1.3.** Пусть  $A$  – симметричная  $n \times n$ -матрица с собственными значениями  $\lambda_1 \geq \dots \geq \lambda_n$  и соответствующими ортонормированными собственными векторами  $z_1, \dots, z_n$ . Если  $\theta_1 \geq \dots \geq \theta_j$  суть собственные значения матрицы  $T_j$ , полученной в результате  $j$  шагов итераций Ланцоша, то

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - \frac{(\lambda_1 - \lambda_n) \operatorname{tg}(\varphi_1)^2}{(c_{j-1}(1 + 2\rho_1))^2},$$

где  $\cos(\varphi_1) = |q_1^T z_1|$ ,  $\rho_1 = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$ ,  $c_{j-1}(x)$  – полином Чебышева степени  $j-1$ .

*Доказательство.* По теореме 8.1.2 имеем

$$\theta_1 = \max_{y \neq 0} \frac{y^T T_j y}{y^T y} = \max_{y \neq 0} \frac{(Q_j y)^T A (Q_j y)}{(Q_j y)^T (Q_j y)} = \max_{0 \neq w \in \mathcal{K}(A, q_1, j)} \frac{w^T A w}{w^T w}.$$

Поскольку  $\lambda_1$  есть максимум  $w^T A w / w^T w$  по всем ненулевым  $w$ , отсюда следует, что  $\lambda_1 \geq \theta_1$ . Чтобы получить нижнюю оценку для  $\theta_1$ , заметим, что

$$\theta_1 = \max_{p \in P_{j-1}} \frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1},$$

<sup>1)</sup> Это верно в случае  $\tau = 1$ . Если  $\tau = -1$ , то в соответствии с теоремой 8.1.5 в качестве локализации следует взять  $[\lambda_{i+1}(\tilde{T}_j), \lambda_i(\tilde{T}_j)]$ ,  $i = 1:n-1$ . – Прим. перев.

где  $P_{j-1}$  – множество полиномов степени  $j-1$ . Если  $q_1 = \sum_{i=1}^n d_i z_i$ , то

$$\frac{q_1^T p(A) A p(A) q_1}{q_1^T p(A)^2 q_1} = \frac{\sum_{i=1}^n d_i^2 p(\lambda_i)^2 \lambda_i}{\sum_{i=1}^n d_i^2 p(\lambda_i)^2} \geq \lambda_1 - (\lambda_1 - \lambda_n) \frac{\sum_{i=2}^n d_i^2 p(\lambda_i)^2}{d_1^2 p(\lambda_1)^2 + \sum_{i=2}^n d_i^2 p(\lambda_i)^2}.$$

Нижнюю оценку можно сделать весьма точной, выбирая полином  $p(x)$  таким образом, чтобы его значение при  $x = \lambda_1$  было большим по сравнению со значениями при остальных собственных значениях. Один из способов это сделать состоит в том, чтобы взять

$$p(x) = c_{j-1} \left( -1 + 2 \frac{x - \lambda_n}{\lambda_2 - \lambda_n} \right),$$

где  $c_{j-1}(z)$  есть  $(j-1)$ -й полином Чебышева, порождаемый рекурсией

$$c_j(z) = 2zc_{j-1}(z) - c_{j-2}(z), \quad c_0 = 1, \quad c_1 = z.$$

Эти полиномы ограничены единицей на  $[-1, 1]$ , но очень быстро растут вне этого интервала. Если определить  $p(x)$  таким образом, то как следствие  $|p(\lambda_i)|$  будет ограничено единицей при  $i = 2:n$ , в то время как  $p(\lambda_1) = c_{j-1}(1 + 2\rho_1)$ . Итак,

$$\theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n) \frac{1 - d_1^2}{d_1^2} \frac{1}{c_{j-1}(1 + 2\rho_1)^2}.$$

Чтобы получить требуемую нижнюю оценку, нужно заметить, что  $\operatorname{tg}(\varphi_1)^2 = (1 - d_1^2)/d_1^2$ .  $\square$

Из данной теоремы немедленно вытекает аналогичный результат, относящийся к  $\theta_j$ :

**Следствие 9.1.4.** В обозначениях теоремы имеем

$$\lambda_n \leq \theta_j \leq \lambda_n + \frac{(\lambda_1 - \lambda_n) \operatorname{tg}(\varphi_n)^2}{c_{j-1}(1 + 2\rho_n)^2},$$

где  $\rho_n = (\lambda_{n-1} - \lambda_n)/(\lambda_1 - \lambda_{n-1})$  и  $\cos(\theta_n) = q_n^T z_n$ .

*Доказательство.* Применяем теорему 9.1.3 с заменой  $A$  на  $-A$ .  $\square$

### 9.1.5. Сравнение степенного метода с методом Ланцоша

Есть смысл сравнить  $\theta_1$  с оценкой для  $\lambda_1$ , отвечающей степенному методу (см. § 7.3.1). Для ясности предположим, что  $\lambda_1 \geq \dots \geq \lambda_n \geq 0$ . После того как к  $q_1$  будут применены  $j-1$  шагов степенного метода, мы получим некоторый вектор в направлении

$$v = A^{j-1} q_1 = \sum_{i=1}^n c_i \lambda_i^{j-1} z_i$$

вместе с оценкой собственного значения

$$\gamma_1 = \frac{v^T A v}{v^T v}.$$

Используя доказательство и обозначения теоремы 9.1.3, легко показать, что

$$\lambda_1 \geq \gamma_1 \geq \lambda_1 - (\lambda_1 - \lambda_n) \operatorname{tg}(\varphi_1)^2 \left( \frac{\lambda_2}{\lambda_1} \right)^{2j-1}. \quad (9.1.5)$$

(Указание: положить в доказательстве  $p(x) = x^{j-1}$ .) Таким образом, мы можем сравнить точность нижних оценок для  $\theta_1$  и  $\gamma_1$ , сопоставляя

$$L_{j-1} \equiv 1 / \left[ c_{j-1} \left( 2 \frac{\lambda_1}{\lambda_n} - 1 \right) \right]^2 \geq 1 / [c_{j-1} (1 + 2 \rho_1)]^2$$

и

$$R_{j-1} = \left( \frac{\lambda_2}{\lambda_1} \right)^{2(j-1)}.$$

Для выборочных значений  $j$  и  $\lambda_2/\lambda_1$  это сделано в следующей таблице.

Таблица 9.1.1.  $L_{j-1}/R_{j-1}$

$\lambda_1/\lambda_2$	$j = 5$	$j = 10$	$j = 15$	$j = 20$	$j = 25$
1.50	$\frac{1.1 \times 10^{-4}}{3.9 \times 10^{-2}}$	$\frac{2.0 \times 10^{-10}}{6.8 \times 10^{-4}}$	$\frac{3.9 \times 10^{-16}}{1.2 \times 10^{-5}}$	$\frac{7.4 \times 10^{-22}}{2.0 \times 10^{-7}}$	$\frac{1.4 \times 10^{-27}}{3.5 \times 10^{-9}}$
1.10	$\frac{2.7 \times 10^{-2}}{4.7 \times 10^{-1}}$	$\frac{5.5 \times 10^{-5}}{1.8 \times 10^{-1}}$	$\frac{1.1 \times 10^{-7}}{6.9 \times 10^{-2}}$	$\frac{2.1 \times 10^{-10}}{2.7 \times 10^{-2}}$	$\frac{4.2 \times 10^{-13}}{1.0 \times 10^{-2}}$
1.01	$\frac{5.6 \times 10^{-1}}{9.2 \times 10^{-1}}$	$\frac{1.0 \times 10^{-1}}{8.4 \times 10^{-1}}$	$\frac{1.5 \times 10^{-2}}{7.6 \times 10^{-1}}$	$\frac{2.0 \times 10^{-3}}{6.9 \times 10^{-1}}$	$\frac{2.8 \times 10^{-4}}{6.2 \times 10^{-1}}$

Превосходство метода Ланцюша очевидно. Этому и не следует удивляться, поскольку  $\theta_1$  есть максимум  $r(x) = x^T A x / x^T x$  на всем  $\mathcal{K}(A, q_1, j)$ , в то время как  $\gamma_1 = r(v)$  – значение для некоторого специального  $v$  из  $\mathcal{K}(A, q_1, j)$ , а именно для  $v = A^{j-1} q_1$ .

### 9.1.6. Сходимость внутренних собственных значений

Мы закончим некоторыми замечаниями по поводу оценок погрешностей для внутренних собственных значений матриц  $T_j$ . Ключевая идея в доказательстве теоремы 9.1.3 заключается в использовании сдвинутых чебышевских полиномов. С помощью такого полинома мы усиливаем компоненту вектора  $q_1$  в направлении  $z_1$ . Чтобы получить оценки для внутреннего числа Ритца  $\theta_1$ , можно использовать схожую идею. Однако оценки не будут столь же хорошими, потому что «усиливающий полином» имеет вид  $q(x) \prod_{k=1}^{j-1} (x - \lambda_k)$ , где  $q(x)$  – полином Чебышева степени  $j-1$  на интервале  $[\lambda_{i+1}, \lambda_n]$ . Подробности см. в Kaniel (1966), Paige (1971), Scott (1978).

#### Задачи

**9.1.1.** Предположим, что матрица  $A \in \mathbf{R}^{n \times n}$  кососимметрична. Вывести алгоритм типа Ланцюша, вычисляющий кососимметричную трехдиагональную матрицу  $T_m$ , такую, что  $AQ_m = Q_m T_m$ , где  $Q_m^T Q_m = I_m$ .

**9.1.2.** Пусть матрица  $A \in \mathbf{R}^{n \times n}$  симметрична и  $r(x) = x^T A x / x^T x$ . Предположим, что подпространство  $S \subseteq \mathbf{R}^n$  обладает тем свойством, что  $x \in S$  влечет за собой  $\nabla r(x) \in S$ . Показать, что  $S$  инвариантно относительно  $A$ .

**9.1.3.** Показать, что если симметричная матрица  $A \in \mathbb{R}^{n \times n}$  имеет кратное собственное значение, то итерации Ланцоша заканчиваются до срока.

**9.1.4.** Показать, что в теореме 9.1.1 индекс  $m$  есть размерность наименьшего  $A$ -инвариантного подпространства, содержащего  $q_1$ .

**9.1.5.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметрична и рассмотрим проблему определения ортонормированной последовательности  $q_1, q_2, \dots$ , обладающей тем свойством, что если матрица  $Q_j = [q_1, \dots, q_j]$  известна, то  $q_{j+1}$  выбирается так, чтобы минимизировать  $\mu_j = \| (I - Q_{j+1} Q_{j+1}^T) A Q_j \|_F$ . Показать, что если  $\text{span}\{q_1, \dots, q_j\} = \mathcal{K}(A, q_1, j)$ , то  $q_{j+1}$  можно выбрать таким образом, что  $\mu_j = 0$ . Объяснить, как эта оптимизационная задача приводит к алгоритму Ланцоша.

### Замечания и литература к § 9.1

К хорошим «глобальным» ссылкам на алгоритм Ланцоша относится Parlett (SEP) (1980b) и Cullum J. K. and Willoughby R. A. (1985a). Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1. Theory, Birkhäuser, Boston.

Классической ссылкой на метод Ланцоша является

Lanczos C. (1950). “An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators”, J. Res. Nat. Bur. Stand. 45, 255–82.

Хотя в этой работе и упоминается о сходимости чисел Ритца, дальнейшие детали читатель найдет в

Kaniel S. (1966). “Estimates for Some Computational Techniques in Linear Algebra”, Math. Comp. 20, 369–78.

Paige C. C. (1971). “The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices”, Ph. D. thesis, London University.

Изложенная в этих работах теория Каниэля – Пейджа прекрасно суммирована в

Scott D. (1978). “Analysis of the Symmetric Lanczos Process”, Electronic Research Laboratory Technical Report UCB/ERL M78/40, University of California, Berkely.

См. также Wilkinson (1965a), Parlett (1980b) и

Saad Y. (1980). “On the Rates of Convergence of the Lanczos and the Block Lanczos Methods”, SIAM J. Num. Anal. 17, 687–706.

Связь между алгоритмом Ланцоша, ортогональными полиномами и теорией моментов обсуждается в

Golub G. H. (1974). “Some Uses of the Lanczos Algorithm in Numerical Linear Algebra”, in Topics in Numerical Analysis, ed., J. J. H. Miller, Academic Press, New York.

Householder A. S. (1968). “Moments and characteristic Roots II,” Numer. Math. 11, 126–28.

Lehmann N. J. (1963). “Optimale Eigenwert einschließungen”, Numer. math. 5, 246–72.

Наше изложение метода Ланцоша мотивировалось замечаниями о неизбежности заполнения при выполнении трехдиагонализации с помощью преобразований Хаусхолдера или Гивенса. В действительности, соблюдая меры предосторожности, иногда можно удержать заполнение на некотором приемлемом уровне. См.

Duff I. S. (1974). “Pivot Selection and Row Ordering in Givens Reduction on Sparse Matrices”, Computing 13, 239–48.

Duff I. S. and Reid J. K. (1976). “A Comparison of Some Methods for the Solution of Sparse Over-Determined Systems of Linear Equations”, J. Inst. Maths. Applic. 17, 267–80.

Kaufman L. (1979). “Application of Dense Householder Transformations to a Sparse Matrix”, ACM Trans. Math. Soft. 5, 442–50.

## 9.2. Практические процедуры Ланцоша

На поведение итераций Ланцоша сильнейшее воздействие оказывают ошибки округления. Основная трудность вызвана потерей ортогональности среди векторов

Ланцоша. Это явление затуманивает вопрос об окончании процесса и усложняет соотношение между собственными значениями матрицы  $A$  и трехдиагональных матриц  $T_j$ . Трудности, навеянные этим явлением, вместе с созданием совершенно устойчивого метода хаусхолдеровой трехдиагонализации, объясняют, почему численные аналитики не обращали внимания на алгоритм Ланцоша в течение 50-х и 60-х годов. Однако интерес к методу оживили как теория Каниэля–Пейджа, так и необходимость решать большие разреженные спектральные задачи, размер которых увеличивался вместе с производительностью компьютеров. Поскольку для получения хороших приближений к экстремальным собственным значениям обычно требуется намного меньше чем  $n$  итераций, метод Ланцоша привлекает скорее не как соперник подхода Хаусхольдера, а как средство для работы с разреженными матрицами.

Успешные реализации метода Ланцоша включают в себя гораздо больше, чем только закодированные соотношения (9.1.3). В этом разделе мы дадим представление о некоторых практических идеях, предложенных для того, чтобы сделать процедуру Ланцоша работоспособной.

### 9.2.1. Реализация в точной арифметике

Весь процесс Ланцоша можно выполнить, сохраняя лишь два  $n$ -вектора. Для этого понадобится формула

$$a_j = q_j^T (Aq_j - \beta_{j-1} q_{j-1})$$

и тщательная организация замещения в (9.1.3).

**Алгоритм 9.2.1 (Алгоритм Ланцоша).** Для заданных симметричной матрицы  $A \in \mathbb{R}^{n \times n}$  и вектора  $w \in \mathbb{R}^n$  с единичной 2-нормой следующий алгоритм вычисляет симметричную трехдиагональную  $j \times j$ -матрицу  $T_j$ , такую, что  $\lambda(T_j) \subset \lambda(A)$ . Предполагается, что есть функция  $A.\text{mult}(w)$ , возвращающая матрично-векторное произведение  $Aw$ . Диагональные и поддиагональные элементы матрицы  $T_j$  хранятся соответственно в  $\alpha(1:j)$  и  $\beta(1:j-1)$ .

```

 $v(1:n) = 0; \beta_0 = 1; j = 0$ 
while  $\beta_j \neq 0$ 
    if  $j \neq 0$ 
        for  $i = 1:n$ 
             $t = w_i; w_i = v_i/\beta_j; v_i = -\beta_j t$ 
        end
    end
     $v = v + A.\text{mult}(w)$ 
     $j = j + 1; \alpha_j = w^T v; v = v - \alpha_j w; \beta_j = \|v\|_2$ 
end
```

Заметим, что  $A$  не изменяется в течение всего процесса. Нам нужна лишь процедура  $A.\text{mult}()$ , вычисляющая матрично-векторные произведения с участием  $A$ . Если  $A$  имеет в среднем по  $k$  ненулевых элементов на строку, то один шаг Ланцоша требует примерно  $(2k+8)n$  флопов.

По окончании процесса собственные значения для  $T_j$  могут быть найдены с помощью симметричного трехдиагонального QR-алгоритма или любого из специальных методов из § 8.4, например биссекций.

Векторы Ланцоша получаются в  $n$ -векторе  $w$ . Если предполагается как-то их использовать, то нужны специальные меры по их сохранению. Для типичных

разреженных задач до момента использования их можно хранить на диске или в каком-либо другом вспомогательном запоминающем устройстве.

### 9.2.2. Ошибки округления

Разработка практической, простой в реализации процедуры Ланцоша требует понимания фундаментального анализа ошибок округления, выполненного Пейджем [Paige (1971, 1976a, 1980)]. Осмысление этих результатов дает наилучшую мотивацию для рассматриваемых в этом разделе нескольких модифицированных процедур Ланцоша.

После  $j$  шагов алгоритма мы получаем матрицу вычисленных векторов Ланцоша  $\hat{Q}_j = [\hat{q}_1, \dots, \hat{q}_j]$  и соответствующую трехдиагональную матрицу

$$\hat{T}_j = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & \cdots & 0 \\ \hat{\beta}_1 & \hat{\alpha}_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \hat{\beta}_{j-1} \\ 0 & \cdots & & \hat{\beta}_{j-1} & \hat{\alpha}_j \end{bmatrix}$$

Пейдж (1971, 1976a) показывает, что если вместо  $r_j$  при вычислении получается  $\hat{r}_j$ , то

$$A\hat{Q}_j = \hat{Q}_j \hat{T}_j + \hat{r}_j e_j^T + E_j, \quad (9.2.1)$$

где

$$\|E_j\|_2 \approx u \|A\|_2. \quad (9.2.2)$$

Это означает, что важное уравнение  $AQ_j = Q_j T_j + r_j e_j^T$  удовлетворяется с рабочей точностью.

К несчастью, с ортогональностью векторов  $\hat{q}_j$  дело обстоит намного хуже. (Речь идет не о нормированности, так как вычисленные векторы Ланцоша имеют по существу единичную длину.) Если  $\hat{\beta}_j = f(l(\|\hat{r}_j\|_2))$  и мы вычисляем  $\hat{q}_{j+1} = f(l(\hat{r}_j/\hat{\beta}_j))$ , то простой анализ показывает, что  $\hat{\beta}_j \hat{q}_{j+1} \approx \hat{r}_j + w_j$ , где  $\|w_j\|_2 \approx u \|\hat{r}_j\|_2 \approx u \|A\|_2$ . Таким образом, мы приходим к выводу о том, что

$$|\hat{q}_{j+1}^T \hat{q}_i| \approx \frac{|\hat{r}_j^T \hat{q}_i| + u \|A\|_2}{|\hat{\beta}_j|}$$

для  $i = 1:j$ . Другими словами, при малом  $\hat{\beta}_j$  можно ожидать значительных отклонений от ортогональности даже в идеальной ситуации, когда  $\hat{r}_j^T \hat{q}_j$  есть нуль. Малое  $\hat{\beta}_j$  влечет за собой потерю точности при вычислении вектора  $\hat{r}_j$ . Мы подчеркиваем, что потеря ортогональности происходит благодаря этой потере точности и не является результатом постепенного накапливания ошибок округления.

**Пример 9.2.1.** Матрица

$$A = \begin{bmatrix} 2.64 & -0.48 \\ -0.48 & 2.36 \end{bmatrix}$$

имеет собственные значения  $\lambda_1 = 3$  и  $\lambda_2 = 2$ . Если алгоритм Ланцоша для этой матрицы с начальным вектором  $q_1 = [0.810, -0.586]^T$  выполняется в плавающей трехзначной арифметике, то  $\hat{q}_2 = [0.707, 0.707]^T$ . Потеря ортогональности происходит из-за того, что подпространство  $\text{span}\{q_1\}$  почти инвариантно для  $A$ . (Вектор  $x = (0.8, -0.6)^T$  является собственным вектором, отвечающим  $\lambda_1$ .)

Вскоре будут даны и дальнейшие детали анализа Пейджа. Сейчас нам достаточно заметить, что на практике потеря ортогональности происходит всегда, а вместе с ней и очевидное ухудшение качества собственных значений матрицы  $\hat{T}_j$ . Количественную оценку этого ухудшения можно получить, объединив (9.2.1) с теоремой 8.1.11. В частности, если в этой теореме взять  $F_1 = \hat{r}_j e_j^T + E_j$ ,  $X_1 = \hat{Q}_j$ ,  $S = \hat{T}_j$  и  $\tau = \|\hat{Q}_j^T \hat{Q}_j - I_j\|_2$ , то можно утверждать, что существуют собственные значения  $\mu_1, \dots, \mu_j \in \lambda(A)$ , такие, что

$$|\mu_i - \lambda_i(T_j)| \leq \sqrt{2} \frac{\|\hat{r}_j\|_2 + \|E_j\|_2}{\sigma_j(Q_j)} + \tau \|A\|_2$$

для  $i = 1:j$ . Очевидное действие с целью помешать знаменателю в верхней оценке приблизиться к нулю заключается в проведении ортогонализации каждого вновь вычисляемого вектора Ланцоша ко всем его предшественникам. Это сразу же ведет к нашей первой «практической» процедуре Ланцоша.

### 9.2.3. Метод Ланцоша с полной переортогонализацией

Пусть заданы  $r_0, \dots, r_{j-1} \in \mathbb{R}^n$ , и предположим, что вычислены матрицы Хаусхолдера  $H_0, \dots, H_{j-1}$ , такие, что матрица  $(H_0 \dots H_{j-1})^T [r_0, \dots, r_{j-1}]$  является верхней треугольной. Пусть  $[q_1, \dots, q_j]$  обозначает первые  $j$  столбцов хаусхолдерова произведения  $(H_0 \dots H_{j-1})$ . Теперь предположим, что задан вектор  $r_j \in \mathbb{R}^n$  и мы хотим вычислить нормированный вектор  $q_{j+1}$ , направленный так же, как

$$w = r_j - \sum_{i=1}^j (q_j^T r_j) q_i \in \text{span}\{q_1, \dots, q_j\}^T.$$

Если хаусхолдерова матрица  $H_j$  обеспечивает треугольный вид матрицы  $(H_0 \dots H_j)^T [r_0, \dots, r_j]$ , то, следовательно, искомый вектор – это  $(j+1)$ -й столбец в  $H_0 \dots H_j$ .

Включив эти хаусхолдеровы вычисления в процедуру Ланцоша, мы можем получить векторы Ланцоша, являющиеся ортогональными с рабочей точностью:

$r_0 = q_1$  (заданный единичный вектор)

Найти хаусхолдерову матрицу  $H_0$  с тем, чтобы  $H_0 r_0 = e_1$ .

$\alpha_1 = q_1^T A q_1$

for  $j = 1:n-1$

$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}$  ( $\beta_0 q_0 = 0$ )

$w = (H_{j-1} \dots H_0) r_j$  (9.2.3)

Найти хаусхолдерову матрицу  $H_j$  с тем, чтобы  $H_j w = (w_1, \dots, w_j, \beta_j, 0, \dots, 0)^T$ .

$q_{j+1} = H_0 \dots H_j e_{j+1}$ ;  $\alpha_{j+1} = q_{j+1}^T A q_{j+1}$

end

Это пример схемы Ланцоша с полной переортогонализацией. Тщательный анализ можно найти в работе Paige (1970). Идея использования матриц Хаусхолдера для усиления ортогональности появилась в работах Golub, Underwood, Wilkinson (1972).

То что в (9.2.3) вычисленные векторы  $\hat{q}_i$  ортогональны с рабочей точностью, следует из свойств ошибок округления при работе с матрицами Хаусхолдера. Заметим, что в силу способа определения  $q_{i+1}$  ничего не меняется, если  $\beta_j = 0$ . Поэтому алгоритм может без помех выполняться до  $j = n-1$ . (Однако на практике можно остановиться при намного меньшем значении  $j$ .)

В любой реализации процесса (9.2.3), конечно, сохраняются хаусхолдеровы векторы  $v_j$ , а соответствующие  $P_j$  никогда не формируются в явном виде. Поскольку  $H_j(1:j, 1:j) = I_j$ , нет необходимости в вычислении первых  $j$  компонент вектора

$w = (H_{j-1} \dots H_0)r_j$ , так как в точной арифметике эти компоненты были бы нулевыми.

Эти возможности экономии, к несчастью, дают очень мало для снижения вычислительных затрат, связанных с полной переортогонализацией. Хаусхолдеровы вычисления увеличивают работу на  $j$ -м шаге Ланцоша на  $O(jn)$  флопов. Более того, для того чтобы вычислить  $q_{j+1}$ , нужно иметь доступ к хаусхолдеровым векторам, связанным с  $H_0, \dots, H_j$ . Для больших  $n$  и  $j$  это обычно влечет за собой неприемлемо большое число пересылок данных.

Таким образом, полная переортогонализация стоит очень дорого. К счастью, есть и более эффективные средства, но они требуют от нас более внимательного анализа того, как теряется ортогональность.

#### 9.2.4. Выборочная ортогонализация

Замечательное ироническое следствие из анализа ошибок округления в работе Paige (1971) состоит в том, что потеря ортогональности неразрывно связана со сходимостью некоторой пары Ритца. Более того, предположим, что симметричный QR-алгоритм, примененный к  $\hat{T}_j$ , дает вычисленные значения Ритца  $\hat{\theta}_1, \dots, \hat{\theta}_j$  и почти ортогональную матрицу собственных векторов  $\hat{S}_j = (\hat{S}_{pq})$ . Если  $\hat{Y}_j = [\hat{y}_1, \dots, \hat{y}_j] = f^T(\hat{Q}_j \hat{S}_j)$ , то можно показать, что для  $i = 1:j$  мы имеем

$$|\hat{q}_{j+1}^T \hat{y}_i| \approx \frac{\mathbf{u} \| A \|_2}{|\hat{\beta}_j| |\hat{s}_{ji}|} \quad (9.2.4)$$

и

$$\| A \hat{y}_i - \hat{\theta}_i \hat{y}_i \|_2 \approx |\hat{\beta}_j| |\hat{s}_{ji}|. \quad (9.2.5)$$

Как видим, поведение последнего из вычисленных векторов Ланцоша  $\hat{q}_{j+1}$  таково, что он имеет нетривиальную и нежелательную компоненту в направлении любого из сошедшихся векторов Ритца. Следовательно, вместо того чтобы ортогонализировать  $\hat{q}_{j+1}$  по отношению ко всем ранее вычисленным векторам Ланцоша, мы можем добиться того же эффекта, проводя ортогонализацию по отношению к намного меньшему множеству сошедшихся векторов Ритца.

Практические аспекты указанного способа усиления ортогональности обсуждаются в работах Parlett и Scott (1979). В предложенной там схеме, известной под названием выборочной ортогонализации, вычисленная пара Ритца  $(\hat{\theta}, \hat{y})$  называется «хорошей», если она удовлетворяет соотношению

$$\| A \hat{y} - \hat{\theta} \hat{y} \|_2 \approx \sqrt{\mathbf{u}} \| A \|_2.$$

Как только вектор  $\hat{q}_{j+1}$  вычислен, он ортонормируется по отношению ко всем хорошим векторам Ритца. Это стоит намного меньше, чем полная переортогонализация, поскольку хороших векторов Ритца обычно намного меньше, чем векторов Ланцоша.

Один из способов реализации выборочной ортогонализации состоит в том, что на каждом шаге диагонализируется  $\hat{T}_j$  и затем оцениваются  $\hat{s}_{ji}$  в свете (9.2.4) и (9.2.5). Гораздо более эффективный подход – оценивать меру потери ортогональности  $\| I_j - \hat{Q}_j^T \hat{Q}_j \|_2$ , используя следующий результат:

**Лемма 9.2.1.** Предположим, что  $S_+ = [S \ d]$ , где  $S \in \mathbf{R}^{n \times j}$  и  $d \in \mathbf{R}^n$ . Если  $S$  удовлетворяет  $\| I_j - S^T S \|_2 \leq \mu$  и  $|1 - d^T d| \leq \delta$ , то  $\| I_{j+1} - S_+^T S_+ \|_2 \leq \mu_+$ , где

$$\mu_+ = \frac{1}{2}(\mu + \delta + \sqrt{(\mu + \delta)^2 + 4 \| S^T d \|_2^2}).$$

**Доказательство.** См. Kahan, Parlett (1974) или Parlett, Scott (1979). Итак, имея оценку для  $\|I_j - \hat{Q}_j^T Q_j\|_2$ , мы можем получить оценку для  $\|I_{j+1} - \hat{Q}_{j+1}^T Q_{j+1}\|_2$ , применяя лемму с  $S = \hat{Q}_j$  и  $d = \hat{q}_{j+1}$ . (В этом случае  $\delta \approx 0$  и мы предполагаем, что  $\hat{q}_{j+1}$  был ортогонализирован по отношению к текущему набору хороших векторов Ритца.) Есть возможность оценить норму вектора  $\hat{Q}_j^T \hat{q}_{j+1}$  с помощью простого рекуррентного соотношения, избавляющего от необходимости доступа к  $\hat{q}_1, \dots, \hat{q}_j$ . См. Kahan, Parlett (1974) или Parlett, Scott (1979). Накладные расходы минимальны, и лишь когда оценки дают сигнал о потере ортогональности, наступает тот момент, когда ожидается пополнение множества хороших векторов Ритца. В эти и только эти моменты проводится диагонализация матрицы  $\hat{T}_j$ .

### 9.2.5. Проблема теневых собственных значений

Значительные усилия были направлены на разработку работоспособной процедуры Ланцюша, вообще не использующей какое-либо усиление ортогональности. Исследования в этом направлении сосредоточены на проблеме «теневых» или «ложных» собственных значений. Это кратные собственные значения для  $\hat{T}_j$ , отвечающие простым собственным значениям для  $A$ . Они возникают из-за того, что итерации начинаются по существу заново, когда теряется ортогональность к какому-либо сошедшемуся вектору Ритца. (По аналогии рассмотрите, что может случиться в процессе ортогональных итераций из § 7.3.2, если мы «забудем» ортогонализировать.)

Как обнаружить теневые собственные значения и что делать, если они есть, эта проблема обсуждается в Cullum, Willoughby (1979) и Parlett, Reid (1981). Эта проблема особенно актуальна в тех приложениях, где для  $A$  требуется найти все собственные значения, – в этих случаях описанные выше процедуры ортогонализации требуют чрезмерных вычислительных затрат.

Трудности с итерациями Ланцюша могут ожидаться даже в том случае, когда  $A$  имеет по-настоящему кратное собственное значение. Это вытекает из того, что матрицы  $\hat{T}_j$  неразложимы, а неразложимые трехдиагональные матрицы не могут иметь кратных собственных значений (см. теорему 7.4.3). Наша следующая практическая процедура Ланцюша пытается преодолеть эту трудность.

### 9.2.6. Блочный Ланцюш

Точно так же, как простой степенной метод имеет своим блочным аналогом одновременные итерации, так и алгоритм Ланцюша имеет блочную версию. Предположим, что  $n = rp$ , и рассмотрим разложение

$$Q^T A Q = \bar{T} = \begin{bmatrix} M_1 & B_1^T & & \cdots & & 0 \\ B_1 & M_2 & \ddots & & & \vdots \\ \ddots & \ddots & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & & B_{r-1}^T \\ 0 & \cdots & & B_{r-1} & & M_r \end{bmatrix},$$

где матрица

$$Q = [X_1, \dots, X_r], \quad X_i \in \mathbb{R}^{n \times p},$$

ортогональная,  $M_i \in \mathbb{R}^{p \times p}$  для всех  $i$ , все матрицы  $B_i \in \mathbb{R}^{p \times p}$  верхние треугольные. Приравнивание блоков в  $AQ = Q\bar{T}$  показывает, что

$$AX_j = X_{j-1}B_{j-1}^T + X_jM_j + X_jB_j, \quad X_0B_0 \equiv 0,$$

для  $j = 1:r - 1$ . В силу ортогональности  $Q$  мы имеем  $M_j = X_j^TAX_j$  для  $j = 1:r$ . Более того, если мы определим

$$R_j = AX_j - X_jM_j - X_{j-1}B_{j-1}^T \in \mathbb{R}^{n \times p},$$

то  $X_{j+1}B_j = R_j$  представляет собой QR-разложение для  $R_j$ . Эти наблюдения наводят на мысль о том, что блоchно трехдиагональная матрица  $\bar{T}$  может быть получена следующим образом:

$X_1 \in \mathbb{R}^{p \times p}$  – заданная матрица, такая, что  $X_1^T X_1 = I_p$ .

$$M_1 = X_1^T AX_1$$

for  $j = 1:r - 1$

$$R_j = AX_j - X_jM_j - X_{j-1}B_{j-1}^T (X_0B_0^T \equiv 0) \quad (9.2.7)$$

$X_{j+1}B_j = R_j$  (QR-разложение для  $R_j$ )

$$M_{j+1} = X_{j+1}^T AX_{j+1}$$

end

В начале  $j$ -го прохода цикла мы имеем

$$A[X_1, \dots, X_j] = [X_1, \dots, X_j] \bar{T}_j + R_j[0, \dots, 0, I_p], \quad (9.2.8)$$

где

$$\bar{T}_j = \begin{bmatrix} M_1 & B_1^T & & \cdots & & 0 \\ B_1 & M_2 & \ddots & & & \vdots \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & & B_{j-1}^T \\ \vdots & & & & & \\ 0 & \cdots & & B_{j-1} & & M_j \end{bmatrix}.$$

Используя рассуждения, аналогичные тем, что были использованы при доказательстве теоремы 9.1.1, мы можем показать, что  $X_j$  совместно ортогональны при условии, что все матрицы  $R_j$  имеют полный ранг. Однако если  $\text{rank}(R_j) < p$  для некоторого  $j$ , то можно выбрать столбцы матрицы  $X_{j+1}$  так, что  $X_{j+1}^T X_i = 0$  для  $i = 1:j$ . См. Golub, Underwood (1977).

Поскольку  $\bar{T}_j$  имеет ленту шириной  $p$ , она может быть приведена к трехдиагональному виду эффективным образом – по алгоритму из работы Шульца (1968). Как только получена трехдиагональная форма, значения Ритца могут быть найдены с помощью симметричного QR-алгоритма.

Чтобы принять разумное решение об использовании блочного алгоритма Ланцоша, необходимо понимать, как размеры блоков влияют на сходимость значений Ритца. Этот вопрос проясняется следующим обобщением теоремы 9.1.3.

**Теорема 9.2.2.** Пусть  $A$  – симметричная  $n \times n$ -матрица с собственными значениями  $\lambda_1 \geq \dots \geq \lambda_n$  и соответствующими ортонормированными собственными векторами  $z_1, \dots, z_n$ . Пусть  $\mu_1 \geq \dots \geq \mu_p$  составляют  $p$  старших собственных значений матрицы  $\bar{T}_j$ , полученной после  $j$  шагов итераций (9.2.7) блочного Ланцоша. Если  $Z_1 = [z_1, \dots, z_p]$  и  $\cos(\theta_p) = \sigma_p(Z_1^T X_1) > 0$ , то для  $k = 1:p$  имеем  $\lambda_k \geq \mu_k \geq \lambda_k - \varepsilon_k^2$ , где

$$\varepsilon_k^2 = \frac{(\lambda_1 - \lambda_k) \tan^2(\theta_p)}{\left[ c_{j-1} \left( \frac{1 + \gamma_k}{1 - \gamma_k} \right) \right]^2}, \quad \gamma_k = \frac{\lambda_k - \lambda_{p+1}}{\lambda_k - \lambda_n};$$

$c_{j-1}(z)$  есть  $(j-1)$ -й полином Чебышева.

*Доказательство.* См. Underwood (1975).  $\square$

Аналогичные неравенства можно получить и для младших собственных значений матрицы  $\bar{T}_j$ , применяя эту теорему с заменой  $A$  на  $-A$ .

Основываясь на теореме 9.2.2 и тщательном анализе итераций (9.2.7) блочного алгоритма Ланцоша, мы можем заключить, что:

- Оценки погрешности для значений Ритца улучшаются с ростом  $p$ .
- Объем работы по вычислению собственных значений для  $\bar{T}_j$  пропорционален  $p^2$ .
- Размер блоков должен быть по меньшей мере таким же, как наибольшая кратность любого из отыскиваемых собственных значений.

Как определять размер блоков на базе компромисса отмеченных выше свойств, в деталях обсуждается в работе Scott (1979).

Потеря ортогональности является бедой и блочного алгоритма Ланцоша. Все описанные выше схемы усиления ортогональности, однако, можно применять и в блочной постановке. Детали см. в работах Scott (1979), Lewis (1977), Ruhe (1979).

### 9.2.7. s-Шаговый алгоритм Ланцоша

Блочный алгоритм Ланцоша (9.2.7) можно использовать итерационно, если нужно вычислить выборочные собственные значения для  $A$ . Чтобы изложить идеи, предположим, что мы хотим найти  $p$  старших собственных значений. Задавшись некоторой матрицей  $X_1 \in \mathbb{R}^{n \times p}$  с ортонормированными столбцами, мы можем действовать следующим образом:

**until** норма  $\|AX_1 - X_1\bar{T}_s\|_2$  достаточно мала.

Вычислить  $X_2, \dots, X_s \in \mathbb{R}^{n \times p}$  по блочному алгоритму Ланцоша.

Сформировать  $\bar{T}_s = [X_1, \dots, X_s]^T A [X_1, \dots, X_s]$

(это  $p$ -диагональная матрица размера  $sp \times sp$ ).

Вычислить ортогональную матрицу  $U = [u_1, \dots, u_{sp}]$ ,

такую, что  $U^T \bar{T}_s U = \text{diag}(\theta_1, \dots, \theta_{sp})$ , где  $\theta_1 \geq \dots \geq \theta_{sp}$ .

Положить  $X_1 = [X_1, \dots, X_s][u_1, \dots, u_p]$ .

**end**

Это блочный аналог  $s$ -шагового алгоритма Ланцоша, который всесторонне анализировался в работах Cullum, Donath (1974) и Underwood (1975).

Та же самая идея может быть использована для вычисления нескольких наименьших собственных значений для  $A$  или смешанной группы как наибольших, так и наименьших собственных значений. См. Cullum (1978). Выбор параметров  $s$  и  $p$  зависит от ограничений на память, а также от факторов, упоминавшихся выше, когда обсуждался выбор размера блоков. Блочный размер  $p$  может быть понижен при появлении хорошего вектора Ритца. Однако это требует проведения ортогонализации к сошедшимся векторам Ритца. См. Cullum, Donath (1974).

### Задачи

**9.2.1.** Докажите лемму 9.2.1.

**9.2.2.** Если  $\text{rank}(R_j) < p$  в (9.2.7), то следует ли отсюда, что  $\text{range}([X_1, \dots, X_j])$  состоит из собственных векторов для  $A$ ?

### Замечания и литература к § 9.2

Хорошие общие ссылки для практического метода Ланцоша включают Parlett (SEP) и

Cullum J. K. and Willoughby R. A. (1985b). Lanczos Algorithms for Large Symmetric, Eigenvalue Computations, Vol. 11. Programs, Birkhäuser, Boston.

Из нескольких вычислительных вариантов метода Ланцоша алгоритм 9.2.1 наиболее устойчив. Детали см. в

Paige C. C. (1972). "Computational Variants of the Lanczos Method for the Eigenproblem", *J. Inst. Math. Applic.* 10, 373–81.

Другие практические детали, связанные с реализацией процедуры Ланцоша, обсуждаются в Parlett B. N. and Nour-Omid B. (1985). "The Use of a Refind Error Bound When Updating Eigenvalues of Tridiagonals", *Lin. Alg. and Its Applic.* 68, 179–220.

Parlett B. N., Simon H., and Stringer L. M. (1982). "On Estimating the Largest Eigenvalue with the Lanczos Algorithm", *Math. Comp.* 38, 153–166.

Scott D. S. (1979). "How to Make the Lanczos Algorithm Converge Slowly", *Math. Comput.* 33, 239–47.

Поведение метода Ланцоша в условиях ошибок округления впервые описано в

Paige C. C. (1971). "The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices", Ph. D. thesis, University of London.

Важные последующие работы включают

Paige C. C. (1976a). "Error Analysis of the Lanczos Algorithm for Tridiagonalizing Symmetric Matrix", *J. Inst. Math. Applic.* 18, 341–49.

Paige C. C. (1980). "Accuracy and Effectiveness of the Lanczos Algorithm for the Symmetric Eigenproblem", *Lin. Alg. and Its Applic.* 34, 235–58.

Детали, относящиеся к полной переортогонализации, можно найти в

Golub G. H., Underwood R., and Wilkinson J. H. (1972). "The Lanczos Algorithm for the Symmetric  $Ax = \lambda Bx$  Problem", Report STAN-CS-72-270, Department of Computer Science, Stanford University, Stanford, California.

Paige C. C. (1979). "Practical Use of the Symmetric Lanczos Process with Reorthogonalization", *BIT* 10, 183–95.

Simon H. (1984). "Analysis of the Symmetric Lanczos Algorithm with Reorthogonalization Methods", *Lin. Alg. and Its Applic.* 61, 101–132.

Более эффективный подход к поддержанию ортогональности относительно уже сошедшихся векторов Ритца исследуется в

Scott D. S. (1978). "Analysis of the Symmetric Lanczos Process", UCB-ERL Technical Report M78/40, University of California, Berkeley.

Parlett B. N. and Scott D. S. (1979). "The Lanczos Algorithm with Selective Orthogonalization", *Math. Comp.* 33, 217–38.

Если не делать ортогонализацию, то необходимо либо следить за потерей ортогональности и вовремя останавливаться, либо изобрести еще какую-то схему, которая поможет проводить различие между теневыми и настоящими собственными значениями. См.

Cullum J. and Willoughby R. A. (1979). "Lanczos and the Computation in Specified Intervals of the Spectrum of Large, Sparse Real Symmetric Matrices", in *Sparse Matrix Proc.*, 1978, ed. I. S. Duff and C. W. Stewart, SIAM Publications, Philadelphia, PA.

Kahan W. and Parlett B. N. (1974). "An Analysis of Lanczos Algorithms for Symmetric Matrices", ERL-M467, University of California, Berkeley.

Kahan W. and Parlett B. N. (1976). "How Far Should You Go with the Lanczos Process?" in *Sparse Matrix Computations*, ed. J. Bunch and D. Rose, Academic Press, New York, pp. 131–44.

Lewis J. (1977). "Algorithms for Sparse Matrix Eigenvalue Problems", Report STAN-CS-77-595, Department of Computer Science, Stanford University, Stanford, California.

Parlett B. N. and Reid J. K. (1981). "Tracking the Progress of the Lanczos Algorithm for Large Symmetric Eigenproblems", *IMA J. Num. Anal.* 1, 135–55.

Блочный алгоритм Ланцоша обсуждается в

Cullum J. (1978). "The Simultaneous Computation of a Few of the Algebraically Largest and Smallest Eigenvalues of a Large Sparse Symmetric Matrix", *BIT* 18, 265–75.

- Cullum J. and Donath W. E. (1974). "A Block Lanczos Algorithm for Computing the q Algebraically Largest Eigenvalues and a Corresponding Eigenspace of Large Sparse Real Symmetric Matrices", Proc. of the 1974 IEEE Conf. on Decision and Control, Phoenix, Arisona, pp. 505–9.
- Golub G. H. and Underwood R. (1977). "The Block Lanczos Method for Computing Eigenvalues", in Mathematical Software III, ed. J. Rice, Academic Press, New York, pp. 364–77.
- Ruhe A. (1979). "Implementation Aspects of Band Lanczos Algorithm for Computation Eigenvalues of Large Sparse Symmetric Matrices", Math. Comp. 33, 680–87.
- Scott D. S. (1979). "Block Lanczos Software for Symmetric Eigenvalue Problems", Report ORNL/CSD-48, Oak Ridge National Laboratory, Union Carbide Corporation, Dak Ridge, Tennessee.
- Underwood R. (1975). "An iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems", Report STAN-CS-75-495, Department of Computer Science, Stanford University, Stanford, California.

В блочном алгоритме Ланцоша строится симметричная ленточная матрица с собственными значениями, которые можно вычислить любым из нескольких способов. Один из подходов описан в

- Schwartz H. R. (1968). "Tridiagonalization of a Symmetric Band Matrix", Numer. Math. 12, 231–41.  
See also HACLA, pp. 273–83.

В некоторых приложениях необходимо оценивать внутренние собственные значения. Алгоритму Ланцоша, однако, свойственно сначала находить крайние собственные значения. Этот вопрос рассматривается в следующих работах:

- Cline A. K., Golub G. H., and Platzman G. W. (1976). "Calculation of Normal Modes of Oceans Using a Lanczos Method", in Sparse Matrix Computations, ed. J. R. Bunch and D. J. Rose, Academic Press, New York, pp. 409–26.
- Ericsson T. and Ruhe A. (1980). "The Spectral Transformation Lanczos Method for the Numerical Solution of Large Sparse Generalized Symmetric Eigenvalue Problems", Math. Comp. 35, 1251–68.

### **Добавления при переводе к § 9.2**

Интересный анализ метода Ланцоша в условиях машинной арифметики и новая устойчивая вычислительная схема предлагаются в

- Собянин А. В. (1985). Анализ ошибок округления и устойчивость в методах типа Ланцоша.– М.: ОВМ АН СССР.

Метод Ланцоша можно рассматривать как метод ортогонализации полиномов – для определенного набора весов в точках, являющихся собственными значениями исходной матрицы. Если рекуррентные соотношения слегка возмутить (например, вследствие округления чисел), то построенные полиномы теряют ортогональность по отношению к рассмотренной мере. Тем не менее вычисленные полиномы остаются ортогональными по отношению к несколько иной мере – с весами в большем числе точек, находящихся в малой окрестности множества собственных значений матрицы. Этот факт согласуется (а возможно, и является причиной) хорошо известного явления копирования собственных значений при реальном счете. Такой взгляд на метод Ланцоша развивается в

- Greenbaum A. (1989). "Behaviour of Slightly Perturbed Lanczos and Conjugate Gradient Recurrences", Lin. Alg. Appl. 113, 7–63.

Отметим также обзор

- Икрамов Х. Д. (1982). Разреженные матрицы. Итоги науки и техники. т. 20.– М.: ВИНИТИ, 179–257.

## **9.3. Приложения и обобщения**

В этом разделе мы вкратце расскажем о том, как итерации Ланцоша могут быть приспособлены для того, чтобы решать большие разреженные линейные системы и задачи наименьших квадратов. Мы обсудим также процессы Арнольди и несимметричного метода Ланцоша.

### 9.3.1. Симметричные положительно определенные системы

Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и положительно определенная, и рассмотрим функционал  $\varphi(x)$ , имеющий вид

$$\varphi(x) = \frac{1}{2} x^T A x - x^T b,$$

где  $b \in \mathbb{R}^n$ . Поскольку  $\nabla \varphi(x) = Ax - b$ , заключаем, что  $x = A^{-1}b$  является единственной точкой минимума для  $\varphi$ . Значит, приближение к точке минимума для  $\varphi$  можно рассматривать как приближение к решению системы  $Ax = b$ .

Один из способов построения последовательности  $\{x_j\}$ , сходящейся к  $x$ , состоит в генерации последовательности ортонормированных векторов  $\{q_j\}$  и определении  $x_j$  как точки минимума  $\varphi$  на  $\text{span}\{q_1, \dots, q_j\}$ ,  $j = 1:n$ . Пусть  $Q_j = [q_1, \dots, q_j]$ . Поскольку

$$x \in \text{span}\{q_1, \dots, q_j\} \Rightarrow \varphi(x) = \frac{1}{2} y^T (Q_j^T A Q_j) y - y^T (Q_j^T b)$$

для некоторого  $y \in \mathbb{R}^j$ , отсюда вытекает, что

$$x_j = Q_j y_j, \quad (9.3.1)$$

где

$$(Q_j^T A Q_j) y_j = Q_j^T b. \quad (9.3.2)$$

Заметим, что  $Ax_n = b$ .

Теперь посмотрим, как можно этот подход к решению системы  $Ax = b$  сделать эффективным, когда  $A$  большая и разреженная. Преодолеть нужно следующие два препятствия:

- Линейная система (9.3.2) должна быть «легко» решаемой.
- Должна быть возможность получения  $x_j$  без явного обращения к векторам  $q_1, \dots, q_j$ , как предполагается в (9.3.1); в противном случае будет чрезмерным число пересылок данных.

Мы покажем, что оба эти требования удовлетворяются, если в качестве  $q_j$  взять векторы Ланцоша.

После  $j$  шагов алгоритма Ланцоша мы получаем разложение

$$A Q_j = Q_j T_j + r_j e_j^T, \quad (9.3.3)$$

где

$$T_j = Q_j^T A Q_j = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ \ddots & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \beta_{j-1} \\ 0 & \cdots & & \beta_{j-1} & \alpha_j \end{bmatrix}.$$

При таком подходе (9.3.2) становится симметричной положительно определенной трехдиагональной системой, которую можно решать на базе  $LDL^T$ -разложения (см. алгоритм 4.3.6). В частности, положим

$$L_j = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mu_1 & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \mu_{j-1} & 1 \end{bmatrix}, \quad D_j = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & d_j \end{bmatrix},$$

и сравнивая элементы в матричном равенстве

$$T_j = L_j D_j L_j^T, \quad (9.3.5)$$

мы находим, что

```
d1 = a1
for i = 2:j
    mu_i = beta_{i-1}/d_{i-1}; d_i = a_i - beta_{i-1}/mu_{i-1}
end
```

Заметим, что для того, чтобы получить  $L_j$  и  $D_j$  из  $L_{j-1}$  и  $D_{j-1}$ , нам нужно лишь найти величины

$$\mu_{j-1} = \beta_{j-1}/d_{j-1}, \quad d_j = a_j - \beta_{j-1} \mu_{j-1}. \quad (9.3.6)$$

Как уже отмечалось, важно уметь вычислять  $x_j$  из (9.3.1) эффективным образом. Для этого мы определим  $C_j \in \mathbf{R}^{n \times j}$  и  $p_j \in \mathbf{R}^j$  уравнениями

$$C_j L_j^T = Q_j, \quad L_j D_j p_j = Q_j^T b \quad (9.3.7)$$

и заметим, что

$$x_j = Q_j T_j^{-1} Q_j^T b = Q_j (L_j D_j L_j^T)^{-1} Q_j^T b = C_j p_j.$$

Рассмотрим столбцовое разбиение  $C_j = [c_1, \dots, c_j]$ . Из (9.3.7) следует, что

$$[c_1, \mu_1 c_1 + c_2, \dots, \mu_{j-1} c_{j-1} + c_j] = [q_1, \dots, q_j],$$

и поэтому

$$C_j = [C_{j-1} \ c_j], \quad c_j = q_j - \mu_{j-1} c_{j-1}.$$

Заметим также, что если мы положим  $p_j = [\rho_1, \dots, \rho_j]^T$  в  $L_j D_j p_j = Q_j^T b$ , то это уравнение принимает вид

$$\left[ \begin{array}{c|c} L_{j-1} D_{j-1} & 0 \\ \hline 0 \cdots 0 & \mu_{j-1} d_{j-1} \end{array} \right] \left[ \begin{array}{c} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{j-1} \\ \rho_j \end{array} \right] = \left[ \begin{array}{c} q_1^T b \\ q_2^T b \\ \vdots \\ q_{j-1}^T b \\ q_j^T b \end{array} \right].$$

В силу  $L_{j-1} D_{j-1} p_{j-1} = Q_{j-1}^T b$  получаем

$$p_j = \begin{bmatrix} p_{j-1} \\ \rho_j \end{bmatrix}, \quad \rho_j = (q_j^T b - \mu_{j-1} d_{j-1} \rho_{j-1})/d_j,$$

и, таким образом,  $x_j = C_j p_j = C_{j-1} p_{j-1} + \rho_j c_j = x_{j-1} + \rho_j c_j$ . Это именно та рекуррентная формула для  $x$ , которая нам нужна. Вместе с (9.3.6) и (9.3.7) она позволяет нам выполнить переход от  $(q_{j-1}, c_{j-1}, x_{j-1})$  к  $(q_j, c_j, x_j)$  с минимальными затратами по числу операций и памяти.

Дальнейшее упрощение получается, если положить  $q_1 = b/\beta_0$ , где  $\beta_0 = \|\beta\|_2$ . При таком выборе начального вектора для Ланцоша мы видим, что  $q_i^T b = 0$  при  $i \geq 2$ . Из

(9.3.3) вытекает, что

$$Ax_j = A Q_j y_j = Q_j T_j y_j + r_j e_j^T y_j = Q_j Q_j^T b + r_j e_j^T y_j = b + r_j e_j^T y_j.$$

Итак, если  $\beta_j = \|r_j\|_2 = 0$  в итерациях Ланцюша, то  $Ax_j = b$ . Более того, поскольку  $\|Ax_j - b\|_2 = \beta_j |e_j^T y_j|$ , итерации обеспечивают оценки текущей невязки. В целом, мы приходим к следующей процедуре.

**Алгоритм 9.3.1.** Если  $b \in \mathbf{R}^n$  и матрица  $A \in \mathbf{R}^{n \times n}$  симметричная положительно определенная, то алгоритм вычисляет вектор  $x \in \mathbf{R}^n$ , такой, что  $Ax = b$ .

```

 $r_0 = b; \beta_0 = \|b\|_2; q_0 = 0; j = 0; x_0 = 0$ 
while  $\beta_j \neq 0$ 
     $q_{j+1} = r_j / \beta_j; j = j + 1; \alpha_j = q_j^T A q_j$ 
     $r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \beta_j = \|r_j\|_2$ 
    if  $j = 1$ 
         $d_1 = \alpha_1; c_1 = q_1; \rho_1 = \alpha_1 / \beta_0; x_1 = \rho_1 q_1$ 
    else
         $\mu_{j-1} = \beta_{j-1} / d_{j-1}; d_j = \alpha_j - \beta_{j-1} \mu_{j-1}$ 
         $c_j = q_j - \mu_{j-1} c_{j-1}; \rho_j = -\mu_{j-1} d_{j-1} \rho_{j-1} / d_j$ 
         $x_j = x_{j-1} + \rho_j c_j$ 
    end
    end
     $x = x_j$ 

```

Этот алгоритм требует одно матрично-векторное умножение и несколько операций saxpy на итерацию<sup>1)</sup>. Численное поведение алгоритма 9.3.1 обсуждается в следующей главе, где мы выводим его заново и узнаем в нем широко известный метод сопряженных градиентов.

### 9.3.2. Симметричные неопределенные системы

Ключевую роль в предыдущих рассмотрениях играет идея вычисления  $LDL^T$ -разложения трехдиагональных матриц  $T_j$ . Если  $A$  (и, как следствие,  $T_j$ ) не положительно определенная, то, к сожалению, это разложение потенциально неустойчиво. В способе обхода этой трудности, предложенном в Paige, Saunders (1975), предлагаются строить рекурсии для  $x_j$ , опираясь на LQ-разложение для  $T_j$ . Именно, на  $j$ -м шаге итераций мы получаем вращения Гивенса  $J_1, \dots, J_{j-1}$ , такие, что

$$T_j J_1 \cdots J_{j-1} = L_j = \begin{bmatrix} d_1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ e_1 & d_2 & 0 & \cdots & \cdots & \cdots & 0 \\ f_1 & e_2 & d_3 & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & f_{j-2} & e_{j-1} & d_j \end{bmatrix}.$$

Заметим, что в согласии с этим разложением  $x_j$  имеет вид

$$x_j = Q_j y_j = Q_j T_j^{-1} Q_j^T b = W_j s_j,$$

<sup>1)</sup> Сюда необходимо добавить одно скалярное произведение и одно вычисление 2-нормы вектора. *Прим. перев.*

где  $W_j \in \mathbb{R}^{n \times j}$  и  $s_j \in \mathbb{R}^j$  определяются как

$$W_j = Q_j J_1 \dots J_{j-1}, \quad L_j s_j = Q_j^T b.$$

Тщательный анализ этих уравнений позволяет вывести формулу для вычисления  $x_j$  по  $x_{j-1}$  и некоторого легко получаемого кратного вектора  $w_j$  – последнего столбца в  $W_j$ . Детали см. в Paige, Saunders (1975).

### 9.3.3. Двухдиагонализация и SVD

Предположим, что для матрицы  $A \in \mathbb{R}^{m \times n}$  проведена двухдиагонализация  $U^T A V = B$ , где

$$\begin{aligned} U &= [u_1, \dots, u_m], \quad U^T U = I_m, \\ V &= [v_1, \dots, v_n], \quad V^T V = I_n, \\ B &= \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ 0 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & 0 & & \alpha_n \end{bmatrix}. \end{aligned} \quad (9.3.8)$$

Напомним (см. § 5.4.3), что это разложение может быть вычислено с помощью преобразований Хаусхолдера и что оно используется как прелюдия к SVD-алгоритму.

К несчастью, если матрица  $A$  большая и разреженная, то в процессе хаусхолдеровой двухдиагонализации можно ожидать появления больших и плотных подматриц. Поэтому будет прекрасно, если мы найдем средства для вычисления  $B$  непосредственно, без последовательной двухдиагонализации подматриц матрицы  $A$ .

Действуя так же, как в § 9.1.2, приравняем столбцы в уравнениях  $AV = UB$  и  $A^T U = VB^T$  для  $j = 1:n$  и получим

$$\begin{aligned} Av_j &= \alpha_j u_j + \beta_{j-1} u_{j-1}, \quad \beta_0 u_0 \equiv 0, \\ A^T u_j &= \alpha_j v_j + \beta_j v_{j+1}, \quad \beta_n v_{n+1} \equiv 0. \end{aligned} \quad (9.3.9)$$

Определив  $r_j = Av_j - \beta_j u_{j-1}$  и  $p_j = A^T u_j - \alpha_j v_j$ , из ортогональности мы можем вывести, что  $\alpha_j = \pm \|r_j\|_2$ ,  $u_j = r_j/\alpha_j$ ,  $\beta_j = \pm \|p_j\|_2$ ,  $v_{j+1} = p_j/\beta_j$ . После правильного упорядочения эти уравнения будут определять метод Ланцоша для двухдиагонализации прямоугольной матрицы:

```

 $v_1$  = заданный  $n$ -вектор с единичной 2-нормой
 $p_0 = v_1$ ;  $\beta_0 = 1$ ;  $j = 0$ ;  $u_0 = 0$ 
while  $\beta_j \neq 0$  (9.3.10)
     $v_{j+1} = p_j/\beta_j$ ;  $j = j + 1$ 
     $r_j = Av_j - \beta_{j-1} u_{j-1}$ ;  $\alpha_j = \|r_j\|_2$ ;  $u_j = r_j/\alpha_j$ 
     $p_j = A^T u_j - \alpha_j v_j$ ;  $\beta_j = \|p_j\|_2$ 
end

```

Если  $\text{rank}(A) = n$ , то мы гарантируем отсутствие нулевых  $\alpha_j$ . Действительно, если  $\alpha_j = 0$ , то  $\text{span}\{Av_1, \dots, Av_j\} \subset \text{span}\{u_1, \dots, u_{j-1}\}$ , что влечет за собой неполноту ранга.

Если  $\beta_j = 0$ , то нетрудно проверить, что

$$A[v_1, \dots, v_j] = [u_1, \dots, u_j] B_j,$$

$$A^T [u_1, \dots, u_j] = [v_1, \dots, v_j] B_j^T,$$

где  $B_j = B(1:j, 1:j)$ , а  $B$  имеет вид (9.3.8). Таким образом, в данном случае векторы  $v$  и  $u$  являются сингулярными векторами и  $\sigma(B_j) \subset \sigma(A)$ . Ланцошева двухдиагонализация обсуждается в Paige (1974). См. также Cullum, Willoughby (1985a, 1985b). По существу, это эквивалентно применению схемы ланцошевой трехдиагонализации к симметричной матрице

$$C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}.$$

В начале § 8.3 мы показали, что  $\lambda_i(C) = \sigma_i(A) = -\lambda_{n+m-i+1}(C)$  для  $i = 1:n$ . Вследствие этого неудивительно, что большие сингулярные значения двухдиагональной матрицы оказываются очень хорошими приближениями к большим сингулярным значениям матрицы  $A$ . Малые сингулярные значения для  $A$  отвечают внутренним собственным значениям для  $C$  и аппроксимируются уже не так хорошо. Аналог теории Каниэля–Пейджа для двухдиагонализации Ланцоша можно найти в Luk (1978), а также в Golub, Luk, Overton (1981). Аналитические, алгоритмические и численные рассмотрения, проведенные в двух предыдущих разделах, естественным образом переносятся и на двухдиагонализацию.

### 9.3.4. Наименьшие квадраты

Задачу наименьших квадратов ( $LS$ )  $\min \|Ax - b\|_2$  с матрицей  $A$  полного ранга также можно решить с помощью двухдиагонализации. Именно,

$$x_{LS} = Vy_{LS} = \sum_{i=1}^n y_i v_i,$$

где  $y = (y_1, \dots, y_n)^T$  – решение системы  $By = (u_1^T b, \dots, u_n^T b)^T$ . Заметим, что вследствие верхнего двухдиагонального вида матрицы  $B$  мы не можем начать вычисление вектора  $b$  до завершения двухдиагонализации. Более того, мы вынуждены хранить векторы  $v_1, \dots, v_n$ ; это малоприятное занятие в случае больших  $n$ .

Развитие основанного на двухдиагонализации алгоритма для разреженных задач наименьших квадратов может пойти по более удачному пути, если  $A$  приводится к нижнему треугольному виду

$$U^T AV = B = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \cdots & \vdots \\ \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & \ddots & \alpha_n \\ 0 & \cdots & 0 & \beta_n \end{bmatrix},$$

где  $V = [v_1, \dots, v_n]$  и  $U = [u_1, \dots, u_{n+1}]$ . Осуществляющая это процедура Ланцоша разрабатывается без труда, а итоговый алгоритм очень похож на (9.3.10).

Пусть  $V_j = [v_1, \dots, v_j]$ ,  $U_j = [u_1, \dots, u_j]$ ,  $B_j = (1:j+1, 1:j)$ , и рассмотрим минимизацию  $\|Ax - b\|_2$  по всем векторам вида  $x = V_j y$ , где  $y \in \mathbb{R}^j$ . Поскольку

$$\|AV_j y - b\|_2 = \|U^T AV_j y - U^T b\|_2 = \|B_j y - U_{j+1}^T b\|_2 + \sum_{i=j+2}^m (u_i^T b)^2,$$

искомый минимизирующий вектор  $x$  имеет вид  $x_j = V_j y_j$ , где  $y_j$  минимизирует  $(j+1) \times j$ -задачу наименьших квадратов  $\min \|B_j y - U_{j+1}^T b\|_2$ .

Вследствие того, что матрица  $B_j$  – нижняя двухдиагональная, легко вычисляются вращения Гивенса  $J_1, \dots, J_j$ , такие, что матрица

$$J_j \dots J_1 B_j = \begin{bmatrix} R_j \\ 0 \end{bmatrix}_1^j$$

является верхней двухдиагональной. Если

$$J_j \dots J_1 U_{j+1}^T b = \begin{bmatrix} d_j \\ u \end{bmatrix}_1^j,$$

то отсюда вытекает, что  $x_j = V_j y_j = W_j d_j$ , где  $W_j = V_j R_j^{-1}$ . В Paige, Saunders (1975) показано, как можно получить  $x_j$  из  $x_{j+1}$  посредством простой рекурсии, привлекающей последний столбец матрицы  $W_j$ . Окончательным результатом является алгоритм для разреженных задач наименьших квадратов, требующий для своей реализации запоминания лишь нескольких  $n$ -векторов.

### 9.3.5. Идея Арнольди

Один из способов обобщения процесса Ланцоша на несимметричные матрицы принадлежит Арнольди (Arnoldi (1951)) и опирается на хессенбергову редукцию  $Q^T A Q = H$ . Именно, если мы запишем  $Q = [q_1, \dots, q_n]$  и приравняем столбцы в  $AQ = QH$ , то получим

$$Aq_j = \sum_{i=1}^{j+1} h_{ij} q_i, \quad 1 \leq j \leq n-1.$$

Таким образом, если мы нашли  $q_1, \dots, q_j$  и вычислили  $h_{ij} = q_i^T A q_j$  при  $i = 1:j$ , то  $q_{j+1} = r_{j+1} / \|r_{j+1}\|_2$ , где

$$r_{j+1} = Aq_j - \sum_{i=1}^j h_{ij} q_i.$$

Если  $r_{j+1} = 0$ , то нетрудно показать, что подпространство  $\text{span}\{q_1, \dots, q_j\}$  инвариантно относительно  $A$ . В целом, мы имеем

```

 $r = q_1; \beta = 1; j = 0$ 
while  $\beta \neq 0$ 
     $h_{j+1,j} = \beta; q_{j+1} = r_j / \beta; j = j + 1$ 
     $w = Aq_j; r = w$ 
    for  $i = 1:j$ 
         $h_{ij} = q_i^T w; r = r - h_{ij} q_i$ 
    end
     $\beta = \|r\|_2$ 
    if  $j < n$ 
         $h_{j+1,j} = \beta$ 
    end
end

```

Это и есть то, что называют итерациями Арнольди. Некоторые их численные свойства обсуждаются в Wilkinson (AEP, 382ff.). Как и в симметричных ланцошевых итерациях, вызывает вопросы потеря ортогональности среди  $q_j$ . Заметим также, что на каждом шаге происходит обращение к каждому из вычисленных векторов  $q_j$ .

Точно так же, как из итераций Ланцоша мы получим процедуры решения симметричных систем  $Ax = b$ , процесс Арнольди может послужить базой для построения процедур решения разреженных линейных систем. Центральная идея состоит в том, чтобы вычислять последовательность приближенных решений  $\{x_j\}$  с тем свойством, что  $x_j$  есть решение задачи

$$\min_{x \in \mathcal{K}(A, q_1, j)} \|Ax - b\|_2,$$

где  $\mathcal{K}(A, q_1, j) = \text{span}\{q_1, Aq_1, \dots, A^{j-1}q_1\} = \text{span}\{q_1, \dots, q_j\}$ . Ссылки на эти методы, связанные с подпространствами Крылова, приводятся в конце параграфа.

### 9.3.6. Несимметричная трехдиагонализация Ланцоша

Еще один подход к обобщению процесса Ланцоша на симметричный случай связан с приведением  $A$  к трехдиагональной форме с помощью преобразований подобия. Однако, как указывается в Wilkinson (AEP, pp. 388–405), из соображений численной устойчивости лучше отказаться от подобных преобразований при трехдиагонализации матрицы общего вида. Несмотря на это, нам кажется, что будет интересно познакомиться с несимметричным процессом Ланцоша, — хотя бы для того, чтобы высветить дополнительные трудности, сопутствующие несимметричным спектральным задачам.

Предположим, что  $A \in \mathbb{R}^{n \times n}$  и существует невырожденная матрица  $X$ , такая, что

$$X^{-1}AX = T = \begin{bmatrix} \alpha_1 & \gamma_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \gamma_{n-1} \\ \vdots & & & & \\ 0 & \cdots & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Рассматривая столбцовые разбиения

$$X = [x_1, \dots, x_n], \\ X^{-1} = Y = [y_1, \dots, y_n],$$

и, приравнивая столбцы в  $AX = XT$  и  $A^T Y = YT^T$ , находим

$$Ax_j = \gamma_{j-1}x_{j-1} + \alpha_jx_j + \beta_jx_{j+1}, \quad \gamma_0x_0 = 0,$$

и

$$A^T y_j = \beta_{j-1}y_{j-1} + \alpha_jy_j + \gamma_jy_{j+1}, \quad \beta_0y_0 \equiv 0,$$

для  $j = 1:n-1$ . Эти уравнения вместе с  $Y^T X = I_n$  влекут за собой

$$\begin{aligned} a_j &= y_j^T Ax_j \\ \beta_j x_{j+1} &= r_j = (A - \alpha_j I)x_j - \gamma_{j-1}x_{j-1}, \\ \gamma_j y_{j+1} &= p_j = (A - \alpha_j I)^T y_j - \beta_{j-1}y_{j-1}. \end{aligned}$$

Некоторый произвол есть в выборе масштабирующих множителей  $\beta_j$  и  $\gamma_j$ . Стандартный выбор – взять  $\beta_j = \|r_j\|_2$  и  $\gamma_j = x_{j+1}^T p_j$  – приводит к следующему алгоритму:

$x_1, y_1$  – заданные ненулевые векторы.

$j = 0; \beta_0 = 1; x_0 = 0; r_0 = x_1; y_0 = 0; p_0 = y_1$

**while**  $\beta_j \neq 0 \wedge r_j^T p_j \neq 0$

$$\gamma_j = r_j^T p_j / \beta_j$$

$$x_{j+1} = r_j / \beta_j; y_{j+1} = p_j / \gamma_j$$

$$j = j + 1$$

$$\alpha_j = y_j^T A x_j; r_j = (A - \alpha_j I) x_j - \gamma_{j-1} x_{j-1}; \beta_j = \|r_j\|_2$$

$$p_j = (A - \alpha_j I)^T y_j - \beta_{j-1} y_{j-1}$$

**end**

(9.3.11)

Пусть  $X_j = [x_1, \dots, x_j]$ ,  $Y_j = [y_1, \dots, y_j]$  и  $T_j$  – ведущая  $j \times j$ -подматрица в  $T$ . Легко проверить, что

$$AX_j = X_j T_j + r_j e_j^T, \quad A^T Y_j = Y_j T_j^T + p_j e_j^T.$$

Таким образом, всякий раз, когда  $\beta_j = \|r_j\|_2 = 0$ , столбцы матрицы  $X_j$  определяют инвариантное подпространство для  $A$ . Окончание по этой причине, следовательно, событие желанное. К сожалению, если  $\gamma_j = x_{j+1}^T p_j = 0$ , то итерации заканчиваются, но не дают никакой информации об инвариантных подпространствах для  $A$  или  $A^T$ .

Эта проблема – одна из нескольких трудностей, связанных с (9.3.11). К числу других трудностей относятся отсутствие сходимости собственных значений матриц  $T_j$  и близкие к линейно зависимые векторы  $x_j$  и  $y_i$ . Нам неизвестны какие-либо успешные реализации несимметричного алгоритма Ланцюша.

### Задачи

**9.3.1.** Показать, что вектор  $x_j$  из алгоритма 9.3.1 удовлетворяет  $\|Ax_j - b\|_2 = \beta_j |e_j^T y_j|$ , где  $T_j y_j = Q_j^T b$ .

**9.3.2.** Привести пример начального вектора, для которого несимметричные итерации Ланцюша (9.3.12) обрываются и при этом не дают какой-либо информации о инвариантном подпространстве. Возьмем

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

**9.3.3.** Модифицировать алгоритм 9.3.1 так, чтобы он реализовал процедуру решения неопределенной симметричной системы, следуя схеме из § 9.3.2.

**9.3.4.** Сколько нужно рабочих векторов, чтобы эффективно реализовать (9.3.10)?

**9.3.5.** Предположим, что  $A$  неполного ранга и  $a_j = 0$  в (9.3.10). Как получить  $u_j$ , причем итерации могли бы продолжаться?

**9.3.6.** Разработайте нижний бидиагональный вариант (9.3.10) и подробно опишите процедуру решения по методу наименьших квадратов, намеченный в § 9.3.4.

### Замечания и литература к § 9.3

Большая часть материала этого параграфа представляет собой выжимку из следующих работ:

Paige C. C. (1974a). "Bidiagonalization of matrices and Solution of Linear Equations", SIAM J. Num. Anal. 11, 197–209.

Paige C. C. and Saunders M. A. (1978). "Solution of Sparse Indefinite Systems of Linear Equations", SIAM J. Num. Anal. 12, 617–29.

Paige C. C. and Saunders M. A. (1975). "A Bidiagonalization Algorithm for Sparse Linear Equations and Least Squares Problems", Report SOL 78-19, Department of Operations Research, Stanford University, Stanford, California.

Paige C. C. and Saunders M. A. (1982a). "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares", ACM Trans. Math. Soft. 8, 43–71.

Paige C. C. and Saunders M. A. (1982b). "Algorithm 583 LSQR: Sparse Linear Equations and Least Squares Problems", ACM Trans. Math. Soft. 8, 195–209.

См. также

Cullum J., Willoughby R. A., and M. Lake (1983). "A Lanczos Algorithm for Computing Singular Values and Vectors of Large Matrices", SIAM J. Sci. and Stat. Comp. 4, 197–215.

Cullum J. and Willoughby R. A. (1985a). Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1 Theory, Birkhäuser, Boston.

Cullum J. and Willoughby R. A. (1985b). Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 11 Programs, Birkhäuser, Boston.

Golub G. H., Luk F. T., and Overton M. (1981). "A Block Lanczos Method for Computing the Singular Values and Corresponding Singular Vectors of a Matrix", ACM Trans. Math. Soft. 7, 149–69.

Luk F. T. (1978). "Sparse and Parallel Matrix Computations", Report STAN-CS-78-685, Department of Computer Science, Stanford University, Standford, California.

Parlett B. N. (1980a). "A New Look at the Lanczos Algorithm for Solving Symmetric Systems of Linear Equations", Lin. Alg. and Its Applic. 29, 323–46.

Van der Vorst H. A. (1982). "A Generalized Lanczos Scheme", Math. Comp. 39, 559–562.

Saad Y. (1987). "On the Lanczos Method for Solving Symmetric Systems with Several Right Hand Sides", Math. Comp. 48, 651–662.

Widlund O. (1978). "A Lanczos Method for a Class of Nonsymmetric Systems of Linear Equations", SIAM J. Num. Anal. 15, 801–12.

Разреженные несимметричные процедуры для  $Ax = b$ , основанные на идеях Ланцоша/Арнольди, обсуждаются в

Arnoldi W. E. (1951). "The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem", Quart. Appl. Math. 9, 17–29.

Saad Y. (1982). "The Lanczos Biorthogonalization Algorithm and Other Oblique Projection Methods for Solving Large Unsymmetric Systems", SIAM J. Numer. Anal. 19, 485–506.

Saad Y. (1984). "Practical Use of Some Krylov Subspace Methods for Solving Indefinite and Nonsymmetric Linear Systems", SIAM J. Sci. and Stat. Comp. 5, 203–228.

Saad Y. and Schultz M. (1986). GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems", SIAM J. Scientific and Stat. Comp. 7, 856–869.

Walker H. F. (1988). "Implementation of the GMRES Method Using Householder Transformations", SIAM J. Sci. Stat. Comp. 9, 152–163.

Обобщения процесса Ланцоша на спектральные задачи других типов обсуждаются в

Golub G. H., Underwood R., and Wilkinson J. H. (1972). "The Lanczos Algorithm for the Symmetric  $Ax = \lambda Bx$  Problem", Report STAN-CS-72-270, Department of Computer Science, Stanford University, Stanford, California.

Van der Vorst H. A. (1982). "A Generalized Lanczos Scheme", Math. Comp. 39, 559–562.

## Глава 10

# Итерационные методы для линейных систем

В конце предыдущей главы было показано, каким образом итерации Ланцоша можно использовать для решения различных линейных уравнений и задач наименьших квадратов. Разработанные методы подходят для больших разреженных задач, так как они не требуют разложения соответствующей матрицы. В этой главе мы продолжим обсуждение процедур решения линейных уравнений, обладающих этим же свойством.

Первый раздел – это беглый обзор классических итерационных методов: Якоби, Гаусса – Зейделя, SOR, чебышевских полуитераций и т. д. Мы даем краткое изложение этих методов, потому что наша главная цель в этой главе – основательно представить метод сопряженных градиентов. В § 10.2 мы подробно излагаем эту важную технику, естественным образом отталкиваясь от метода наискорейшего спуска. Напомним, что в § 9.3 метод сопряженных градиентов уже появился в связи с итерациями Ланцоша. Метод выводится заново для того, чтобы объяснить некоторые его практические варианты, которым посвящен § 10.3.

Мы предупреждаем читателя о непоследовательности в обозначениях этой главы. В § 10.1 разработка методов на « $i, j$ -уровне» делает необходимым использование верхних индексов:  $x_i^{(k)}$  обозначает  $i$ -ю компоненту вектора  $x^{(k)}$ . В других разделах, однако, развитие алгоритмов происходит без явного упоминания компонент векторов или матриц. Поэтому в § 10.2 и § 10.3 мы можем отказаться от верхних индексов, а последовательности векторов обозначать как  $\{x_k\}$ .

## 10.1. Стандартные итерации

В гл. 3 и 4 процедуры решения линейных уравнений были связаны с разложением матрицы коэффициентов  $A$ . Методы такого типа называются *прямыми методами*. Если матрица  $A$  большая и разреженная, то искомые составляющие разложений могут быть плотными, и это ограничивает практическое применение прямых методов. Одним из исключений является случай ленточной матрицы  $A$  (см. § 4.3). Но даже во многих задачах с ленточными матрицами сама лента оказывается разреженной, внося трудности в реализацию алгоритмов типа ленточного метода Холецкого.

Одна из причин огромного интереса к процедурам решения разреженных линейных уравнений связана с тем значением, которое придается умению находить численные решения уравнений в частных производных. В действительности именно исследователи в области вычислительных методов для уравнений в частных производных ответственны за многие из подходов к разреженным матрицам, которые теперь используются повсеместно.

Грубо говоря, есть два подхода к разреженной задаче  $Ax = b$ . Один из них заключается в выборе подходящего прямого метода и его адаптации с тем, чтобы

учесть разреженность в  $A$ . Типичные стратегии адаптации связаны с разумным использованием структур данных и со специальными, минимизирующими заполнения стратегиями выбора ведущего элемента. Есть обширная литература по этим вопросам; заинтересованному читателю следует обратиться к книгам George, Liu (1981) и Duff, Erisman, Reid (1986).

Противоположностью прямым методам являются итерационные методы. Эти методы порождают последовательность приближенных решений  $\{x_k\}$ , и по существу  $A$  участвует лишь в матрично-векторных умножениях. При оценивании качества итерационного метода неизменно в центре внимания вопрос о том, как быстро сходятся итерации  $x^{(k)}$ . В этом параграфе мы представим некоторые основные итерационные методы, обсудим их практические реализации и докажем несколько характерных теорем о поведении итераций.

### 10.1.1. Итерации Якоби и Гаусса–Зейделя

Простейшей итерационной схемой, возможно, являются *итерации Якоби*. Они определяются для матриц с ненулевыми диагональными элементами. На метод может натолкнуть запись  $3 \times 3$ -системы  $Ax = b$  в следующем виде:

$$\begin{aligned} x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11}, \\ x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22}, \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}. \end{aligned}$$

Предположим, что  $x^{(k)}$  – какое-то приближение к  $x = A^{-1}b$ . Чтобы получить новое приближение  $x^{(k+1)}$ , естественно взять

$$\begin{aligned} x_1^{(k+1)} &= (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11}, \\ x_2^{(k+1)} &= (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)})/a_{22}, \\ x_3^{(k+1)} &= (b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})/a_{33}. \end{aligned} \quad (10.1.1)$$

Это и определяет итерации Якоби в случае  $n = 3$ . Для произвольных  $n$  мы имеем

**for**  $i = 1:n$

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{l=i+1}^n a_{il}x_l^{(k)} \right) / a_{ii} \quad (10.1.2)$$

**end**.

Заметим, что в итерациях Якоби при вычислении  $x_i^{(k+1)}$  не используется информация, полученная в самый последний момент. Например, при вычислении  $x_2^{(k+1)}$  используется  $x_1^{(k)}$ , хотя уже известна компонента  $x_1^{(k+1)}$ . Если мы пересмотрим итерации Якоби с тем, чтобы всегда использовать самые последние оценки для  $x_i$ , то получим

**for**  $i = 1:n$

$$x_i^{(k+1)} = \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) / a_{ii} \quad (10.1.3)$$

**end**

Так определяется то, что называется *итерациями Гаусса–Зейделя*.

Для итераций Якоби и Гаусса–Зейделя переход от  $x^{(k)}$  к  $x^{(k+1)}$  в сжатой форме описывается в терминах матриц  $L$ ,  $D$  и  $U$ , определяемых следующим образом:

$$L = \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \\ a_{21} & 0 & \cdots & & \vdots \\ a_{31} & a_{32} & \ddots & & 0 \\ \vdots & & & 0 & 0 \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{bmatrix},$$

$$D = \text{diag}(a_{11}, \dots, a_{nn}), \quad (10.1.4)$$

$$U = \begin{bmatrix} 0 & a_{12} & \cdots & \cdots & a_{1n} \\ 0 & 0 & \cdots & & \vdots \\ 0 & 0 & \ddots & & a_{n-2,n} \\ \vdots & & & & a_{n-1,n} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

Именно, шаг Якоби имеет вид  $M_J x^{(k+1)} = N_J x^{(k)} + b$ , где  $M_J = D$  и  $N_J = -(L + U)$ . С другой стороны, шаг Гаусса–Зейделя определяется как  $M_G x^{(k+1)} = N_G x^{(k)} + b$ , где  $M_G = (D + L)$  и  $N_G = -U$ .

### 10.1.2. Расщепления и сходимость

Процедуры Якоби и Гаусса–Зейделя – это типичные представители большого семейства итерационных методов, имеющих вид

$$Mx^{(k+1)} = Nx^{(k)} + b, \quad (10.1.5)$$

где  $A = M - N$  есть расщепление матрицы  $A$ . Для практического применения итераций (10.1.5) должна «легко» решаться система с матрицей  $M$ . Заметим, что для Якоби и Гаусса–Зейделя матрица  $M$  соответственно диагональная и нижняя треугольная.

Сходятся ли итерации (10.1.5) к  $x = A^{-1}$ , зависит от собственных значений матрицы  $M^{-1}N$ . Именно, определим спектральный радиус произвольной  $n \times n$ -матрицы  $G$  как

$$\rho(G) = \max \{ |\lambda| : \lambda \in \lambda(G) \};$$

тогда величина  $\rho(M^{-1}N)$  и определяет, будут ли итерации (10.1.5) успешными.

**Теорема 10.1.1.** Предположим, что  $b \in \mathbb{R}^n$  и матрица  $A = M + N \in \mathbb{R}^{n \times n}$  невырожденная. Если  $M$  невырожденная и спектральный радиус матрицы  $M^{-1}N$  удовлетворяет неравенству  $\rho(M^{-1}N) < 1$ , то итерации  $x^{(k)}$ , определенные согласно  $Mx^{(k+1)} = Nx^{(k)} + b$ , сходятся к  $x = A^{-1}b$  при любом начальном векторе  $x^{(0)}$ .<sup>1)</sup>

**Доказательство.** Пусть  $e^{(k)} = x^{(k)} - x$  обозначает погрешность на  $k$ -й итерации. Вследствие того что  $Mx = Nx + b$ , находим  $M(x^{(k+1)} - x) = N(x^{(k)} - x)$ , и, значит, погрешность для  $x^{(k+1)}$  имеет вид  $e^{(k+1)} = M^{-1}Ne^{(k)} = (M^{-1}N)^k e^{(0)}$ . Из леммы 7.3.2 мы знаем, что  $(M^{-1}N)^k \rightarrow 0$  тогда и только тогда, когда  $\rho(M^{-1}N) < 1$ .  $\square$

<sup>1)</sup> Обычно формулируют более полное (см. задачу 10.1.6) утверждение, а именно условие  $\rho(M^{-1}N) < 1$  необходимо и достаточно для сходимости при любом начальном векторе. – Прим. перев.

Этот результат является центральным при таком изучении итерационных методов, когда разработка алгоритмов обычно происходит по следующим направлениям:

- Предлагаются расщепления  $A = M - N$ , такие, что линейные системы  $Mz = d$  «легко» решаются.
- Описываются классы матриц, для которых итерационная матрица  $G = M^{-1}N$  удовлетворяет  $\rho(G) < 1$ .
- Устанавливаются дальнейшие результаты относительно  $\rho(G)$ , дающие представление о том, каким образом погрешности  $e^{(k)}$  стремятся к нулю.

Например, рассмотрим итерации Якоби  $Dx^{(k+1)} = -(L + U)x^{(k)} + b$ . Одним из условий, обеспечивающих  $\rho(M_J^{-1}N_J) < 1$ , является строгое преобладание. Действительно, если  $A$  обладает этим свойством (определение см. в § 3.4.10), то

$$\rho(M_J^{-1}N_J) \leq \|D^{-1}(L + U)\|_{\infty} = \max_{1 \leq i \leq n} \sum_{\substack{j=1 \\ j \neq i}}^n \left| \frac{a_{ij}}{a_{ii}} \right| < 1.$$

Как правило, чем «сильнее» диагональное преобладание, тем быстрее сходимость, но есть и контрпримеры: см. задачу 10.1.8.

Для метода Гаусса–Зейделя, чтобы установить его сходимость для симметричной положительно определенной  $A$ ; оценка спектрального радиуса осуществляется сложнее.

**Теорема 10.1.2.** Если матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и положительно определенная, то итерации Гаусса–Зейделя (10.1.3) сходятся при любом  $x^{(0)}$ .

**Доказательство.** Запишем  $A = L + D + L^T$ , где  $D = \text{diag}(a_{ii})$  и  $L$  нижняя строго треугольная. В свете теоремы 10.1.1. наша задача – показать, что все собственные значения матрицы  $G = -(D + L)^{-1}L^T$  находятся внутри единичного круга. Вследствие положительной определенности матрицы  $D$  имеем  $G_1 \equiv D^{1/2}GD^{-1/2} = -(I + L_1)^{-1}L_1^T$ , где  $L_1 = D^{-1/2}LD^{1/2}$ . Поскольку  $G$  и  $G_1$  имеют одинаковые собственные значения, мы должны убедиться в том, что  $\rho(G_1) < 1$ . Если  $G_1x = \lambda x$  и  $x^Hx = 1$ , то  $-L_1^Tx = \lambda(I + L_1)x$  и, следовательно,  $-x^HL_1^Tx = \lambda(1 + x^HL_1x)$ . Положив  $a + bi = x^HL_1x$ , получаем

$$|\lambda|^2 = \left| \frac{-a + bi}{1 + a + bi} \right|^2 = \frac{a^2 + b^2}{1 + 2a + a^2 + b^2}.$$

Однако в силу положительной определенности матрицы  $D^{-1/2}AD^{-1/2} = I + L_1 + L_1^T$  нетрудно показать, что  $0 < 1 + x^HL_1x + x^HL_1x = 1 + 2a$ , откуда и следует, что  $|\lambda| < 1$ .  $\square$

Этот результат используется постоянно, из-за того что многие матрицы, возникающие при дискретизации эллиптических уравнений в частных производных, являются симметричными положительно определенными. В литературе появилось множество других результатов в том же духе. См. Varga (1962), Young (1971), Hageman, Young (1981).

### 10.1.3. Практическая реализация Гаусса–Зейделя

Теперь мы сосредоточимся на некоторых практических деталях, связанных с итерациями Гаусса–Зейделя. Особенно просто реализуется шаг Гаусса–Зейделя (10.1.3) с замещением:

```

for  $i = 1 : n$ 
     $x_i = \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^n a_{ij}x_j \right) / a_{ii}$ 
end

```

Число флопов в этих вычислениях примерно в два раза больше числа ненулевых элементов в матрице  $A$ . Нет никакого смысла в более точном оценивании вычислительной работы, потому что реализация в действительности сильно зависит от структуры решаемой задачи.

Для того чтобы подчеркнуть эту мысль, рассмотрим применение алгоритма 10.1.1. к блочно-трехдиагональной системе размера  $NM \times NM$ :

$$\begin{bmatrix} T & -I_N & \cdots & 0 \\ -I_N & T & \ddots & \vdots \\ \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -I_N \\ 0 & \cdots & -I_N & T \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_M \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_M \end{bmatrix}, \quad (10.1.6)$$

где

$$T = \begin{bmatrix} 4 & -1 & \cdots & 0 \\ -1 & 4 & \ddots & \vdots \\ \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 4 \end{bmatrix}, \quad g_j = \begin{bmatrix} G(1,j) \\ G(2,j) \\ \vdots \\ G(N,j) \end{bmatrix}, \quad f_j = \begin{bmatrix} F(1,j) \\ F(2,j) \\ \vdots \\ F(N,j) \end{bmatrix}.$$

Эта задача возникает при дискретизации уравнения Пуассона в прямоугольнике. Легко показать, что здесь матрица  $A$  положительно определенная.

При условии что  $G(i,j) = 0$ , если  $i \in \{0, N+1\}$  или  $j \in \{0, M+1\}$ , один шаг Гаусса–Зейделя с замещением принимает следующую форму:

```

for  $j = 1 : M$ 
    for  $i = 1 : M$ 
         $G(i,j) = (F(i,j) + G(i-1,j) + G(i+1,j) + G(i,j-1) + G(i,j+1)) / 4$ 
    end
end

```

Заметим, что в этой задаче для матрицы  $A$  не требуется никакой памяти.

#### 10.1.4. Последовательная верхняя релаксация

Метод Гаусса–Зейделя очень привлекателен в силу своей простоты. К несчастью, если спектральный радиус для  $M_G^{-1}N_G$  близок к единице, то метод может оказаться непозволительно медленным из-за того, что ошибки стремятся к нулю как  $\rho(M_G^{-1}N_G)^k$ . Чтобы исправить это, возьмем  $\omega \in \mathbb{R}$  и рассмотрим следующую модификацию шага Гаусса–Зейделя:

```

for  $i = 1 : n$ 
     $x_i^{(k+1)} = \omega \left( b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij}x_j^{(k)} \right) / a_{ii} + (1 - \omega)x_i^{(k)}$ 
end

```

(10.1.7)

Так определяется метод последовательной верхней релаксации (SOR – Successive over relaxation). Используя (10.1.4), мы видим, что в матричных обозначениях шаг SOR выглядит как

$$M_\omega x^{(k+1)} = N_\omega x^{(k)} + \omega b, \quad (10.1.8)$$

где  $M_\omega = D + \omega L$  и  $N_\omega = (1 - \omega)D - \omega U$ . Для небольшого числа специфических (но важных) задач, таких, как (10.1.6), значение релаксационного параметра  $\omega$ , минимизирующего  $\rho(M_\omega^{-1}N_\omega)$ , является известным. Более того, в результате  $\rho(M_1^{-1}N_1) = \rho(M_G^{-1}N_G)$  значительно сокращается. В более сложных задачах, однако, для того чтобы определить подходящее  $\omega$ , может возникнуть необходимость в выполнении весьма трудного анализа собственных значений. Полный обзор «теории SOR» появился в Young (1971). Некоторые практические схемы для оценивания оптимального  $\omega$  обсуждаются в O’Leary (1976) и Wachpress (1966).

### 10.1.5. Метод чебышевских полупротераций

Еще один подход к ускорению сходимости итерационного метода извлекает пользу из полиномов Чебышева. Предположим, что посредством итераций  $Mx^{(j+1)} = Nx^{(j)} + b$  получены  $x^{(1)}, \dots, x^{(k)}$  и мы хотим определить коэффициенты  $v_j(k)$ ,  $j = 0 : k$ , такие, что вектор

$$y^{(k)} = \sum_{j=0}^k v_j(k) x^{(j)} \quad (10.1.9)$$

лучше приближает решение, чем  $x^{(k)}$ . Если  $x^{(0)} = \dots = x^{(k)} = x$ , то разумно потребовать, чтобы  $y^{(k)} = x$ . Следовательно, будем считать, что

$$\sum_{j=0}^k v_j(k) = 1. \quad (10.1.10)$$

Посмотрим, как при этом ограничении выбирать  $v_j(k)$  с тем, чтобы минимизировать погрешность для  $y^{(k)}$ .

Вспомним, что согласно теореме 10.1.1  $x^{(k)} - x = (M^{-1}N)^k e^{(0)}$ , где  $e^{(0)} = x^{(0)} - x$ . Мы видим, что

$$y^{(k)} - x = \sum_{j=0}^k v_j(k)(x^{(j)} - x) = \sum_{j=0}^k v_j(k)(M^{-1}N)^j e^{(0)}.$$

Работая с 2-нормами, отсюда получаем

$$\|y^{(k)} - x\|_2 \leq \|p_k(G)\|_2 \|e^{(0)}\|_2, \quad (10.1.11)$$

где  $G = M^{-1}N$  и

$$p_k(z) = \sum_{j=0}^k v_j(k) z^j.$$

Заметим, что условие (10.1.10) влечет за собой  $p_k(1) = 1$ .

Теперь предположим, что матрица  $G$  симметричная и ее собственные значения  $\lambda_i$  удовлетворяют неравенствам  $-1 < \alpha \leq \lambda_i \leq \dots \leq \lambda_1 \leq \beta < 1$ . Отсюда вытекает, что

$$\|p_k(G)\|_2 = \max_{\lambda_i \in \lambda(A)} |p_k(\lambda_i)| \leq \max_{\alpha \leq \lambda \leq \beta} |p_k(\lambda)|.$$

Таким образом, для того чтобы сделать норму  $p_k(G)$  малой, нам нужен полином  $p_k(z)$ , принимающий малые значения на  $[\alpha, \beta]$  и подчиненный ограничению  $p_k(1) = 1$ .

Рассмотрим полиномы Чебышева  $c_j(z)$ , порождаемые рекуррентным соотношением  $c_j(z) = 2zc_{j-1}(z) - c_{j-2}(z)$ , где  $c_0(z) = 1$  и  $c_1(z) = z$ . Эти полиномы удовлетворяют неравенству  $|c_j(z)| \leq 1$  на  $[-1, 1]$ , но быстро растут вне этого интервала. Как следствие, полином

$$p_k(z) = \frac{c_k \left( -1 + 2 \frac{z - \alpha}{\beta - \alpha} \right)}{c_k(\mu)},$$

где

$$\mu = -1 + 2 \frac{1 - \alpha}{\beta - \alpha} = 1 + 2 \frac{1 - \beta}{\beta - \alpha},$$

удовлетворяет условию  $p_k(1) = 1$  и оказывается малым на  $[\alpha, \beta]$ . В силу определения  $p_k(z)$  и уравнения (10.1.11)

$$\|y^{(k)} - x\|_2 \leq \frac{\|x - x^{(0)}\|_2}{|c_k(\mu)|}.$$

Таким образом, чем больше  $\mu$ , тем значительнее ускорение сходимости.

Для того чтобы изложенное выше стало практической процедурой ускорения, для вычисления  $y^{(k)}$  нам нужен метод более эффективный, чем (10.1.9). Мы молчаливо предполагаем, что  $n$  большое и поэтому вновь обращаться к  $x^{(0)}, \dots, x^{(k)}$  при больших  $k$  может быть неудобно или даже невозможно.

К счастью, для  $y^{(k)}$  можно вывести трехчленные рекуррентные соотношения, опираясь на трехчленные рекуррентные соотношения для многочленов Чебышева. Именно, можно показать, что если

$$\omega_{k+1} = 2 \frac{2 - \beta - \alpha}{\beta - \alpha} \frac{c_k(\mu)}{c_{k+1}(\mu)},$$

то

$$\begin{aligned} y^{(k+1)} &= \omega_{k+1} (y^{(k)} - y^{(k-1)} + \gamma z^{(k)}) + y^{(k-1)}, \\ Mz^{(k)} &= b - Ay^{(k)}, \\ \gamma &= 2/(2 - \alpha - \beta), \end{aligned} \tag{10.1.12}$$

где  $y^{(0)} = x^{(0)}$  и  $y^{(1)} = x^{(1)}$ . Об этой схеме мы будем говорить как о чебышевском полуитерационном методе, связанном с  $My^{(k+1)} = Ny^{(k)} + b$ . Для эффективности ускорения нам нужны хорошие нижние и верхние оценки для  $\alpha$  и  $\beta$ . Как и для SOR, установить эти параметры может быть трудно, за исключением небольшого числа специфических задач.

Чебышевские полуитерационные методы всесторонне анализируются в Golub, Varga (1961), а также в Varga (1962, гл. 5).

### 10.1.6. Симметричный метод SOR

При выводе чебышевского ускорения мы предполагали, что итерационная матрица  $G = M^{-1}N$  симметрична. Таким образом, наш простой анализ неприменим в случае SOR-итераций с несимметричной матрицей  $M_\omega^{-1}N_\omega$ . Однако есть возможность симметризовать метод SOR, сделав его пригодным к применению чебышевского ускорения. Идея состоит в сочетании метода SOR и схемы SOR с обратным ходом:

**for**  $i := n : -1 : 1$

$$x_i^{(k+1)} = \omega \left( b - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right) / a_{ii} + (1 - \omega) x_i^{(k)} \quad (10.1.13)$$

**end**

Эти итерации получаются, когда неизвестные в (10.1.7) перевычисляются в обратном порядке. Обратный метод SOR можно описать в матричных терминах, используя (10.1.4). Именно, имеем  $\tilde{M}_\omega x^{(k+1)} = \tilde{N}_\omega x^{(k)} + \omega b$ , где

$$\tilde{M}_\omega = D + \omega U, \quad \tilde{N}_\omega = (1 - \omega)D - \omega L. \quad (10.1.14)$$

Если  $A$  симметричная ( $U = L^T$ ), то  $\tilde{M}_\omega = M_\omega^T$  и  $\tilde{N}_\omega = N_\omega^T$  и мы получаем итерации

$$\begin{aligned} M_\omega x^{(k+1/2)} &= N_\omega x^{(k)} + \omega b, \\ M_\omega^T x^{(k+1)} &= N_\omega^T x^{(k+1/2)} + \omega b. \end{aligned} \quad (10.1.15)$$

Ясно, что  $G = M_\omega^{-1} N_\omega^T M_\omega^{-1} N_\omega$  представляет собой итерационную матрицу для этого метода. Из определений  $M_\omega$  и  $N_\omega$  следует, что

$$G = M^{-1} N \equiv (M_\omega D^{-1} M_\omega^T)^{-1} (N_\omega^T D^{-1} N_\omega). \quad (10.1.16)$$

Если  $D$  имеет положительные диагональные элементы и  $KK^T = (N_\omega^T D^{-1} N_\omega)$  – разложение Холецкого, то  $K^T G K^{-T} = K^T (M_\omega D^{-1} M_\omega^T)^{-1} K$ . Таким образом,  $G$  подобна симметричной матрице и имеет вещественные собственные значения.

Итерационный метод (10.1.15) называется методом симметричной последовательной верхней релаксации (SSOR). Он часто используется вместе с чебышевским полуитерационным ускорением.

### Задачи

**10.1.1.** Показать, что итерации Якоби можно записать в виде  $x^{(k+1)} = x^{(k)} + Hr^{(k)}$ , где  $r^{(k)} = b - Ax^{(k)}$ . Получить аналогичное утверждение для итераций Гаусса – Зейделя.

**10.1.2.** Показать, что если  $A$  обладает строгим диагональным преобладанием, то итерации Гаусса – Зейделя сходятся.

**10.1.3.** Показать, что итерации Якоби сходятся для симметричных положительно определенных  $2 \times 2$ -систем.

**10.1.4.** Показать, что если матрица  $A = M - N$  вырожденная, то нельзя получить  $\rho(M^{-1}N) < 1$  ни для какой невырожденной  $M$ .

**10.1.5.** Доказать (10.1.16).

**10.1.6.** Доказать обращение теоремы 10.1.1. Другими словами, показать, что если итерации  $Mx^{(k+1)} = Nx^{(k)} + b$  всегда сходятся, то  $\rho(M^{-1}N) < 1$ .

**10.1.7.** (R. S. Varga). Предположим, что  $A_1 = \begin{bmatrix} 1 & -1/2 \\ -1/2 & 1 \end{bmatrix}$  и  $A_2 = \begin{bmatrix} 1 & -3/4 \\ -1/12 & 1 \end{bmatrix}$ .

Пусть  $J_1$  и  $J_2$  – соответствующие итерационные матрицы метода Якоби. Показать, что  $\rho(J_1) > \rho(J_2)$ , т. е. опровергнув заявление о том, что усиление диагонального преобладания влечет за собой более быструю сходимость метода Якоби.

### Замечания и литература к § 10.1

С намного большей основательностью, чем здесь, обзор области итерационных методов дают три исчерпывающих тома:

Hageman L. A. and Young D. M. (1981). Applied Iterative Methods, Academic Press, New York.

[Есть перевод: Хейгеман Л., Янг Д. Прикладные итерационные методы.–М.: Мир, 1986.]

Varga R. S. (1962). Matrix Iterative Analysis. Prentice-Hall, Englewood Cliffs, New Jersey.

Young D. M. (1971). Iterative Solution of Large Systems, Academic Press, New York.

Мы также настоятельно рекомендуем гл. 7 из

Ortega J. (1972). Numerical Analysis: A Second Course, Academic Press, New York.

Часто предпочтение отдается прямому (неитерационному) решению больших разреженных систем. См.

Duff I. S., Erisman A. M. and Reid J. K. (1986). Direct Methods for Sparse Matrices, Oxford University Press.

George J. A. and Liu J. W. (1981). Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ. [Имеется перевод: Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений.—М.: «Мир», 1984.]

Мы видели, что число обусловленности  $k(A)$  имеет важное значение, когда к системе  $Ax = b$  применяются прямые методы. Однако обусловленность системы имеет отношение и к эффективности итерационного метода. См.

Arioli M. and Romani F. (1985) "Relations Between Condition Numbers and the Convergence of the Jacobi Method for Real Positive Definite Matrices", Numer. Math. 46, 31–42.

Как уже отмечалось, наиболее полное изложение метода SOR дает Young (1971). Предметом «теории SOR» является выработка рекомендаций по выбору параметра релаксации  $\omega$ . При этом существенно, как упорядочены уравнения и неизвестные. См.

Bernal M. J. M. and Verner J. H. (1968). "On generalizing of the Theory of Consistent Orderings for Successive Over-Relaxation Methods". Numer. Math. 12, 215–22.

Nicolaides R. A. (1974). "On a Geometrical Aspect of SOR and the Theory of Consistent Ordering for Positive Definite Matrices", Numer. Math. 12, 99–104.

Young D. M. (1970). "Convergence Properties of the Symmetric and Unsymmetric Over-Relaxation Methods", Math. Comp. 2, 793–807.

Young D. M. (1972). "Generalization of Property A and Consistent Ordering", SIAM J. Num. Anal. 9, 454–63.

Интересны также эвристические методы для оценки  $\omega$ . См.

O'Leary D. P. (1976). "Hybrid Conjugate Gradient Algorithms", Report STAN-CS-76-548, Department of Computer Science, Stanford University (Ph. D. thesis).

Wachpress E. L. (1966). Iterative Solution of Elliptic Systems, Prentice-Hall, Englewood Cliffs, New Jersey.

Анализ чебышевских полуитераций появился в

Golub G. H. and Varga R. S. (1961). "Chebyshev Semi-Iterative Methods, Successive Over-Relaxation Iterative Methods, and Second-Order Richardson Iterative Methods, Parts I and II", Numer. Math. 3, 157–68.

Эта работа использует предположение о вещественности собственных значений соответствующей итерационной матрицы. Как действовать, когда это не так, обсуждается в

Eiermann M. and Niethammer W. (1983). "On the Construction of Semi-iterative Methods" SIAM J. Numer. Anal. 20, 1153–1160.

Golub G. H. and Overton M. (1988). "The Convergence of Inexact Chebyshev and Richardson Iterative Methods for Solving Linear Systems", Numer. Math. 53, 571–594.

Manteuffel T. A. (1977). "The Chebyshev Iteration for Nonsymmetric Linear Systems", Numer. Math. 28, 307–27.

Niethammer W. and Varga R. S. (1983). "The Analysis of k-step Iterative Methods for Linear Systems from Summability Theory", Numer. Math. 41, 177–206.

Иногда есть возможность «симметризации» итерационного метода, влекущей за собой упрощение процесса ускорения, поскольку все собственные значения здесь вещественные. Так обстоит дело в случае метода SSOR, обсуждаемого в

Sheldon J. W. (1955). "On the Numerical Solution of Elliptic Difference Equations", Math. Tables Aids Comp. 9, 101–12.

Некоторое внимание уделялось параллельной реализации классических итераций. См.

Evans D. J. (1984). "Parallel SOR Iterative methods", Parallel Computing 1, 3–18.

- Johnsson S. L. and Ho C. T. (1987). "Multiple Tridiagonal Systems, the Alternating Direction Methods, and Boolean Cube Configured Multiprocessors", Report YALEU DCS RR-532, Dept. of Computing Science, Yale University, New Haven, CT.
- Patel N. and Jordan H. (1984). "A Parallelized Point Rowwise Successive Over-Relaxation Method on a Multiprocessor", Parallel Computing 1, 207–222.
- Plemmons R. J. (1986). "A Parallel Block Iterative Scheme Applied to Computations in Structural Analysis", SIAM J. Alg. and Disc. Methods 7, 337–347.

### Добавление при переводе к § 10.1

Обстоятельное изложение основных итерационных методов и их применений можно найти в следующих отечественных монографиях:

- Марчук Г. И., Кузнецов Ю. А. (1972). Итерационные методы и квадратичные функционалы.– Новосибирск: Наука.
- Самарский А. А., Николаев Е. С. (1978). Методы решения сеточных уравнений.– М.: Наука.
- Воеводин В. В., Кузнецов Ю. А. (1984). Матрицы и вычисления.– М.: Наука.
- Марчук Г. И. (1986). Методы расщепления и переменных направлений.– М.: ОВМ АН СССР.
- Перевод
- Дьяконов Е. Г. (1989). Минимизация вычислительной работы.– М.: Наука.

## 10.2. Метод сопряженных градиентов

Трудность, связанная с SOR, чебышевскими полуитерациями и такого же типа методами, заключается в том, что они зависят от параметров, правильный выбор которых иногда бывает затруднителен. Например, для того чтобы чебышевское ускорение было успешным, нам нужны хорошие оценки для наибольшего и наименьшего собственных значений соответствующей итерационной матрицы  $M^{-1}N$ . Если эта матрица не устроена по-особому, то получение их в аналитическом виде, скорее всего, невозможно, а вычисление дорого.

В этом разделе мы представим метод, в котором нет этой трудности,— хорошо известный метод сопряженных градиентов Хестенса–Штифеля. Мы вывели этот метод в § 9.3.1 при обсуждении – как приспособить алгоритм Ланцоша к решению симметричных положительно определенных линейных систем. Здесь мы выводим метод сопряженных градиентов заново для того, чтобы подготовить почву для § 10.3, где будет показана его связь с другими итерационными схемами для  $Ax = b$  и будут описаны некоторые из полезных его обобщений.

### 10.2.1. Наискорейший спуск

Выход метода мы начнем с того, что обсудим, как может происходить рыскание при минимизации функционала  $\varphi(x)$ , имеющего вид

$$\varphi(x) = \frac{1}{2}x^T Ax - x^T b,$$

где  $b \in \mathbb{R}^n$  и матрица  $A \in \mathbb{R}^{n \times n}$  предполагается положительно определенной и симметричной. Минимальное значение  $\varphi$  равно  $-b^T A^{-1} b / 2$  и достигается при  $x = A^{-1} b$ . Таким образом, минимизация  $\varphi$  и решение системы  $Ax = b$  – эквивалентные задачи.

Одной из самых простых стратегий минимизации  $\varphi$  является метод наискорейшего спуска. В текущей точке  $x_c$  функция  $\varphi$  убывает наиболее быстро в направлении антиградиента  $-\nabla\varphi(x_c) = b - Ax_c$ . Мы называем

$$r_c = b - Ax_c$$

невязкой вектора  $x_c$ . Если невязка ненулевая, то  $\varphi(x_c + ar_c) < \varphi(x_c)$  для некоторого положительного  $a$ . В методе наискорейшего спуска (с точной минимизацией на прямой) мы берем  $a = r_c^T r_c / r_c^T A r_c$ , дающее минимум для  $\varphi(x_c + ar_c)$ . Получаем

```

 $k = 0; x_0 = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 
     $a_k = r_{k-1}^T r_{k-1} / r_{k-1}^T A r_{k+1}$ 
     $x_k = x_{k-1} + a_k r_{k-1}$ 
     $r_k = b - Ax_k$ 
end

```

(10.2.1)

Выбор начального вектора в виде  $x_0 = 0$  не является ограничением. Если  $\tilde{x}$  считается более подходящим начальным приближением, то мы просто применим (10.2.1), заменив  $b$  на  $b - A\tilde{x}$ . В этом случае искомое решение аппроксимируется векторами  $x_k + \tilde{x}$ .

Глобальная сходимость метода наискорейшего спуска является следствием легко устанавливаемого неравенства

$$\varphi(x_k) + \frac{1}{2} b^T A^{-1} b \leq \left(1 - \frac{1}{k_2(A)}\right) (\varphi(x_{k-1}) + \frac{1}{2} b^T A^{-1} b). \quad (10.2.2)$$

К несчастью, скорость сходимости может быть недопустимо медленной, если  $k_2(A) = \lambda_1(A)/\lambda_n(A)$  большое. В этом случае линии уровня для  $\varphi$  являются сильно вытянутыми гиперэллипсоидами, а минимизация соответствует поиску самой нижней точки на относительно плоском дне крутого оврага. При наискорейшем спуске мы вынуждены переходить с одной стороны оврага на другую вместо того, чтобы спуститься к его дну. Направления градиента, возникающие при итерациях, являются слишком близкими; это и замедляет продвижение к точке минимума.

## 10.2.2. Произвольные направления спуска

Чтобы избежать ловушек при наискорейшем спуске, мы рассмотрим последовательную минимизацию  $\varphi$  вдоль какого-либо множества направлений  $\{p_1, p_2, \dots\}$ , которые не обязаны соответствовать невязкам  $\{r_0, r_1, \dots\}$ . Легко показать, что минимум  $\varphi(x_{k-1} + ap_k)$  по  $a$  дает

$$a = a_k = p_k^T r_{k-1} / p_k^T A p_k.$$

При таком выборе, как легко видеть, получаем

$$\varphi(x_{k-1} + a_k p_k) = \varphi(x_{k-1}) - \frac{1}{2} (p_k^T r_{k-1})^2 p_k^T A p_k. \quad (10.2.3)$$

Таким образом, для того чтобы обеспечить уменьшение  $\varphi$ , мы должны потребовать, чтобы  $p_k$  не был ортогонален к  $r_{k-1}$ . Это приводит к следующей стратегии минимизации:

```

 $k = 0; x_0 = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 
    Выбрать направление  $p_k$  так, чтобы  $p_k^T r_{k-1} \neq 0$ 
     $a_k = p_k^T r_{k-1} / p_k^T A p_k$ 
     $x_k = x_{k-1} + a_k p_k$ 
     $r_k = b - Ax_k$ 
end

```

(10.2.4)

Заметим, что  $x_k \in \text{span}\{p_1, \dots, p_k\}$ . Проблема, конечно, состоит в том, как выбирать эти векторы с тем, чтобы гарантировать глобальную сходимость и в то же время обойти ловушки наискорейшего спуска.

### 10.2.3. A-сопряженные направления спуска

По-видимому, идеальный подход был бы связан с выбором линейно независимых  $p_i$  с тем, чтобы каждое  $x_i$  было решением задачи

$$\min_{x \in \text{span}\{p_1, \dots, p_k\}} \varphi(x). \quad (10.2.5)$$

Это гарантировало бы не только глобальную сходимость, но и конечность процесса, так как должно быть  $Ax_n = b$ . Заметим, что то, что нам нужно, это такой вектор  $p_k$ , для которого решение одномерной задачи минимизации

$$\min_a \varphi(x_{k-1} + ap_k)$$

одновременно дает решение для  $k$ -мерной задачи минимизации (10.2.5). Это серьезное ограничение, но, оказывается, этого можно добиться.

Пусть  $P_k = [p_1, \dots, p_k] \in \mathbb{R}^n \times k$  является матрицей направлений спуска. Если  $x \in \text{range}(p_k)$ , то  $x = P_{k-1}y + ap_k$  для некоторых  $y \in \mathbb{R}^{k-1}$  и  $a \in \mathbb{R}$ . Для  $x$  такого вида легко показать, что

$$\varphi(x) = \varphi(P_{k-1}y) + ay^T P_{k-1}^T Ap_k + \frac{a^2}{2} p_k^T Ap_k - ap_k^T b.$$

Присутствие «смешанного члена»  $ay^T P_{k-1}^T Ap_k$  усложняет минимизацию. Без него задача минимизации  $\varphi$  по  $\text{range}(P_k)$  расщепилась бы на задачу минимизации по  $\text{range}(P_{k-1})$ , решение которой  $x_{k-1}$  предполагается известным, и простую минимизацию для скаляра  $a$ . Один из способов добиться этого расщепления – потребовать, чтобы направление  $p_k$  было  $A$ -сопряженным по отношению к  $p_1, \dots, p_{k-1}$ , т. е.

$$P_{k-1}^T Ap_k = 0. \quad (10.2.6)$$

Если это сделать и определить  $x_{k-1} \in \text{range}(P_{k-1})$  и  $a_k \in \mathbb{R}$  соотношениями

$$\begin{aligned} \varphi(x_{k-1}) &= \min_y \varphi(P_{k-1}y), \\ a_k &= p_k^T b / p_k^T Ap_k, \end{aligned}$$

то задача

$$\min_{y, a} \varphi(P_{k-1}y + a_k p_k) = \min_y \varphi(P_{k-1}y) + \min_a \left\{ \frac{a^2}{2} p_k^T Ap_k - ap_k^T b \right\}$$

будет решена, если взять  $P_{k-1}y = x_{k-1}$  и  $a = a_k$ . Поскольку  $x_{k-1}$  есть линейная комбинация векторов  $p_1, \dots, p_{k-1}$ , отсюда вытекает, что  $p_k^T Ax_{k-1} = 0$  и, следовательно,

$$a_k = p_k^T r_{k-1} / p_k^T Ap_k,$$

т. е. рецепт для  $a_k$  точно такой же, как в (10.2.4). Таким образом, получаем

```

 $k = 0; x_0 = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 

```

```

Выбрать  $p_k \in \text{span}\{Ap_1, \dots, Ap_{k-1}\}^\perp$  так, чтобы  $p_k^T r_{k-1} \neq 0.$  (10.2.7)
 $a_k = p_k^T r_{k-1} / r_{k-1}^T A r_{k-1}$ 
 $x_k = x_{k-1} + a_k p_k$ 
 $r_k = b - Ax_k$ 
end

```

Здесь вектор  $x_k = x_{k-1} + a_k p_k$  минимизирует  $\varphi$  на линейной оболочке направлений спуска, т. е. на подпространстве  $\text{span}\{p_1, \dots, p_k\}.$

#### 10.2.4. Метод сопряженных градиентов

Чтобы выполнить (10.2.7), мы должны быть уверены, что возможен выбор  $p_k,$  при котором  $p_{k-1}^T Ap_k = 0$  и  $p_k^T r_{k-1} \neq 0.$  Предположим, что векторы  $p_1, \dots, p_{k-1}$  ненулевые и удовлетворяют условиям  $p_i^T Ap_j = 0$  при  $i \neq j.$  Поскольку  $x_{k-1} \in \text{span}\{p_1, \dots, p_{k-1}\},$  отсюда вытекает, что для любого  $p \in \mathbb{R}^n$  мы имеем  $p^T r_{k-1} = p^T b - p^T A p_{k-1} z$  для какого-то  $z \in \mathbb{R}^{k-1}.$  Если  $p^T r_{k-1} = 0$  для всех  $p,$  являющихся  $A$ -сопряженными с  $p_1, \dots, p_{k-1},$  то  $b \in \text{span}\{Ap_1, \dots, Ap_{k-1}\}.$  А это влечет за собой  $x = A^{-1}b \in \text{span}\{p_1, \dots, p_{k-1}\}$  и, значит,  $x_{k-1} = x,$  т. е.  $r_{k-1} \neq 0.$  Следовательно, если  $r_{k-1} \neq 0,$  то можно найти ненулевой вектор  $p_k,$   $A$ -сопряженный с  $p_1, \dots, p_{k-1}$  и удовлетворяющий условию  $p_k^T r_{k-1} \neq 0.$

Поскольку наша цель – осуществить быстрое сокращение величины невязок, естественно выбирать в качестве  $p_k$  вектор, который ближе всего к  $r_{k-1}$  среди векторов,  $A$ -сопряженных с  $p_1, \dots, p_{k-1}.$  Так мы получаем «нулевую версию» метода сопряженных градиентов:

```

k = 0; x_0 = 0; r_0 = b
while r_k != 0
    k = k + 1
    if k = 1
        p_1 = r_0
    else
        Пусть  $p_k$  минимизирует  $\|p - r_{k-1}\|_2$ 
        среди всех векторов  $p \in \text{span}\{Ap_1, \dots, Ap_{k-1}\}^T.$ 
    end
     $a_k = p_k^T r_{k-1} / p_k^T A p_k$ 
     $x_k = x_{k-1} + a_k p_k$ 
     $r_k = b - Ax_k$ 
end
x = x_k

```

(10.2.8)

Конечность этого алгоритма обеспечивается тем, что векторы  $p_k$  ненулевые и, будучи ненулевыми  $A$ -сопряженными векторами, линейно независимы. Более точно, или  $r_{k-1} = 0$  для некоторого  $k \leq n,$  или мы вычислим  $x_n,$  минимизирующий  $\varphi$  на  $\text{span}\{p_1, \dots, p_n\} = \mathbb{R}^n.$

#### 10.2.5. Несколько крайне необходимых наблюдений

Чтобы сделать (10.2.8) эффективной процедурой для решения разреженных систем  $Ax = b,$  нам нужен эффективный метод вычисления  $p_k.$  В качестве первого шага в этом направлении изучим свойства  $p_k$  как решения определенной задачи наименьших квадратов.

**Лемма 10.2.1.** Для  $k \geq 2$  векторы  $p_k$ , получаемые согласно (10.2.8), удовлетворяют

соотношению

$$p_k = r_{k-1} - AP_{k-1}z_{k-1},$$

где  $z_{k-1}$  есть решение задачи  $\min_z \|r_{k-1} - AP_{k-1}z\|_2$ .

**Доказательство.** Предположим, что  $z_{k-1}$  – есть решение приведенной выше задачи наименьших квадратов и  $p = r_{k-1} - AP_{k-1}z_{k-1}$  – минимальная невязка. Отсюда получаем  $p^T AP_{k-1} = 0$ . Более того, поскольку  $p = [I - (AP_{k-1})(AP_{k-1})^+]r_{k-1}$  – ортогональная проекция вектора  $r_{k-1}$  на  $\text{range}(AP_{k-1})$ ,  $p$  является ближайшим вектором к  $r_{k-1}$  на  $\text{range}(AP_{k-1}^\perp)$ . Таким образом,  $p = p_{k-1}$ .  $\square$

Используя этот результат, мы можем установить несколько важных соотношений между невязками  $r_i$  и направлениями спуска  $p_i$ .

**Теорема 10.2.2.** После  $k$  итераций метода сопряженных направлений имеем

$$r_j = r_{j-1} - \alpha_j Ap_j, \quad (10.2.9)$$

$$P_j^T r_j = 0, \quad (10.2.10)$$

$$\begin{aligned} \text{span}\{p_1, \dots, p_j\} &= \text{span}\{r_0, \dots, r_{j-1}\} \\ &= \text{span}\{b, Ab, \dots, A^{j-1}b\}. \end{aligned} \quad (10.2.11)$$

**Доказательство.** Уравнение (10.2.9) получается, если подействовать матрицей  $A$  на обе части равенства  $x_j = x_{j-1} + \alpha_j p_j$  и учесть определение невязки.

Чтобы установить (10.2.10), заметим, что если  $(P_j^T AP_j)y_j = P_j^T b$ , то  $y_j$  минимизирует

$$\varphi(P_j y) = \frac{1}{2} y^T (P_j^T AP_j) y - y^T P_j^T b.$$

Таким образом,  $P_j^T AP_j y_j = P_j^T b$ , и поэтому  $x_j = P_j y_j$  и  $P_j^T r_j = P_j^T(b - AP_j y_j) = 0$ .

Чтобы доказать соотношение (10.2.11), необходима индукция. При  $j = 1$  оно очевидно верно. Предположим, что оно выполняется для некоторого  $j$ , удовлетворяющего  $1 \leq j < k$ . Вследствие (10.2.9)  $r_j \in \text{span}\{b, Ab, \dots, A^j b\}$  и, значит,  $P_{j+1} = r_j - AP_j z_j \in \text{span}\{b, Ab, \dots, A^j b\}$ . Оба пространства  $\text{span}\{r_0, \dots, r_j\}$  и  $\text{span}\{p_1, \dots, p_{j+1}\}$  имеют размерность  $j+1$ , вследствие чего они должны совпадать с  $\text{span}\{b, Ab, \dots, A^j b\}$ .  $\square$

Теперь мы установим взаимную ортогональность невязок  $r_j$ .

**Теорема 10.2.3.** После  $k-1$  шагов алгоритма сопряженных градиентов (10.2.8) невязки  $r_0, \dots, r_{k-1}$  взаимно ортогональны.

**Доказательство.** Из уравнения  $x_i = x_{i-1} + \alpha_i p_i$  вытекает, что  $r_i = r_{i-1} - \alpha_i Ap_i$ . Таким образом, по лемме 10.2.1, для  $j = 1:k$

$$p_j = r_{j-1} - [Ap_1, \dots, Ap_{j-1}]z_{j-1} \in \text{span}\{r_0, \dots, r_{j-1}\}.$$

Следовательно,  $[p_1, \dots, p_k] = [r_0, \dots, r_{k-1}]T$  для некоторой верхней треугольной матрицы  $T \in \mathbb{R}^{k \times k}$ . Поскольку векторы  $p_i$  независимы, матрица  $T$  невырожденная и поэтому

$$[r_0, \dots, r_{k-1}] = [p_1, \dots, p_k]T^{-1}.$$

Как следствие  $r_{j-1} \in \text{span}\{p_1, \dots, p_j\}$  для  $j = 1:k$ . Но в силу (10.2.10)  $P_{k-1}^T r_{k-1} = 0$ . Поскольку  $r_0, \dots, r_{k-2} \in \text{span}\{p_1, \dots, p_{k-1}\}$ , мы должны заключить, что  $r_j r_{k-1} = 0$  для  $j = 0:k-2$ .

Теперь покажем, каким образом  $p_k$  выражается в виде линейной комбинации  $p_{k-1}$

и  $r_{k-1}$ . Представим вектор  $z_{k-1}$  из леммы 10.2.1 в блочном виде

$$z_{k-1} = \begin{bmatrix} w \\ \mu \end{bmatrix} \quad \begin{matrix} k-2 \\ 1 \end{matrix}$$

и воспользуемся тождеством  $r_{k-1} = r_{k-2} - \alpha_{k-1} A p_{k-1}$ . Мы видим, что

$$p_k = r_{k-1} - A P_{k-1} z_{k-1} = r_{k-1} - A P_{k-2} w - \mu A p_k = \left(1 + \frac{\mu}{\alpha_k}\right) r_{k-1} + s_{k-1},$$

где

$$s_{k-1} = -\frac{\mu}{\alpha_{k-1}} r_{k-2} - A P_{k-2} w.$$

Вследствие взаимной ортогональности невязок  $r_i$  векторы  $s_{k-1}$  и  $r_{k-1}$  ортогональны. Таким образом, задача наименьших квадратов из леммы 10.2.1 сводится к выбору  $\mu$  и  $w$ , таких, что величина

$$\|p_k\|_2^2 = \left(1 + \frac{\mu}{\alpha_{k-1}}\right)^2 \|r_{k-1}\|_2^2 + \|s_{k-1}\|_2^2$$

минимальна. Поскольку 2-норма для  $r_{k-2} - A P_{k-2} z$  имеет минимум в точке  $z_{k-2}$ , дающей невязку  $p_{k-1}$ , отсюда вытекает, что  $s_{k-1}$  есть кратное вектора  $p_{k-1}$ . Следовательно,  $p_k \in \text{span}\{r_{k-1}, p_{k-1}\}$ . Не ограничивая общности, мы можем считать, что  $p_k = r_{k-1} + \beta_k p_{k-1}$ , и так как  $p_{k-1}^T A p_k = 0$  и  $p_{k-1}^T r_{k-1} = 0$ , отсюда вытекает, что

$$\beta_k = \frac{p_{k-1}^T A r_{k-1}}{p_{k-1}^T A p_{k-1}}, \quad \alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_k^T A p_k}.$$

Это приводит к «версии 1» метода сопряженных градиентов:

```

 $k = 0; x_0 = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 
    if  $k \neq 1$ 
         $p_1 = r_0$ 
    else
         $\beta_k = -p_{k-1}^T A r_{k-1} / p_{k-1}^T A p_{k-1}$ 
         $p_k = r_{k-1} + \beta_k p_{k-1}$ 
    end
     $\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$ 
     $x_k = x_{k-1} + \alpha_k p_k$ 
     $r_k = b - Ax_k$ 
end
 $x = x_k$ 

```

(10.2.12)

Кажется, что эти итерации – в том виде, как они записаны – требуют трех различных умножений матрицы на вектор на каждый проход цикла. Однако если вычислять невязки с помощью рекурсии  $r_k = r_{k-1} - \alpha_k A p_k$  и подставить

$$r_{k-1}^T r_{k-1} = -\alpha_{k-1} r_{k-1}^T A p_{k-1}, \quad (10.2.13)$$

$$r_{k-2}^T r_{k-2} = \alpha_{k-1} p_{k-1}^T A p_{k-1} \quad (10.2.14)$$

в формулу для  $\beta_k$ , то у нас получится следующая более эффективная реализация метода.

**Алгоритм 10.2.1 (Сопряженные градиенты).** Для симметричной положительной определенной матрицы  $A \in \mathbb{R}^{n \times n}$  и вектора  $b \in \mathbb{R}^n$  следующий алгоритм вычисляет решение  $x \in \mathbb{R}^n$  системы  $Ax = b$ .

```

 $k = 0; x_0 = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 
    if  $k = 1$ 
         $p_1 = r_0$ 
    else
         $\beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}$ 
         $p_k = r_{k-1} + \beta_k p_{k-1}$ 
    end
     $\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$ 
     $x_k = x_{k-1} + \alpha_k p_k$ 
     $r_k = r_{k-1} - \alpha_k A p_k$ 
end
 $x = x_k$ 
```

Эта процедура и есть по существу алгоритм сопряженных градиентов, появившийся в оригинальной работе Хестенса и Штифеля (Hestenes, Stiefel (1952)). Детали, связанные с его практической реализацией, обсуждаются в § 10.3. Но уже здесь укажем на то, что каждая итерация требует лишь одного умножения матрицы на вектор.

### 10.2.6. Связь с алгоритмом Ланцоша

В § 9.3.1 мы вывели метод сопряженных градиентов из алгоритма Ланцоша. Теперь давайте проследим за связями между этими двумя алгоритмами в обратном порядке, «выводя» процесс Ланцоша из сопряженных градиентов. Определим матрицу невязок  $R_k \in \mathbb{R}^{n \times n}$  как  $R_k = [r_0, \dots, r_{k-1}]$  и верхнюю двухдиагональную матрицу  $B_k \in \mathbb{R}^{k \times k}$  как

$$B_k = \begin{bmatrix} 1 & -\beta_2 & \cdots & 0 \\ & 1 & -\beta_3 & \vdots \\ & \ddots & \ddots & \ddots \\ & \vdots & \ddots & \ddots & -\beta_k \\ 0 & \cdots & & & 1 \end{bmatrix}.$$

Из уравнений  $p_j = r_{j-1} + \beta_j p_{j-1}$ ,  $j = 2 : k$  и равенства  $p_1 = r_0$  следует, что  $R_k = P_k B_k$ . Как следствие  $A$ -сопряженности столбцов матрицы  $P_k = [p_1, \dots, p_k]$  мы видим, что матрица  $R_k^T A R_k = B_k^T \text{diag}(p_1^T A p_1, \dots, p_k^T A p_k) B_k^T$  трехдиагональная. Из (10.2.11) и теоремы 10.2.3 вытекает, что если

$$\Delta = \text{diag}(p_0, \dots, p_{k-1}), \quad \rho_i = \|r_i\|_2,$$

то столбцы в  $R_k \Delta^{-1}$  образуют ортонормированный базис подпространства  $\text{span}\{b, Ab, \dots, A^{k-1}b\}$ . Следовательно, столбцы этой матрицы являются, по сути, векторами Ланцоша из алгоритма 9.3.1, т. е.  $q_i = \pm r_{i-1} / \rho_{i-1}$ , где  $i = 1 : k$ . Более того, отвечающая этим ланцошевым векторам трехдиагональная матрица задается формулой

$$T_k = \Delta^{-1} B_k^T \text{diag}(p_i^T p_i) B_k \Delta^{-1}. \quad (10.2.16)$$

Диагональные и поддиагональные элементы этой матрицы включают величины, которые легко извлекаются из итераций сопряженных градиентов. Таким образом, порождая векторы  $x_k$  из алгоритма 10.2.1, мы можем получить хорошие оценки крайних собственных значений (и числа обусловленности) матрицы  $A$ .

### 10.2.7. Некоторые практические детали

Критерий окончания итераций в алгоритме 10.2.1 нереалистичен. Ошибки округления приводят к тому, что невязки теряют ортогональность, а конечность процесса не гарантируется математически. Более того, когда применяется метод сопряженных градиентов, то обычно  $n$  настолько большое, что в  $O(n)$  итерациях объем работы неприемлемо большой. Как следствие этих наблюдений метод принято рассматривать как какую-то подлинно итерационную технику с критерием останова, основанном на максимуме  $k_{\max}$  для числа итераций и норме невязки. Это приводит к следующей практической версии алгоритма 10.2.1:

```

 $k = 0; x = 0; r = b; \rho_0 = \|r\|_2^2$ 
while  $\sqrt{\rho_k} > \varepsilon \|b\|_2 \wedge k < k_{\max}$ 
     $k = k + 1$ 
    if  $k = 1$ 
         $p = r$ 
    else
         $\beta_k = \rho_{k-1}/\rho_{k-2}$ 
         $p = r + \beta_k p$ 
    end
     $w = Ap$ 
     $a_k = \rho_{k-1}/p^T w$ 
     $x = x + a_k p$ 
     $r = r - a_k w$ 
     $\rho_k = \|r\|_2^2$ 
end
```

Этот алгоритм требует одно умножение матрицы на вектор и  $10n$  флоков на одну итерацию. Заметим, что по существу нужна память только для четырех  $n$ -векторов:  $x, r, p$  и  $w$ . Индексация скаляров<sup>1)</sup> не является необходимой и оставлена здесь только для того, чтобы облегчить сравнение с алгоритмом 10.2.1.

Критерий останова может также опираться на эвристические оценки для погрешности  $A^{-1}r_k$ , получаемые с помощью аппроксимации  $\|A^{-1}\|_2$  обратной величиной к наименьшему собственному значению трехдиагональной матрицы  $T_k$  вида (10.2.16).

Идея рассматривать сопряженные градиенты как итерационный метод берет начало в Reid (1971b). Итерационная точка зрения на метод полезна, но в этом случае центральное значение для успешного применения имеет скорость сходимости.

### 10.2.8. Сходимость метода

Этот параграф мы завершим изучением сходимости итераций  $\{x_k\}$  метода сопряженных градиентов. Будут приведены два результата, и оба они говорят о том, что метод работает хорошо, когда матрица  $A$  близка к единичной – либо по норме, либо в том смысле, что отклонение имеет малый ранг.

<sup>1)</sup> Тем не менее важно как-то различать  $\rho_{k-1}$  и  $\rho_{k-2}$ . – Прим. перев.

**Теорема 10.2.4.** Если  $A = I + B$  – симметричная положительно определенная  $n \times n$ -матрица и  $\text{rank}(B) = r$ , то алгоритм 10.2.1 сходится не более чем за  $r$  шагов.

**Доказательство.** Если  $S_k = \text{span}\{b, Ab, \dots, A^{k-1}b\}$ , то из равенства  $\text{rank}(A - I) = r$  вытекает, что  $\dim(S_k) \leq r$  для всех  $k$ . Поскольку  $\text{span}\{p_1, \dots, p_k\} = S_k$  и векторы  $p_i$  независимы, отсюда следует, что итерации не могут продолжаться более чем  $r$  шагов.  $\square$

Из теоремы 10.2.4 вытекает важная метатеорема:

- Если  $A$  близка к  $r$ -ранговой модификации единичной матрицы, то алгоритм 10.2.1 почти сходится за  $r$  шагов.

В следующем параграфе мы покажем, как используется это эвристическое утверждение.

Оценку погрешности в другом стиле можно получить в терминах  $A$ -нормы, которая определяется следующим образом:

$$\|w\|_A = \sqrt{w^T Aw}.$$

**Теорема 10.2.5.** Предположим, что матрица  $A \in \mathbb{R}^{n \times n}$  симметричная положительно определенная и  $b \in \mathbb{R}^n$ . Пусть алгоритм 10.2.1 вырабатывает итерации  $\{x_k\}$  и  $k = k_2(A)$ . Тогда

$$\|x - x_k\|_A \leq 2 \|x - x_0\|_A \left( \frac{\sqrt{k} - 1}{\sqrt{k} + 1} \right)^k.$$

**Доказательство.** См. Luenberger (1973, р. 187).  $\square$

Точность приближений  $\{x_k\}$  часто намного лучше, чем та, что предсказывается этой теоремой. Тем не менее оказывается очень полезной эвристическая версия теоремы 10.2.5:

- Метод сопряженных градиентов сходится очень быстро по  $A$ -норме, если  $k_2(A) \approx 1$ .

В следующем параграфе мы покажем, каким образом мы иногда можем превратить заданную задачу  $Ax = b$  в связанную с ней задачу  $\tilde{A}\tilde{x} = \tilde{b}$ , где  $\tilde{A}$  близка к единичной матрице.

### Задачи

**10.2.1.** Проверить, что в (10.2.1) невязки удовлетворяют соотношению  $r_i^T r_j = 0$  при  $j = i + 1$ .

**10.2.2.** Проверить (10.2.2).

**10.2.3.** Проверить (10.2.3).

**10.2.4.** Проверить (10.2.13) и (10.2.14).

**10.2.5.** Дать формулы для элементов трехдиагональной матрицы  $T_k$  вида (10.2.16).

**10.2.6.** Сравнить требуемые объем вычислений и память для практических реализаций алгоритмов 9.3.1 и 10.2.1.

### Замечания и литература к § 10.2

Метод сопряженных градиентов является представителем более широкого класса методов так называемых методов сопряженных направлений. В алгоритме сопряженных направлений направления поиска являются  $B$ -сопряженными для некоторой подходящим образом выбранной матрицы  $B$ . Обсуждение этих методов дается в

Stewart G. W. (1973). "Conjugate Direction Methods for Solving Systems of Linear Equations", Numer. Math. 21, 284–97.

Классической ссылкой на метод сопряженных градиентов является

Hestenes M. R. and Stiefel E. (1952). "Methods of Conjugate Gradients for Solving Linear Systems", J. Res. Nat. Bur. Stand. 49, 409–36.

Анализ метода в точной арифметике можно найти в главе 2 книги

Hestenes M. R. (1980). Conjugate Direction Methods in Optimization, Springer-Verlag, Berlin.

См. также

Axelsson O. (1977). "Solution of Linear Systems of Equations: Iterative Methods", Sparse Matrix Techniques: Copenhagen, 1976, ed. V. A. Barker, Springer-Verlag, Berlin.

Faddeev D. K. and Faddeeva V. N. (1963). Computational Methods of Linear Algebra, W. H. Freeman and Co., San Francisco, California.

Фаддеев Д. К., Фаддеева В. Н. (1963). Вычислительные методы линейной алгебры.—М.–Л.: Физматгиз.

По поводу обсуждения характера сходимости метода сопряженных градиентов см.

Luenberger D. G. (1973). Introduction to Linear and Nonlinear Programming, Addison-Wesley, New York.

Van der Sluis A. and Van Der Vorst H. A. (1986). "The Rate of Convergence of Conjugate Gradients", Numer Math. 48, 543–560.

Идея использования метода сопряженных градиентов как итерационного впервые обсуждается в

Reid J. K. (1971b). "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations", in Large Sparse Sets of Linear Equations, ed. J. K. Reid, Academic Press, New York, pp. 231–54.

Несколько авторов пытались объяснить поведение алгоритма в условиях конечной арифметики. См.

Cullum J. and Willoughby R. A. (1977). "The Equivalence of the Lanczos and the Conjugate Gradient Algorithms", IBM Research Report RE-6903.

Greenbaum A. (1981). "Behavior of the Conjugate Gradient Algorithm in Finite Precision Arithmetic", Report UCRL 85752, Lawrence Livermore Laboratory, Livermore, California.

Wozniakowski H. (1980). "Roundoff Error Analysis of a New Class of Conjugate Gradient Algorithms", Lin. Alg. and Its Applic. 29, 507–29.

См. также анализ в

Cullum J. and Willoughby R. (1980). "The Lanczos Phenomena: An Interpretation Based on Conjugate Gradient Optimization", Lin. Alg. and Its. Applic. 29, 63–90.

Jennings A. (1977a). "Influence of the Eigenvalue Spectrum on the Convergence Rate of the Conjugate Gradient Method", J. Inst. Math. Applic. 20, 61–72.

Stewart G. W. (1975a). "The Convergence of the method of Conjugate Gradients at Isolated Extreme Points in the Spectrum", Numer. Math. 24, 85–93.

Наконец, упомянем о том, что метод можно использовать для вычисления собственного вектора большой разреженной симметричной матрицы. См.

Ruhe A. and Wiberg T. (1972). "The Method of Conjugate Gradients Used in Inverse Iteration", BIT 12, 543–54.

## Добавления при переводе к § 10.2

Методы сопряженных направлений и связанная с ними теория подробно освещаются в гл. 13 учебника

Воеводин В. В. (1980). Линейная алгебра.—М.: Наука.

См. также

- Воеводин В. В. (1979). О методах сопряженных направлений.– ЖВМ и МФ, 19, № 15, 1313–1317.  
 Марчук Г. И., Кузнецов Ю. А. (1980). Теория и применение обобщенного метода сопряженных градиентов.– Новосибирск: ВЦ СО АН СССР.  
 Кузнецов Ю. А. (1983). Метод сопряженных градиентов, его обобщения и применения. Вычислительные процессы и системы. Вып. 1.– М.: Наука.

## 10.3. Сопряженные градиенты с предобусловливанием

Предыдущий параграф мы закончили замечанием о том, что метод сопряженных градиентов работает хорошо для матриц, которые либо хорошо обусловлены, либо имеют лишь небольшое число различных собственных значений. (Последнее относится к случаю, когда  $A$  есть малоранговое возмущение единичной матрицы). В этом параграфе мы покажем, как проводится предобусловливание линейной системы с тем, чтобы матрица коэффициентов приобретала одну из этих прекрасных особенностей. Наше изложение будет весьма сжатым и неформальным. Более основательно эти вопросы представлены в Golub, Meurant (1983) и Axelsson (1985).

### 10.3.1. Вывод

Рассмотрим симметричную положительно определенную  $n \times n$ -систему  $Ax = b$ . Идея предобусловливания в сопряженных градиентах состоит в том, чтобы «стандартный» метод сопряженных градиентов применялся к преобразованной системе

$$\tilde{A}\tilde{x} = \tilde{b}, \quad (10.3.1)$$

где  $\tilde{A} = C^{-1}AC^{-1}$ ,  $\tilde{x} = Cx$ ,  $\tilde{b} = C^{-1}b$ , а  $C$  – симметричная положительно определенная матрица. Ввиду наших замечаний в § 10.2.8 нам следует попытаться выбрать  $C$  таким образом, чтобы  $\tilde{A}$  оказалась хорошо обусловленной матрицей или матрицей с кластеризованными собственными значениями. По причинам, которые скоро появятся, матрица должна быть к тому же «простой».

Применяя алгоритм 10.2.1 к (10.3.1), мы получим итерации

```

 $k = 0; \tilde{x}_0 = 0; \tilde{r}_0 = \tilde{b}$ 
while  $\tilde{r}_k \neq 0$ 
     $k = k + 1$ 
    if  $k = 1$ 
         $\tilde{p}_1 = \tilde{r}_0$ 
    else
         $\beta_k = \tilde{r}_{k-1}^T \tilde{r}_{k-1} / \tilde{r}_{k-2}^T \tilde{r}_{k-2}$ 
         $\tilde{p}_k = \tilde{r}_{k-1} + \beta_k \tilde{p}_{k-1}$ 
    end
     $a_k = \tilde{r}_{k-1}^T \tilde{r}_{k-1} / \tilde{p}_k^T C^{-1} AC^{-1} \tilde{p}_k$ 
     $\tilde{x}_k = \tilde{x}_{k-1} + a_k \tilde{p}_k$ 
     $\tilde{r}_k = \tilde{r}_{k-1} - a_k C^{-1} AC^{-1} \tilde{p}_k$ 
end
 $\tilde{x} = \tilde{x}_k$ 
```

(10.3.2)

Здесь  $\tilde{x}_k$  должно рассматриваться как приближение к  $\tilde{x}$ , а  $\tilde{r}_k$  – невязка в преобразованных координатах, т. е.  $\tilde{r}_k = \tilde{b} - \tilde{A}\tilde{x}_k$ . Конечно, получив  $\tilde{x}$ , мы можем найти  $x$  из уравнения  $x = C^{-1}\tilde{x}$ . Однако можно избежать явного обращения к матрице  $C^{-1}$ , записав  $\tilde{p}_k = Cp_k$ ,  $\tilde{x}_k = Cx_k$  и  $\tilde{r}_k = C^{-1}r_k$ . Действительно, подставив эти выражения

в (10.3.2) и вспомнив, что  $\tilde{b} = C^{-1}b$  и  $\tilde{x} = Cx$ , мы получим

```

 $k = 0; Cx_0 = 0; C^{-1}r_0 = C^{-1}b$ 
while  $C^{-1}r_k \neq 0$ 
     $k = k + 1$ 
    if  $k = 1$ 
         $Cp_1 = C^{-1}r_0$ 
    else
         $\beta_k = (C^{-1}r_{k-1})^T(C^{-1}r_{k-1})/(C^{-1}r_{k-2})^T(C^{-1}r_{k-2})$ 
         $Cp_k = C^{-1}r_{k-1} + \beta_k Cp_{k-1}$ 
    end
     $a_k = (C^{-1}r_{k-1})^T(C^{-1}r_{k-1})/(Cp_k)^T(C^{-1}AC^{-1})(Cp_k)$ 
     $Cx_k = Cx_{k-1} + a_k Cp_k$ 
     $C^{-1}r_k = C^{-1}r_{k-1} - a_k(C^{-1}AC^{-1})Cp_k$ 
end
 $Cx = Cx_k$ 
```

Если мы определим предобусловливатель  $M$  как  $M = C^2$  (это также положительно определенная матрица) и будем считать, что  $z_k$  – решение системы  $Mz_k = r_k$ , то (10.3.3) упрощается следующим образом:

**Алгоритм 10.3.1 (Сопряженные градиенты с предобусловливанием).** Для заданной симметричной положительно определенной матрицы  $A \in \mathbb{R}^{n \times n}$  и вектора  $b \in \mathbb{R}^n$  следующий алгоритм решает линейную систему  $Ax = b$ , используя метод сопряженных градиентов с предобусловливателем  $M \in \mathbb{R}^{n \times n}$ .

```

 $k = 0; x_0 = 0; r_0 = b$ 
while ( $r_k \neq 0$ )
    Решить  $Mz_k = r_k$ 
     $k = k + 1$ 
    if  $k = 1$ 
         $p_1 = z_0$ 
    else
         $\beta_k = r_{k-1}^T z_{k-1} / r_{k-2}^T z_{k-2}$ 
         $p_k = z_{k-1} + \beta_k p_{k-1}$ 
    end
     $a_k = r_{k-1}^T z_{k-1} / p_k^T Ap_k$ 
     $x_k = x_{k-1} + a_k p_k$ 
     $r_k = r_{k-1} - a_k Ap_k$ 
end
 $x = x_k$ 
```

Относительно этой процедуры следует сделать несколько важных замечаний:

- Можно доказать, что невязки и направления спуска удовлетворяют соотношениям

$$r_j^T M^{-1} r_i = 0, \quad i \neq j, \quad (10.3.4)$$

$$p_j^T (C^{-1} AC^{-1}) p_i = 0, \quad i \neq j. \quad (10.3.5)$$

- Знаменатели  $r_{k-2}^T z_{k-2} = z_{k-2}^T Mz_{k-2}$  не обращаются в нуль, так как  $M$  положительно определенная.
- Несмотря на то что преобразование  $C$  играло главную роль при выводе алгоритма, его действие проявляется лишь через предобусловливатель  $M = C^2$ .
- Для того чтобы алгоритм 10.3.1 был эффективным для разреженных матриц,

должны легко решаться линейные системы  $Mz = r$  и сходимость должна быть быстрой.

Выбор хорошего предобусловливателя может оказать впечатляющее воздействие на скорость сходимости. Ниже мы обсудим некоторые возможности такого выбора.

### 10.3.2. Предобусловливатели, связанные с неполным разложением Холецкого

Одна из наиболее важных стратегий предобусловливания связана с вычислением неполного разложения Холецкого матрицы  $A$ . Этот подход в идеальном плане предполагает вычисление нижней треугольной матрицы  $H$ , которая обладает некоторой хорошо обрабатываемой структурой разреженности и в каком-то смысле «блазка» к точному множителю  $G$  в разложении Холецкого для  $A$ . Предобусловливатель берется в виде  $M = HH^T$ . Чтобы оценить достоинства этого выбора, рассмотрим следующие факты:

- Существует единственная симметричная положительно определенная матрица  $C$ , такая, что  $M = C^2$ .
- Существует ортогональная матрица  $Q$ , такая, что  $C = QH^T$ , т. е.  $H^T$  – это верхний треугольный сомножитель в  $QR$ -разложении матрицы  $C$ .

Следовательно, получаем эвристическое соотношение

$$\begin{aligned} \tilde{A} &= C^{-1}AC^{-1} = C^{-T}AC^{-1} = \\ &= (HQ^T)^{-1}A(QH^T)^{-1} = Q(H^{-1}GG^TH^{-T})Q^T \approx I. \end{aligned} \quad (10.3.6)$$

Таким образом, чем лучше  $H$  аппроксимирует  $G$ , тем меньше обусловленность для  $\tilde{A}$ , и тем лучше работает алгоритм 10.3.1.

Легкий, но эффективный способ получения просто устроенной матрицы  $H$ , которая аппроксимирует  $G$ , это подправить редукцию Холецкого, полагая  $h_{ij}$  равным нулю, если соответствующий элемент  $a_{ij}$  нулевой. Проделав это с версией Холецкого, построенной на одноранговых модификациях, мы получаем

```

for k = 1 : n
    A(k, k) = sqrt(A(k, k))
    for i = k + 1 : n
        if A(i, k) ≠ 0
            A(i, k) = A(i, k) / A(k, k)
        end
    end
    for j = k + 1 : n
        for i = j : n
            if A(i, j) ≠ 0
                A(i, j) = A(i, j) - A(i, k) * A(j, k)
            end
        end
    end
end

```

(10.3.7)

На практике матрица  $A$  и множитель  $H$  неполного разложения Холецкого для нее хранятся в памяти с использованием каких-либо подходящих структур данных, и циклы приведенного выше алгоритма принимают при этом весьма специфический вид.

К несчастью, алгоритм (10.3.7) не всегда устойчив. Классы положительно определенных матриц, для которых неполный Холецкий устойчив, устанавливаются в Manteuffel (1979). См. также Elman (1986).

### 10.3.3. Неполные блочные предобусловливатели

Как почти все в этой книге, идеи неполного разложения, намеченные в общих чертах в предыдущем разделе, имеют блочный аналог. Для иллюстрации мы рассмотрим неполное блочное разложение Холецкого симметричной положительно определенной блочно-трехдиагональной матрицы

$$A = \begin{bmatrix} A_1 & E_1^T & 0 \\ E_1 & A_2 & E_2^T \\ 0 & E_2 & A_3 \end{bmatrix}.$$

В интересах иллюстрации мы предположим, что блоки  $A_i$  трехдиагональные, а  $E_i$  – диагональные. Матрицы такого строения возникают при стандартной 5-точечной дискретизации самосопряженных эллиптических дифференциальных уравнений на двумерной области.

Случай  $3 \times 3$ -матриц обладает достаточной общностью. Наше обсуждение будет основано на Concus, Golub, Meurant (1985). Пусть

$$G = \begin{bmatrix} G_1 & 0 & 0 \\ F_1 & G_2 & 0 \\ 0 & F_2 & G_3 \end{bmatrix}$$

есть точный множитель в блочном разложении Холецкого для  $A$ . Хотя  $G$  разреженная, если ее рассматривать как блочную матрицу, отдельные блоки являются плотными, за исключением  $G_1$ . Это можно увидеть из требуемых вычислений:

$$\begin{aligned} G_1 G_1^T &= B_1 \equiv A_1, \\ F_1 &= E_1 G_1^{-1}, \\ G_2 G_2^T &= B_2 \equiv A_2 - F_1 F_1^T = A_2 - E_1 B_1^{-1} E_1^T, \\ F_2 &= E_2 G_2^{-1}, \\ G_3 G_3^T &= B_3 \equiv A_3 - F_2 F_2^T = A_3 - E_2 B_2^{-1} E_2. \end{aligned}$$

Для блочного разложения Холецкого мы ищем, следовательно, приближенный множитель вида

$$\tilde{G} = \begin{bmatrix} \tilde{G}_1 & 0 & 0 \\ \tilde{F}_1 & \tilde{G}_2 & 0 \\ 0 & \tilde{F}_2 & \tilde{G}_3 \end{bmatrix},$$

причем такой, чтобы легко решались системы для предобусловителя  $M = \tilde{G} \tilde{G}^T$ . Для этого вводится требование разреженности блоков в  $\tilde{G}$ , и в данном случае это разумно, так как блоки  $A_i$  трехдиагональные, а блоки  $E_i$  диагональные:

$$\tilde{G}_1 \tilde{G}_1^T = \tilde{B}_1 \equiv A_1,$$

$$\tilde{F}_1 = E_1 \tilde{G}_1^{-1},$$

$$\tilde{G}_2 \tilde{G}_2^T = \tilde{B}_2 \equiv A_2 - E_1 \Lambda_1 E_1^T, \quad \Lambda_1 \text{ (трехдиагональная)} \approx \tilde{B}_1^{-1},$$

$$\tilde{F}_2 = E_2 \tilde{G}_2^{-1},$$

$$\tilde{G}_3 \tilde{G}_3^T = B_3 \equiv A_3 - E_2 \Lambda_2 E_2^T, \quad \Lambda_2 \text{ (трехдиагональная)} \approx \tilde{B}_2^{-1}.$$

Заметим, что все  $\tilde{B}_i$  трехдиагональные. Ясно, что  $\Lambda_i$  должны выбираться тщательно, чтобы обеспечить симметрию и положительную определенность для  $\tilde{B}_i$ . Далее, отсюда следует, что матрицы  $\tilde{G}_i$  нижние двухдиагональные. Матрицы  $\tilde{F}_i$  полные, но их не нужно формировать в явном виде. Например, в процессе решения системы  $Mz = r$  мы должны решать систему вида

$$\begin{bmatrix} \tilde{G}_1 & 0 & 0 \\ \tilde{F}_1 & \tilde{G}_2 & 0 \\ 0 & \tilde{F}_2 & \tilde{G}_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}.$$

Для выполнения матрично-векторных умножений, связанных с  $\tilde{F}_i = \tilde{E}_i \tilde{G}_i^{-1}$ , можно использовать прямое исключение:

$$\tilde{G}_1 w_1 = r_1,$$

$$\tilde{G}_2 w_2 = r_2 - \tilde{F}_1 w_1 = r_2 - E_1 \tilde{G}_1^{-1} w_1,$$

$$\tilde{G}_3 w_3 = r_3 - \tilde{F}_2 w_2 = r_3 - E_2 \tilde{G}_2^{-1} w_2.$$

Выбор  $\Lambda_i$  требует деликатности, так как в итоге  $\tilde{B}_i$  должна быть положительно определенной. В нашей организации вычислений центральный вопрос, каким образом аппроксимировать обратную к симметричной положительно определенной трехдиагональной  $m \times m$ -матрице  $T = (t_{ij})$  с помощью симметричной трехдиагональной матрицы  $\Lambda$ . Есть несколько разумных подходов:

- Положить  $\Lambda = \text{diag}(1/t_{11}, \dots, 1/t_{nn})$ .
- Взять в качестве  $\Lambda$  трехдиагональную часть в  $T^{-1}$ . Это может вычисляться эффективно, так как существуют  $u, v \in \mathbb{R}^m$ , такие, что нижняя треугольная часть в  $T^{-1}$  совпадает с нижней треугольной частью матрицы  $uv^T$ . См. Asplund (1959).
- Положить  $\Lambda = U^T U$ , где  $U$  – нижняя двухдиагональная часть матрицы  $G^{-1}$ , такой, что  $T = GG^T$  – разложение Холецкого. Это можно найти за  $O(m)$  флоков.

Эти аппроксимации и идущие от них свойства соответствующих предобусловителей обсуждаются в Concus, Golub, Meurant (1985).

#### 10.3.4. Идеи декомпозиции области

Численное решение эллиптических уравнений в частных производных часто приводит к линейным системам вида

$$\begin{bmatrix} A_1 & \dots & \dots & B_1 \\ \vdots & A_2 & & B_2 \\ & \ddots & \ddots & \vdots \\ \vdots & & A_p & B_p \\ B_1^T & B_2^T & \dots & B_p^T & Q \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ z \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_p \\ f \end{bmatrix}, \quad (10.3.8)$$

если неизвестные упорядочиваются подходящим образом. См. Meurant (1984). Здесь  $A_i$  – симметричные положительно определенные,  $B_i$  – разреженные, а последний блочный столбец обычно намного уже остальных.

Наш пример с  $p = 2$  служит для того, чтобы увязать (10.3.8) и его блочную структуру с геометрией соответствующей задачи и выбранной декомпозицией области. Предположим, что нам нужно решить эллиптическую задачу на области следующей формы:

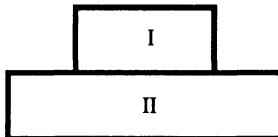


Рис. 10.3.1. Декомпозиция области.

Есть три «типа» переменных: внутренние для подобласти I (собранные в подвекторе  $x_1$ ), внутренние для подобласти II (собранные в подвекторе  $x_2$ ) и лежащие на границе между двумя областями (собранные в подвекторе  $z$ ). Заметим, что внутренние неизвестные одной подобласти не взаимодействуют с внутренними неизвестными другой подобласти, – это и объясняет нулевые блоки в (10.3.8). Заметим также, что число граничных неизвестных обычно мало по сравнению с общим числом неизвестных.

Теперь давайте исследуем связанные с (10.3.8) возможности предобусловливания. Для простоты будем по-прежнему считать, что  $p = 2$ . Если положить

$$M = L \begin{bmatrix} M_1^{-1} & 0 & 0 \\ 0 & M_2^{-1} & 0 \\ 0 & 0 & S^{-1} \end{bmatrix} L^T,$$

где

$$L = \begin{bmatrix} M_1 & 0 & 0 \\ 0 & M_2 & 0 \\ B_1^T & B_2^T & S \end{bmatrix},$$

то

$$M = \begin{bmatrix} M_1 & 0 & B_1 \\ 0 & M_2 & B_2 \\ B_1^T & B_2^T & S_* \end{bmatrix}, \quad (10.3.9)$$

при этом  $S_* = S + B_1^T M_1^{-1} B_1 + B_2^T M_2^{-1} B_2$ . Посмотрим, как мы могли бы выбрать блочные параметры  $M_1$ ,  $M_2$  и  $S$  с тем, чтобы получить эффективный предобусловливатель.

Если сравнить (10.3.9) и (10.3.8) в случае  $p = 2$ , то видно, что есть смысл в том, чтобы  $M_i$  аппроксимировало  $A_i$ , а  $S_*$  аппроксимировало  $Q$ . Последнее достигается, если  $S \approx Q - B_1^T M_1^{-1} B_1 - B_2^T M_2^{-1} B_2$ . Имеется несколько подходов к выбору  $S$ , и все они навеяны тем фактом, что мы не можем формировать плотные матрицы  $B_i M_i^{-1} B_i^T$ . Например, как обсуждалось в предыдущем разделе, для  $M_i^{-1}$  могли бы использоваться трехдиагональные аппроксимации. См. Meurant (1989).

Если подобласти устроены достаточно регулярно, а линейные системы с точ-

ными  $A_i$  легко решаются (скажем, с использованием быстрых процедур для уравнения Пуассона), то мы можем положить  $M_i = A_i$ . Отсюда следует, что  $M = A + E$ , причем  $\text{rank}(E) = m$ , где  $m$  – число граничных неизвестных. Таким образом, метод сопряженных градиентов с предобусловливанием теоретически будет сходиться за  $m$  шагов.

Независимо от того, какие аппроксимации будут включены в процесс, мы видим, что есть замечательные возможности для распараллеливания вследствие расщепления на независимые задачи для подобластей. Число подобластей на самом деле является обычно функцией геометрии задачи и в то же время числа доступных для вычислений процессоров.

### 10.3.5. Полиномиальные предобусловливатели

Вектор  $z$ , определяемый предобусловливающей системой  $Mz = r$ , до сих пор должен был рассматриваться как приближение к решению системы  $Az = r$ , поскольку  $M$  – приближение к  $A$ . Один из способов получения такого приближения к решению – применить  $p$  шагов стационарного метода  $M_1 z^{(k+1)} = N_1 z^{(k)} + r$ ,  $z^{(0)} = 0$ . Отсюда вытекает, что если  $G = M_1^{-1} N_1$ , то

$$z \equiv z^{(p)} = (I + G + \dots + G^{p-1}) M_1^{-1} r.$$

Таким образом, если  $M = (I + G + \dots + G_{p-1}) M_1^{-1}$ , то  $Mz = r$ , и мы можем рассматривать  $M$  как предобусловливателя. Важно, конечно, чтобы  $M$  была симметричной положительно определенной, и это ограничивает выбор  $M_1$ ,  $N_1$  и  $p$ . Матрица  $M$  является полиномом от  $G$ , и поэтому о ней говорят как о полиномиальном предобусловливателе. Этот тип предобусловливателя заманчив с точки зрения векторной или параллельной обработки, и, как следствие, он привлек к себе значительное внимание.

### 10.3.6. Заключительное предложение

Обсуждение полиномиального предобусловливателя указывает на важную связь между классическими итерациями и алгоритмом сопряженных градиентов с предобусловливанием. Многие итерационные методы имеют своим основным шагом

$$x_k = x_{k-2} + \omega_k (\gamma_k z_{k-1} + x_{k-1} - x_{k-2}), \quad (10.3.10)$$

где  $Mz_{k-1} = r_{k-1} = b - Ax_{k-1}$ . Например, если мы положим  $\omega_k = 1$  и  $\gamma_k = 1$ , то

$$x_k = M^{-1}(b - Ax_{k-1})x_{k-1},$$

т.е.  $Mx_k = Nx_{k-1} + b$ , где  $A = M - N$ . Таким образом, рассмотренные в § 10.1 методы Якоби, Гаусса – Зейделя, SOR и SSOR имеют вид (10.3.10). То же верно относительно полуитерационного чебышевского метода (10.1.12).

Следуя Concus, Golub, O’Leary (1976), можно и алгоритм 10.3.1 организовать таким образом, чтобы его центральный шаг имел вид (10.3.10):

```

 $x_1 = 0; x_0 = 0; k = 0; r_0 = b$ 
while  $r_k \neq 0$ 
     $k = k + 1$ 
    Решить  $Mz_{k-1} = r_{k-1}$  относительно  $z_{k-1}$ 
     $\gamma_{k-1} = z_{k-1}^T Mz_{k-1} / z_{k-1}^T Az_{k-1}$ 
    if  $k = 1$ 
        if  $k = 1$ 
             $\omega_1 = 1$ 

```

```

else
     $\omega_k = \left( 1 - \frac{\gamma_{k-1} z_{k-1}^T M z_{k-1}}{\gamma_{k-2} z_{k-2}^T M z_{k-2} \omega_{k-1}} \right)^{-1}$ 
end
 $x_k = x_{k-2} + \omega_k (\gamma_{k-1} z_{k-1} + x_{k-1} - x_{k-2})$ 
 $r_k = b - Ax_k$ 
end
 $x = x_n$ 

```

Итак, мы можем рассматривать скаляры  $\omega_k$  и  $\gamma_k$  в (10.3.11) как ускоряющие параметры, которые могут выбираться для того, чтобы увеличить скорость сходимости итераций  $Mx_k = Nx_{k-1} + b$ . Следовательно, любой итерационный метод, основанный на расщеплении  $A = M - N$ , может ускоряться с помощью алгоритма сопряженных градиентов, при условии что матрица  $M$  (предобусловливатель) симметрична и положительно определена.

### Задачи

**10.3.1.** Разобраться с процедурой неполного разложения, основанной на гахру-версии Холецкого, т. е. на алгоритме 4.2.1.

**10.3.2.** Сколько  $n$ -векторов требуется хранить в памяти при практической реализации алгоритма 10.3.1? Не учитывайте промежуточную память, которая может требоваться при решении системы  $Mz = r$ .

### Замечания и литература к § 10.3

Наше обсуждение метода сопряженных градиентов с предобусловливанием следует работам Axelsson O. (1985). "A Survey of Preconditioned Iterative Methods for Linear Systems of Equations", BIT 25, 166–187.

Concus P., Golub G. H. and Meurant G. (1985). "Block Preconditioning for the Conjugate Gradient Method", SIAM J. Sci. and Stat. Comp. 6, 220–252.

Concus P., Golub G. H. and O'Leary D. P. (1976). "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations", in Sparse Matrix Computations, ed. J. R. Bunch and D. J. Rose, Academic Press, New York.

Golub G. H. and Meurant G. (1983) Résolution Numérique des Grandes Systèmes Linéaires, Collection de la Direction des Etudes et Recherches de l'Electricité de France, vol 49, Eoyelles, Paris.

К числу других работ, связанных с расширением метода сопряженных градиентов, относятся

Bramble J. H., Pasciak J. E. and Schatz A. H. (1986). "The construction of Preconditioners for Elliptic Problems by Substructuring I", Math. Comp. 47, 103–134.

Bramble J. H., Pasciak J. E. and Schatz A. H. (1986). "The construction of Preconditioners for Elliptic Problems by Substructuring II", Math. Comp. 49, 1–17.

Chin R. C., Manteuffel T. A. and de Pillis J. (1984). ADI as a Preconditioning for Solving the Convection-Diffusion Equation", SIAM J. Sci. and Stat. Comp. 5, 281–299.

O'Leary D. P. (1980a). "The Block Conjugate Gradient Algorithm and Related Methods", Lin. Alg. and Its Applic. 29, 293–322.

Reid J. K. (1972). "The Use of Conjugate Gradients for Systems of Linear Equations Possessing Property A", SIAM J. Num. Anal. 9, 325–32.

Идеи неполной факторизации подробно освещаются в

Chan T. F., Jackson K. R. and Zhu B. (1983). "Alternating Direction Incomplete Factorizations", SIAM J. Numer. Anal. 20, 239–257.

Elman H. (1986). "A Stability Analysis of Incomplete LU Factorization", Math. Comp. 47, 191–218.

Manteuffel T. A. (1979). "Shifted Incomplete Cholesky Factorization", in Sparse Matrix Proceedings, 1978, ed. I. S. Duff and G. W. Stewart, SIAM Publications, Philadelphia, PA.

Meijerink J. A. and Van H. A. der Vorst (1977). "An Iterative Solution Method for Linear Equation Systems of Which the Coefficient Matrix is a Symmetric M-Matrix", Math. Comp. 31, 148–62.

Rodrigue G. and Wolitzer D. (1984). "Preconditioning by Incomplete Block Cycle Reduction", *Math. Comp.* 42, 549–566.

Превосходный обзор по декомпозиции области в терминах матричной алгебры дан в

Meurant G. (1989). "Domain Decomposition Methods for Partial Differential Equations on Parallel Computers", to appear *Int'l J. Supercomputing Applications*.

В основе алгоритма сопряженных градиентов – функционал  $\varphi(x)$ . Однако если вместо него рассматривается минимизация функционала  $\|Ax - b\|_2$ , то весь вывод подвергается небольшим изменениям. В результате получается так называемый метод миниминальных невязок, анализируемый в

Cline A. K. (1976b). "Several Observations on the Use of Conjugate Gradient Methods", ICASE Report 76, NASA Langley Research Center, Hampton, Virginia.

Его можно «вывести», заменив в алгоритме 10.2.1 все скалярные произведения  $u^T v$  на  $u^T Av$ . Метод минимальных невязок иногда приводит к несколько более быстрому уменьшению нормы невязки. Сходные модификации базового алгоритма сопряженных градиентов классифицируются в

Ashby S., Manteuffel T. A. and Saylor P. E. (1988). "A Taxonomy for Conjugate Gradient Methods", Report UCRL-98508, Lawrence Livermore National Laboratory, Livermore, CA.

В то же время в работе

Dennis J. E. Jr. and Turner K. (1987). "Generalized Conjugate Directions", *Lin. Alg. and Its. Applic.* 88/89, 187–209.

предлагается унифицированный подход к получению семейства процедур сопряженных направлений.

Некоторые характерные работы, рассматривающие развитие несимметричных процедур сопряженных градиентов, включают

Young D. M. and Jeas K. C. (1980). "Generalized Conjugate Gradient Acceleration of Nonsymmetrizable Iterative Methods", *Lin. Alg. and Its Applic.* 34, 159–94.

Axelsson O. (1980). "Conjugate Gradient Type Methods for Unsymmetric and Unconsistent Systems of Linear Equations", *Lin. Alg. and Its. Applic.* 29, 1–16.

Saad Y. (1981). "Krylov Subspace Methods for Solving Large Unsymmetric Linear Systems", *Math. Comp.* 37, 105–128.

Jeas K. C. and Young D. M. (1983). "On the Simplification of Generalized Conjugate Gradient Methods for Nonsymmetrizable Linear Systems", *Lin. Alg. and Its Applic.* 52/53, 399–417.

Faber V. and Manteuffel T. (1984). "Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method", *SIAM J. Numer. Anal.* 21, 352–362.

Van der Vorst H. A. (1986). "An Iterative Solution Method for Solving  $f(A)x = b$  Using Krylov Subspace Information Obtained for the Symmetric Positive Definite Matrix A", *J. Comp. and App. Math.* 18, 249–263.

Полиномиальные предобусловливатели обсуждаются в

Johnson O. G., Micchelli C. A. and Paul G. (1983). "Polynomial Preconditioners for Conjugate Gradient Calculations", *SIAM J. Numer. Anal.* 20, 362–376.

Eisenstat S. C. (1984). "Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods", *SIAM J. Sci. and Stat. Computing* 2, 1–4.

Adams L. (1985). "m-step Preconditioned Conjugate Gradient Methods", *SIAM J. Sci. and Stat. Comp.* 6, 452–463.

Ashby S. F. (1987). "Polynomial Preconditioning for Conjugate Gradient Methods", Ph. D. Thesis, Dept. of Computer Science, University of Illinois.

Различные векторные/параллельные реализации обсуждаются в

Ashcraft C. C. and Crimes R. (1988). "On Vectorizing Incomplete Factorization and SSOR Preconditioners", *SIAM J. Sci. and Stat. Comp.* 9, 122–151.

Axelsson O. and Polman B. (1986). "On Approximate Factorization Methods for Block Matrices Suitable for Vector and Parallel processors", *Lin. Alg. and Its Applic.* 77, 3–26.

- Dubois P. F., Greenbaum A. and Rodrigue G. H. (1979). "Approximating the Inverse of a Matrix for Use on Iterative Algorithms on Vector Processors", Computing 22, 257–268.
- Gropp W. D. and Keyes D. E. (1988). "Complexity of Parallel Implementation of Domain Decomposition Techniques for Elliptic Partial Differential Equations", SIAM J. Sci. and Stat. Comp. 9, 312–326.
- Jordan T. (1984). "Conjugate Gradient Preconditioners for Vector and Parallel Processors", in G. Birkhoff and A. Schoenstadt (eds.), Proceedings of the Conference on Elliptic Problem Solvers, Academic Press, NY.
- Melhem R. (1987). "Toward Efficient Implementation of Preconditioned Conjugate Gradient Methods on Vector Supercomputers", Int'l J. Supercomputing Applications I, 70–98.
- Meurant G. (1984). "The Block Preconditioned Conjugate Gradient Method on vector Computers", BIT 24, 623–633.
- Poole E. L. and Ortega J. M. (1987). "Multicolor ICCG Methods for Vector Computers", SIAM J. Numer. Anal. 24, 1394–1418.
- Seager M. K. (1986). "Parallelizing Conjugate Gradient for the Cray X-MP", Parallel Computing 3, 35–47.
- Van der Vorst H. A. (1982). "A Vectorizable Variant of Some ICCG Methods", SIAM J. Sci. and Stat. Comp. 3, 350–356.
- Van der Vorst H. A. (1986). "The Performance of Fortran Implementations for Preconditioned Conjugate Gradients on Vector Computers", Parallel Computing 3, 49–58.

Интересное приложение сопряженных градиентов с предобусловливанием к теплицевым системам обсуждается в

- Chan T. F. (1988). "An Optimal Circulant Preconditioners for Toeplitz Systems", SIAM J. Sci. Stat. Comp. 9, 766–771.

#### Добавления при переводе к § 10.3

Интересные свойства и различные способы предобусловливания с использованием матриц типа теплицевых изучаются в

- Тыртышников Е. Е. (1989). Теплицевые матрицы, некоторые их аналоги и приложения.– М.: ОВМ АН СССР.

Новый подход к построению многоуровневых предобусловливателей, связанных с многосеточными методами декомпозиции области, предложен в

- Kuznetsov Yu. A. (1989). Algebraic multigrid domain decomposition.– Sov. J. Numer. Anal. Modelling, 4, N 5, 351–379.

В основе широкого класса обобщений методов сопряженных направлений лежит понятие псевдоортогональности: упорядоченная система векторов  $s_1, \dots, s_m$  называется  $R$ -псевдоортогональной, если  $(Rs_i, s_i) = 0$  при  $i < j \leq m$  и  $(Rs_j, s_j) \neq 0$ . Общая схема метода сопряженных направлений для решения системы  $Ax = b$  должна описывать вычисление последовательных приближений  $x_i$ , соответствующих им невязок  $r_i$  и вспомогательных векторов  $s_i$ , образующих САВ-псевдоортогональную систему для подходящих фиксированных матриц  $C$  и  $B$ . Нулевой шаг состоит из вектора начального приближения  $x_0$ , вычисления невязки  $r_0 = b - Ax_0$  и присваивания  $s_1 = r_0$ . Далее, на  $i$ -м шаге выполняются такие действия:

$$a_i = (Cr_{i-1}, r_{i-1}) / (CABs_i, s_i),$$

$$r_i = r_{i-1} - a_i ABs_i,$$

$$x_i = x_{i-1} + a_i Bs_i,$$

$$s_{i+1} = r_i + \sum_{j=1}^i \beta_{i+1,j} s_j,$$

где  $\beta_{i+1,j}$  находятся из треугольной системы

$$\begin{bmatrix} (Rs_1, s_1) & 0 \\ \vdots & \ddots \\ (Rs_1, s_i) & \dots & (Rs_i, s_i) \end{bmatrix} \begin{bmatrix} \beta_{i+1,1} \\ \vdots \\ \beta_{i+1,i} \end{bmatrix} = - \begin{bmatrix} (Rr_i, s_1) \\ \vdots \\ (Rr_i, s_i) \end{bmatrix}.$$

Если  $r_i = 0$ , то процесс заканчивается на  $i$ -м шаге. Если  $r_{i-1} \neq 0$ , то  $a_i = 0$  или  $(Rs_i, s_i) = 0$ , тогда процесс обрывается на  $i$ -м шаге. Указанная общая схема рассмотрена в

Воеводин В. В. (1979). О методах сопряженных направлений.– ЖВМ и МФ, 19, № 5, 1313–1317.

В этой же работе показано, что для двучленности соотношений, т. е. для того чтобы  $\beta_{i+1,j} = 0$  при  $j \leq i - 1$  достаточным является условие  $(C A B C^{-1})^* = \alpha I + \beta A B$ , где  $\alpha, \beta$  – какие-то числа. Для  $k$ -членности соотношений, т. е. для того чтобы  $\beta_{i+1,j} = 0$  при  $j \leq i - k$ , достаточным является обобщенное условие Воеводина  $(C A B C^{-1})^* = \sum_{j=0}^k \alpha_j (A B)^j$ , где  $\alpha_j$  – какие-то числа.

Е. Е. Тыртышников доказал следующую теорему о необходимости этого условия: пусть для всех начальных приближений, таких, что процесс не обрывается, выполняется условие Воеводина и хотя бы для одного начального приближения процесс заканчивается в точности на  $n$ -м шаге. Предположим, что  $n \geq 2k + 3$  и матрицы  $A, B, C$  невырожденные. Тогда  $\beta_{i+1,j} = 0$  при  $j \leq i - 1$ . См.

Воеводин В. В., Тартышников Е. Е. (1981). Об обобщении методов сопряженных направлений.– Численные методы алгебры.– М.: Изд-во Москов. ун-та.

# Глава 11

## Функции от матриц

Задача вычисления функции  $f(A)$  от матрицы  $A$  размера  $n \times n$  часто встречается в теории управления и других прикладных областях. Грубо говоря, если скалярная функция  $f(z)$  определена на спектре  $\lambda(A)$ , то функцию от матрицы  $f(A)$  определяют, подставляя матрицу  $A$  вместо переменной  $z$  в «формулу» для  $f(z)$ . Например, если  $f(z) = (1 + z)/(1 - z)$  и  $1 \notin \lambda(A)$ , то  $f(A) = (I + A)(I - A)^{-1}$ .

Эти вычисления становятся особенно интересными, когда функция  $f$  трансцендентная. Один из подходов в этой более сложной ситуации состоит в вычислении какого-нибудь спектрального разложения  $A = YBY^{-1}$  и использовании формулы  $f(A) = Yf(B)Y^{-1}$ . Если матрица  $B$  достаточно простая, то часто можно вычислить  $f(B)$  непосредственно. Это показано в § 11.1 для разложений Жордана и Шура. Неудивительно, что на основе последнего разложения получается более устойчивая  $f(A)$  процедура.

Другой подход к вычислению функции от матрицы состоит в аппроксимации требуемой функции  $f(A)$ , легко вычисляемой функцией  $g(A)$ . Например, функция  $g$  может быть усеченным рядом Тейлора, аппроксимирующим функцию  $f$ . Оценки погрешности, связанный с такой аппроксимацией функции от матрицы, даны в § 11.2.

В последнем разделе мы обсуждаем специальную и очень важную задачу вычисления матричной экспоненты  $e^A$ .

### 11.1. Спектральные методы

Для заданной матрицы  $A$  размера  $n \times n$  и скалярной функции  $f(z)$  существует несколько способов определить функцию от матрицы  $f(A)$ . Неформальный способ определения – подставить матрицу  $A$  вместо переменной  $z$  в формулу для  $f(z)$ . Например, если  $p(z) = 1 + z$  и  $r(z) = (1 - (z/2))^{-1}(1 + (z/2))$  при  $z \neq 2$ , то, конечно, разумно определить  $p(A)$  и  $r(A)$  как

$$p(A) = I + A$$

и

$$r(A) = \left( I - \frac{A}{2} \right)^{-1} \left( I + \frac{A}{2} \right), \quad 2 \notin \lambda(A).$$

Подстановка  $A$  вместо  $z$  также срабатывает и в случае трансцендентных функций, например,

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$

Однако для того чтобы сделать последующие алгоритмические разработки корректными, мы должны более аккуратно определить  $f(A)$ .

### 11.1.1. Определение

Существует много способов строгого определения понятия функции от матрицы. Смотри [Rinehart, 1955]. Возможно, наиболее изящный подход основан на применении контурного интеграла. Пусть функция  $f(z)$  аналитична на замкнутом множестве, ограниченном замкнутым контуром  $\Gamma$ , который окружает спектр  $\lambda(A)$ . Определим  $f(A)$  как матрицу

$$f(A) = \frac{1}{2\pi i} \oint_{\Gamma} f(z)(zI - A)^{-1} dz. \quad (11.1.1)$$

В этом определении можно сразу узнать матричный вариант интегральной теоремы Коши. Интеграл (11.1.1) определяется поэлементно:

$$f(A) = (f_{kj}) \Rightarrow f_{kj} = \frac{1}{2\pi i} \oint_{\Gamma} f(z) e_k^T (zI - A)^{-1} e_j dz.$$

Заметим, что элементы матрицы  $(zI - A)^{-1}$  аналитичны на  $\Gamma$  и матрица  $f(A)$  определена в тех случаях, когда функция  $f(z)$  аналитична в окрестности  $\lambda(A)$ .

### 11.1.2. Жорданова характеристизация

Определение (11.1.1), совершенно бесполезное с вычислительной точки зрения, можно использовать для вывода более практических характеризаций  $f(A)$ . Например, если матрица  $f(A)$  определена, и

$$A = XBX^{-1} = X \operatorname{diag}(B_1, \dots, B_p) X^{-1}, \quad B_i \in \mathbb{C}^{n_i \times n_i},$$

то несложно проверить, что

$$f(A) = Xf(B)X^{-1} = X \operatorname{diag}(f(B_1), \dots, f(B_p)) X^{-1}. \quad (11.1.2)$$

Для случая когда матрицы  $B_i$  – жордановы блоки, мы получаем следующий результат.

**Теорема 11.1.1.** Пусть  $X^{-1}AX = \operatorname{diag}(J_1, \dots, J_p)$  – жорданова каноническая форма (JCF) матрицы  $A \in \mathbb{C}^{n \times n}$  с матрицами

$$F_i = \begin{bmatrix} \lambda_i & 1 & & \dots & 0 \\ 0 & \lambda_i & 1 & & \vdots \\ \ddots & \ddots & \ddots & & \\ \vdots & \ddots & \ddots & 1 & \\ 0 & \dots & 0 & \lambda_i & \end{bmatrix},$$

являющимися жордановыми блоками размера  $m_i \times m_i$ . Если функция  $f(z)$  аналитична в некотором открытом множестве, включающем  $\lambda(A)$ , то

$$f(A) = X \operatorname{diag}(f(J_1), \dots, f(J_p)) X^{-1},$$

где

$$f(J_i) = \begin{bmatrix} f(\lambda_i) & f^{(1)}(\lambda_i) & & \dots & \frac{f^{(m_i-1)}(\lambda_i)}{(m_i-1)!} \\ 0 & f(\lambda_i) & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \\ 0 & \dots & & f^{(1)}(\lambda_i) & f(\lambda_i) \end{bmatrix}.$$

*Доказательство.* Принимая во внимание замечания, предшествующие формулировке теоремы, достаточно рассмотреть матрицу  $f(G)$ , где

$$G = \lambda I + E, \quad E = (\delta_{i,j-1})$$

—жорданов блок размера  $q \times q$ . Пусть матрица  $(zI - G)$  невырождена. Так как

$$(zI - G)^{-1} = \sum_{k=0}^{q-1} \frac{E^k}{(z - \lambda)^{k+1}},$$

то из интегральной теоремы Коши следует, что

$$f(G) = \sum_{k=0}^{q-1} \left[ \frac{1}{2\pi i} \oint_{\Gamma} \frac{f(z)}{(z - \lambda)^{k+1}} dz \right] E^k = \sum_{k=0}^{q-1} \frac{f^{(k)}(\lambda)}{k!} E_k.$$

Заметив, что  $E^k = (\delta_{i,j-k})$ , получаем требуемый результат.  $\square$

**Следствие 11.1.2.** Если  $A \in \mathbf{C}^{n \times n}$ ,  $A = X \operatorname{diag}(\lambda_1, \dots, \lambda_n) X^{-1}$ , и матрица  $f(A)$  определена, то  $f(A) = X \operatorname{diag}(f(\lambda_1), \dots, f(\lambda_n)) X^{-1}$ .

*Доказательство.* Здесь все жордановы блоки имеют размер  $1 \times 1$ .  $\square$

Этот результат показывает тесную связь между функцией от матрицы  $f(A)$  и собственной системой матрицы  $A$ . К несчастью, вычислительные достоинства JCF-подхода к проблеме функций от матрицы весьма сомнительны, если только матрица  $A$  не является диагонализуемой с хорошо обусловленной матрицей собственных векторов. В самом деле, можно ожидать, что погрешности округления порядка  $u k_2(X)$  испортят результат вычисления, так как придется решать линейную систему с матрицей  $X$ . Следующий пример наводит на мысль, что плохо обусловленных преобразований подобия следует избегать при вычислении функции от матрицы.

**Пример 11.1.1.** Если

$$A = \begin{bmatrix} 1 + 10^{-5} & 1 \\ 0 & 1 - 10^{-5} \end{bmatrix},$$

то любая матрица собственных векторов является масштабированным по столбцам вариантом матрицы

$$X = \begin{bmatrix} 1 & -1 \\ 0 & 2(1 - 10^{-5}) \end{bmatrix}$$

и имеет число обусловленности во второй норме порядка  $10^5$ . Используя компьютер с машинной точностью  $u \approx 10^{-7}$ , мы найдем, что

$$\text{fl}[X^{-1} \operatorname{diag}(\exp(1 + 10^{-5}), \exp(1 - 10^{-5})) X] = \begin{bmatrix} 2.718307 & 2.750000 \\ 0.000000 & 2.718254 \end{bmatrix},$$

хотя

$$e^A = \begin{bmatrix} 2.718309 & 2.718282 \\ 0.000000 & 2.718255 \end{bmatrix}.$$

### 11.1.3. Метод разложения Шура

Некоторые трудности, связанные с жордановым подходом к проблеме функции от матрицы, можно обойти, положившись на разложение Шура. Если  $A = Q T Q^H -$

разложение Шура матрицы  $A$ , то

$$f(A) = Qf(T)Q^H.$$

Для того чтобы этот подход был эффективным, нам нужен алгоритм вычисления функций от верхних треугольных матриц. К несчастью, явные выражения для  $f(T)$ , как показывает следующая теорема, очень сложные.

**Теорема 11.1.3.** Пусть  $T = (t_{ij})$  – верхняя треугольная матрица  $n \times n$  с собственными значениями  $\lambda_i = t_{ii}$ , и пусть матрица  $f(T)$  определена. Если  $f(T) = (f_{ij})$ , то  $f_{ij} = 0$  при  $i > j$ ,  $f_{ij} = f(\lambda_i)$  при  $i = j$  и для всех  $i < j$  имеем равенство

$$f_{ij} = \sum_{(s_0, \dots, s_k) \in S_{ij}} t_{s_0, s_1} t_{s_1, s_2} \dots t_{s_{k-1}, s_k} f[\lambda_{s_0}, \dots, \lambda_{s_k}],$$

где  $S_{ij}$  означает множество всех строго возрастающих целочисленных последовательностей, начинающихся с  $i$  и заканчивающихся  $j$ , и  $f[\lambda_{s_0}, \dots, \lambda_{s_k}]$  – разделенная разность  $k$ -го порядка функции  $f$  на сетке  $\{\lambda_{s_0}, \dots, \lambda_{s_k}\}$ .

*Доказательство.* Смотри [Descloux, 1963], [Davis, 1973], [Van Loan, 1975b].  $\square$

Вычисление матрицы  $f(T)$  в соответствии с теоремой 11.1.3 потребовало бы  $O(2^n)$  флопов. К счастью, Парлетт [Parlett, 1974a] получил изящный рекуррентный метод для определения строго верхней треугольной части матрицы  $F = f(T)$ . Он требует только  $2n^3/3$  флопов и может быть выведен из следующего коммутационного свойства:

$$FT = TF. \quad (11.1.3)$$

В самом деле, сравнивая элементы  $(i,j)$  левой и правой частей этого уравнения, находим, что

$$\sum_{k=i}^j f_{ik} t_{kj} = \sum_{k=i}^j t_{ik} f_{kj}, \quad j > i,$$

и, следовательно, если элементы  $t_{ii}$  и  $t_{jj}$  различны, то

$$t_{ij} = t_{ij} \frac{f_{jj} - f_{ii}}{t_{jj} - t_{ii}} + \sum_{k=i+1}^{j-1} \frac{t_{ik} f_{kj} - f_{ik} t_{kj}}{t_{jj} - t_{ii}}. \quad (11.1.4)$$

Из этого равенства мы заключаем, что элемент  $f_{ij}$  является линейной комбинацией своих соседей, стоящих левее и ниже в матрице  $F$ . Например, элемент  $f_{25}$  зависит от  $f_{22}, f_{23}, f_{24}, f_{55}, f_{45}$  и  $f_{35}$ . Поэтому вся верхняя треугольная часть матрицы  $F$  может быть вычислена диагональ за диагональю, начиная с  $f(t_{11}), \dots, f(t_{nn})$ . Полная процедура имеет следующий вид:

**Алгоритм 11.1.1.** Этот алгоритм вычисляет функцию от матрицы  $F = f(T)$ , где  $T$  – верхняя треугольная матрица с различными собственными значениями и функция  $f$  определена на спектре  $\lambda(T)$ .

```

for i = 1:n
  f_ii = f(t_ii)
  end
  for p = 1:n-1
    for i = 1:n-p
      j = i + p
      s = t_ij * (f_jj - f_ii)
      f_ij = s / (t_jj - t_ii)
    end
  end

```

```

for  $k = i + 1 : j - 1$ 
     $s = s + t_{ik} f_{kj} - f_{ik} t_{kj}$ 
end
 $f_{ij} = s / (t_{jj} - t_{ii})$ 
end
end

```

Этот алгоритм требует  $2n^3/3$  флопов. Предполагая, что  $T = QAQ^H$  – форма Шура матрицы  $A$ , имеем  $f(A) = QFQ^H$ , где  $F = f(T)$ . Понятно, что большую часть работы при вычислении  $f(A)$  этим способом составит вычисление разложения Шура.

**Пример 11.1.2.** Если

$$T = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 3 & 4 \\ 0 & 0 & 5 \end{bmatrix}$$

и  $f(z) = (1+z)/z$ , то элементы матрицы  $F = (f_{ij}) = (T)$  можно определить по формулам

$$f_{11} = (1+1)/1 = 2,$$

$$f_{22} = (1+3)/3 = 4/3,$$

$$f_{33} = (1+5)/5 = 6/5,$$

$$f_{12} = t_{12} (f_{22} - f_{11}) / (t_{22} - t_{11}) = -2/3,$$

$$f_{23} = t_{23} (f_{33} - f_{22}) / (t_{33} - t_{22}) = -4/15,$$

$$f_{13} = [t_{13} (f_{33} - f_{11}) + (t_{12} f_{23} - f_{12} t_{23})] / (t_{33} - t_{11}) = -1/15.$$

#### 11.1.4. Метод блочного разложения Шура

Если матрица  $A$  имеет близкие или кратные собственные значения, то алгоритм 11.1.1 приводит к плохим результатам. В этом случае разумно использовать блочную версию алгоритма 11.1.1. Мы обрисуем в общих чертах такую процедуру, следя за работе [Parlett, 1974a]. Первый шаг состоит в выборе матрицы  $Q$  в разложении Шура, такой, чтобы близкие или кратные собственные значения были собраны в блоках  $T_{11}, \dots, T_{pp}$ , расположенных на диагонали матрицы  $T$ . В частности, мы должны найти разбиение на блоки

$$T = \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1p} \\ 0 & T_{22} & \cdots & T_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T_{pp} \end{bmatrix}, \quad F = \begin{bmatrix} F_{11} & F_{12} & \cdots & F_{1p} \\ 0 & F_{22} & \cdots & F_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_{pp} \end{bmatrix},$$

при котором  $\lambda(T_{ii}) \cap \lambda(T_{jj}) = \emptyset$ ,  $i \neq j$ . Блочные размеры можно определить, используя методы § 7.6.

Затем мы вычисляем подматрицы  $F_{ii} = f(T_{ii})$  при  $i = 1 : p$ . Поскольку собственные значения блока  $T_{ii}$  предположительно близкие, эти вычисления требуют специальных методов. (Некоторые возможности рассматриваются в двух следующих разделах.) Если диагональные блоки матрицы  $F$  известны, то блоки строго верхней треугольной части матрицы  $F$  могут быть найдены рекуррентно, как в скалярном случае. Для того чтобы вывести соответствующие формулы, мы приравняем  $(i,j)$

блоки левой и правой частей уравнения  $FT = TF$  при  $i < j$  и получим следующее обобщение равенства (11.1.4):

$$F_{ij}T_{jj} - T_{ii}F_{ij} = T_{ij}F_{jj} - F_{ii}T_{ij} + \sum_{k=i+1}^{j-1} (T_{ik}F_{kj} - F_{ik}T_{kj}). \quad (11.1.5)$$

Это линейная система, неизвестными которой являются элементы блоков  $F_{ij}$  и чья правая часть «известна», если мы вычисляем  $F_{ij}$  по блочным наддиагоналям. Уравнение (11.1.5) можно решать по алгоритму Бартелса–Стюарта (алгоритм 7.6.2).

Описанный здесь блочный метод Шура удобен, когда вычисляются вещественные функции от вещественных матриц. После вычисления вещественной формы Шура  $A = QTQ^T$  блочный алгоритм можно использовать для того, чтобы справиться с выступами размера  $2 \times 2$  на диагонали матрицы  $T$ .

### Задачи

**11.1.1.** Используя определение (11.1.1), показать, что (a)  $Af(A) = f(A)A$ , (b) матрица  $f(A)$  верхняя треугольная, если матрица  $A$  – верхняя треугольная, и (c) матрица  $f(A)$  эрмитова, если матрица  $A$  эрмитова.

**11.1.2.** Переписать алгоритм 11.1.1 так, чтобы матрица  $f(T)$  вычислялась столбец за столбцом.

**11.1.3.** Пусть  $A = X \operatorname{diag}(\lambda_i) X^{-1}$ , где  $X = [x_1, \dots, x_n]$  и  $X^{-1} = [y_1, \dots, y_n]^H$ . Показать, что если матрица  $f(A)$  определена, то

$$f(A) = \sum_{k=1}^n f(\lambda_k) x_k y_k^H.$$

**11.1.4.** Показать, что

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{matrix} p \\ q \end{matrix} \Rightarrow f(T) = \begin{bmatrix} F_{11} & F_{12} \\ 0 & F_{22} \end{bmatrix} \begin{matrix} p \\ q \end{matrix},$$

где  $F_{11} = f(T_{11})$  и  $F_{22} = f(T_{22})$ . Считать, что матрица  $f(T)$  определена.

### Замечания и литература к § 11.1

В следующих книгах проблема функций от матрицы изложена лучше, чем обычно:

Bellman R. (1969). Introduction to Matrix Analysis, McGraw-Hill, New York.

Гантмахер Ф. Р. Теория матриц. Т. 1, 2.–М.: Наука, 1967.

Mirsky L. (1955). An Introduction to Linear Algebra, Oxford University Press, London.

Представление матрицы  $f(A)$  контурным интегралом, приведенное в этом разделе, полезно в функциональном анализе благодаря универсальности этого представления. См.

Dunford N. and Schwartz J. (1958). Linear Operators, Part I, Interscience, New York. [Имеется перевод: Данфорд Н., Шварц Дж. Линейные операторы. Т. 1. Общая теория.–М.: ИЛ, 1962.]

Как мы уже говорили, возможны другие определения  $f(A)$ . Однако для матричных функций, обычно встречающихся на практике, все эти определения эквивалентны. См.

Rinehart R. F. (1955). “The Equivalence of Definitions of a Matric Function”, Amer. Math. Monthly 62, 395–414.

Различные аспекты жорданова представления детально изложены в

Frame J. S. (1964a). “Matrix Functions and Applications, Part II”, IEEE Spectrum I (April), 102–8.

Frame J. S. (1964b). “Matrix Functions and Applications, Part IV”, IEEE Spectrum I (June), 123–31.

Следующие работы касаются разложения Шура и его связи с  $f(A)$  проблемой:

Davis D. (1973). “Explicit Functional Calculus”, Lin. Alg. and Its Applic. 6, 193–99.

- Descloux J. (1963). "Bounds for the Spectral Norm of Functions of Matrices", Numer. Math. 5, 185–90.
- Van Loan C. F. (1975b). "A Study of the Matrix Exponential", Numerical Analysis, Report No. 10, Dept. of Maths., University of Manchester, England.
- Алгоритм 11.1.1 и различные вычислительные трудности, возникающие при его применении к матрицам, имеющим близкие или кратные собственные значения, обсуждаются в
- Parlett B. N. (1974a). "Computation of Functions of Triangular Matrices", Memorandum No. ERL-M481, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.
- Parlett B. N. (1976). "A Recurrence Among the Elements of Functions of Triangular Matrices", Lin. Alg. and Its Appl. 14, 117–21.
- Компромисс между методами Жордана и Шура для  $f(A)$  проблемы достигается, если матрицу  $A$  привести к блочно-диагональной форме, как описано в § 7.6.3. Смотри
- Kågström B. (1977). "Numerical Computation of Matrix Functions", Department of Information Processing Report UMINF-58.77, University of Umeå, Sweden.

## 11.2. Аппроксимационные методы

Теперь мы рассмотрим класс методов вычисления функций от матрицы, в которых собственные значения не играют главную роль. Эти методы основаны на следующей идее: если функция  $g(z)$  аппроксимирует  $f(z)$  на спектре  $\lambda(A)$ , то матрица  $g(A)$  аппроксимирует матрицу  $f(A)$ , например

$$e^A \approx I + A + \frac{A^2}{2!} + \dots + \frac{A^q}{q!}.$$

Мы начнем с оценки нормы  $\|f(A) - g(A)\|$  при помощи представлений Жордана и Шура функции от матрицы. И продолжим некоторыми замечаниями о величине матричных многочленов.

### 11.2.1. Жорданов анализ

Жорданово представление функции от матрицы (теорема 11.1.1) можно использовать для оценки погрешности аппроксимации матрицы  $f(A)$  матрицей  $g(A)$ .

**Теорема 11.2.1.** Пусть  $X^{-1}AX = \text{diag}(J_1, \dots, J_p)$  есть JCF-матрицы  $A \in \mathbb{C}^{n \times n}$  с матрицами

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \dots & 0 \\ 0 & \lambda_i & 1 & & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & 1 \\ \vdots & & & & \ddots \\ 0 & \dots & & & \lambda_i \end{bmatrix},$$

являющимиися жордановыми блоками размера  $m_i \times m_i$ . Если функции  $f(z)$  и  $g(z)$  аналитичны в открытом множестве, содержащем  $\lambda(A)$ , то

$$\|f(A) - g(A)\|_2 \leq k_2(X) \max_{\substack{1 \leq i \leq p \\ 0 \leq r \leq m_i - 1}} m_i \frac{|f^{(r)}(\lambda_i) - g^{(r)}(\lambda_i)|}{r!}.$$

*Доказательство.* Полагая  $h(z) = f(z) - g(z)$ , имеем

$$\|f(A) - g(A)\|_2 = \|X \operatorname{diag}(h(J_1), \dots, h(J_p)) X^{-1}\|_2 \leq k_2(X) \max_{1 \leq i \leq p} \|h(J_i)\|_2.$$

Используя теорему 11.1.1 и уравнение (2.3.8), мы заключаем, что

$$\|h(J_i)\|_2 \leq m_i \max_{0 \leq r \leq m_i - 1} \frac{|h^{(r)}(\lambda_i)|}{r!},$$

доказывая тем самым теорему.  $\square$

### 11.2.2. Анализ на основе разложения Шура

Если мы будем исходить из разложения Шура, а не Жордана, то получим альтернативную оценку.

**Теорема 11.2.2.** Пусть  $Q^H A Q = T = \operatorname{diag}(\lambda_i) + N$  – разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$  с матрицей  $N$ , являющейся строго верхней треугольной частью  $T$ . Если функции  $f(z)$  и  $g(z)$  аналитичны на замкнутом выпуклом множестве  $\Omega$ , внутренняя часть которого содержит  $\lambda(A)$ , то

$$\|f(A) - g(A)\|_F \leq \sum_{r=0}^{n-1} \delta_r \frac{\|N\|^r\|_F}{r!},$$

где

$$\delta_r = \sup_{z \in \Omega} |f^{(r)}(z) - g^{(r)}(z)|.$$

*Доказательство.* Пусть  $h(z) = f(z) - g(z)$  и  $H = (h_{ij}) = h(A)$ . Пусть  $S_{ij}^{(r)}$  означает множество целочисленных строго возрастающих последовательностей  $(s_0, \dots, s_r)$ , таких, что  $s_0 = i$ ,  $s_r = j$ . Заметим, что

$$S_{ij} = \bigcup_{r=1}^{j-i} S_{ij}^{(r)}$$

и, следовательно, из теоремы 11.1.3 для всех  $i < j$  мы получаем следующее равенство:

$$h_{ij} = \sum_{r=1}^{j-1} \sum_{s \in S_{ij}^{(r)}} n_{s_0, s_1} n_{s_1, s_2} \dots n_{s_{r-1}, s_r} h[\lambda_{s_0}, \dots, \lambda_{s_r}].$$

Теперь, так как множество  $\Omega$  выпуклое, а функция  $h$  аналитическая, мы имеем неравенство

$$|h[\lambda_{s_0}, \dots, \lambda_{s_r}]| \leq \sup_{z \in \Omega} \frac{|h^{(r)}(z)|}{r!} = \frac{\delta_r}{r!}. \quad (11.2.1)$$

Более того, если  $|N|^r = (n_{ij}^{(r)})$  при  $r \geq 1$ , то можно показать, что

$$n_{ij}^{(r)} = \begin{cases} 0, & j < i + r, \\ \sum_{s \in S_{ij}^{(r)}} |n_{s_0, s_1} n_{s_1, s_2} \dots n_{s_{r-1}, s_r}|, & j \geq i + r. \end{cases} \quad (11.2.2)$$

Беря абсолютные величины в выражении для  $h_{ij}$  и используя затем (11.2.1) и (11.2.2), завершаем доказательство.  $\square$

Оценки из вышеупомянутых теорем наводят на мысль, что для аппроксимации  $f(A)$  требуется нечто большее, чем просто аппроксимация  $f(z)$  на спектре матрицы  $A$ . В частности, мы видим, что если собственная система матрицы  $A$  плохо обусловлена и/или отклонение матрицы  $A$  от нормальности велико, то различие между  $f(A)$  и  $g(A)$  может быть значительно большим, чем максимум  $|f(z) - g(z)|$  на  $\lambda(A)$ . Следовательно, хотя аппроксимационные методы избегают вычислений, связанных с собственными значениями, они оказываются подверженными влиянию структуры собственной системы матрицы  $A$ . Об этом мы еще будем говорить в следующем разделе.

**Пример 11.2.1.** Пусть

$$A = \begin{bmatrix} -0.01 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0.01 \end{bmatrix}.$$

Если  $f(z) = e^z$  и  $g(z) = 1 + z + z^2/2$ , то  $\|f(A) - g(A)\| \approx 10^{-5}$  и в норме Фробениуса, и во второй норме. Поскольку  $\chi_2(X) \approx 10^7$ , погрешность  $O(1)$ , предсказанная теоремой 11.2.1, пессимистична. С другой стороны, погрешность, предсказанная на основе разложения Шура, есть  $O(10^{-2})$ .

### 11.2.3. Тейлоровы аппроксиманты

Популярный способ аппроксимации функции от матрицы, такой, как  $e^A$ , состоит в усечении ее ряда Тейлора. Условия, при которых функцию от матрицы  $f(A)$  можно представить рядом Тейлора, легко устанавливаются.

**Теорема 11.2.3.** Если функция  $f(z)$  представима степенным рядом

$$f(z) = \sum_{k=0}^{\infty} c_k z^k$$

в открытом круге, содержащем  $\lambda(A)$ , то

$$f(A) = \sum_{k=0}^{\infty} c_k A^k.$$

*Доказательство.* Мы докажем теорему для случая, когда матрица  $A$  диагонализуема. В задаче 11.2.1, мы подскажем, как обойтись без этого предположения. Пусть  $X^{-1}AX = D = \text{diag}(\lambda_1, \dots, \lambda_n)$ . Используя следствие 11.1.2, мы имеем

$$\begin{aligned} f(A) &= X \text{diag}(f(\lambda_1), \dots, f(\lambda_n)) X^{-1} = X \text{diag}\left(\sum_{k=0}^{\infty} c_k \lambda_1^k, \dots, \sum_{k=0}^{\infty} c_k \lambda_n^k\right) X^{-1} = \\ &= X \left( \sum_{k=0}^{\infty} c_k D^k \right) X^{-1} = \sum_{k=0}^{\infty} c_k (XDX^{-1})^k = \sum_{k=0}^{\infty} c_k A^k. \quad \square \end{aligned}$$

Некоторые важные трансцендентные функции от матрицы имеют особенно простые представления:

$$\ln(I - A) = \sum_{k=1}^{\infty} \frac{A^k}{k}, \quad |\lambda| < 1, \quad \lambda \in \lambda(A),$$

$$\sin(A) = \sum_{k=0}^{\infty} (-1)^k \frac{A^{2k+1}}{(2k+1)!},$$

$$\cos(A) = \sum_{k=0}^{\infty} (-1)^k \frac{A^{2k}}{(2k)!}.$$

Следующая теорема устанавливает границы погрешностей, возникающих, когда функции от матрицы, такие, как эти, аппроксимируют усеченными рядами Тейлора.

**Теорема 11.2.4.** Если функция  $f(z)$  представима рядом Тейлора

$$f(z) = \sum_{k=0}^{\infty} a_k z^k$$

в открытом круге, содержащем собственные значения матрицы  $A \in \mathbb{C}^{n \times n}$ , то

$$\left\| f(A) - \sum_{k=0}^q a_k A^k \right\|_2 \leq \frac{n}{(q+1)!} \max_{0 \leq s \leq 1} \|A^{q+1} f^{(q+1)}(As)\|_2.$$

*Доказательство.* Определим матрицу  $E(s)$  как

$$f(As) = \sum_{k=0}^q a_k (As)^k + E(s), \quad 0 \leq s \leq 1. \quad (11.2.3)$$

Если функция  $f_{ij}(s)$  есть  $(i,j)$ -й элемент матрицы  $f(As)$ , то она обязательно аналитична и, следовательно,

$$f_{ij}(s) = \left( \sum_{k=0}^q \frac{f_{ij}^{(k)}(0)}{k!} s^k \right) + \frac{f_{ij}^{(q+1)}(\varepsilon_{ij})}{(q+1)!} s^{q+1}, \quad (11.2.4)$$

где  $\varepsilon_{ij}$  удовлетворяет неравенствам  $0 \leq \varepsilon_{ij} \leq s \leq 1$ .

Сравнивая степени  $s$  в (11.2.3) и (11.2.4), мы заключаем, что  $e_{ij}(s)$  –  $(i,j)$ -й элемент матрицы  $E(s)$ , имеет вид

$$e_{ij}(s) = \frac{f_{ij}^{(q+1)}(\varepsilon_{ij})}{(q+1)!} s^{q+1}.$$

Функция  $f_{ij}^{(q+1)}(s)$  является  $(i,j)$ -м элементом матрицы  $A^{q+1} f^{(q+1)}(As)$  и, следовательно,

$$|e_{ij}(s)| \leq \max_{0 \leq s \leq 1} \frac{f_{ij}^{(q+1)}(s)}{(s+1)!} \leq \max_{0 \leq s \leq 1} \frac{\|A^{q+1} f^{(q+1)}(As)\|_2}{(q+1)!}.$$

Применяя неравенство (2.3.8), завершаем доказательство.  $\square$

**Пример 11.2.2.** Если

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix},$$

то

$$e^A = \begin{bmatrix} -0.735759 & 0.0551819 \\ -1.471518 & 1.103638 \end{bmatrix}.$$

Для  $q = 59$  теорема 11.2.4 предсказывает, что

$$\left\| e^A - \sum_{k=0}^q \frac{A^k}{k!} \right\|_2 \leq \frac{n}{(q+1)!} \max_{0 \leq s \leq 1} \|A^{q+1} e^{As}\|_2 \leq 10^{-60}.$$

Однако если использовать приближенную арифметику с  $\beta = 16$ ,  $t = 6$ , то мы найдем

$$\text{fl} \left( \sum_{k=0}^{59} \frac{A^k}{k!} \right) = \begin{bmatrix} -22.25880 & -1.4322766 \\ -61.49931 & -3.474280 \end{bmatrix}.$$

Проблема в том, что некоторые из частичных сумм имеют большие элементы. Например, матрица  $I + \dots + A^{17}/17!$  имеет элементы порядка  $10^7$ . Поскольку машинная точность равна примерно  $10^{-7}$ , при вычислениях появляются погрешности округления большие, чем норма решения.

Пример 11.2.2 отчетливо показывает недостаток аппроксимации усеченными рядами Тейлора: она хороша только при небольшом числе членов. Иногда эту проблему можно обойти, меняя скалярную функцию. Например, многократно применяя формулы двойных углов:

$$\cos(2A) = 2\cos(A)^2 - I, \quad \sin(2A) = 2\sin(A)\cos(A),$$

можно «построить» синус и косинус от матрицы из подходящих аппроксимаций усеченными рядами Тейлора:

```

 $S_0$  = тейлорова аппроксимация  $\sin(A/2^k)$ 
 $C_0$  = тейлорова аппроксимация  $\cos(A/2^k)$ 
for  $j = 1 : k$ 
     $S_j = 2S_{j-1}C_{j-1}$ 
     $C_j = 2C_{j-1}^2 - 1$ 
end

```

Здесь  $K$  – положительное целое, выбранное так, что, скажем,  $\|A\|_\infty \approx 2^k$ . Смотри [Serbin, Blalock, 1979].

#### 11.2.4. Оценка многочлена от матрицы

Коль скоро приближение трансцендентных функций матрицы так часто включает оценку многочленов, имеет смысл обратить внимание на детали вычисления многочлена вида

$$p(A) = b_0I + b_1A + \dots + b_qA^q,$$

где скаляры  $b_0, \dots, b_q \in \mathbb{R}$  заданы. Самый очевидный подход состоит в использовании схемы Горнера:

**Алгоритм 11.2.1.** Для заданной матрицы  $A$  и вектора  $b(0 : q)$  следующий алгоритм вычисляет  $F = b_qA^q + \dots + b_1A + b_0I$ .

```

 $F = b_qA + b_{q-1}I$ 
for  $k = q-2 : -1 : 0$ 
     $F = AF + b_kI$ 
end

```

Это требует  $q - 1$  матричных умножений. Однако в отличие от скалярного случая этот процесс суммирования не является оптимальным. Для того чтобы увидеть почему, положим  $q = 9$ . Можно заметить, что

$$p(A) = A^3(A^3(b_9A^3 + (b_8A^2 + b_7A + b_6I)) + (b_5A^2 + b_4A + b_3I) + b_2A^2 + b_1A + b_0I).$$

Следовательно, матрицу  $F = p(A)$  можно оценить всего лишь за четыре матричных умножения:

$$\begin{aligned}
 A_2 &= A^2, \\
 A_3 &= AA_2, \\
 F_1 &= b_9A_3 + b_8A_2 + b_7A + b_6I,
 \end{aligned}$$

$$F_2 = A_3 F_1 + b_5 A_2 + b_4 A + b_3 I,$$

$$F_3 = A_3 F_2 + b_2 A_2 + b_1 A + b_0 I.$$

В общем случае, если  $s$  – произвольное целое, удовлетворяющее неравенству  $1 \leq s \leq \sqrt{q}$ , то

$$p(A) = \sum_{k=0}^r B_k (A^s)^k, \quad r = [\lfloor q/s \rfloor]^1, \quad (11.2.5)$$

где

$$B_k = \begin{cases} b_{sk+s-1} A^{s-1} + \dots + b_{sk+1} A + b_{sk} I, & k = 0 : r-1 \\ b_q A^{q-sr} + \dots + b_{sr+1} A + b_r I, & k = r. \end{cases}$$

Если степени  $A^2, \dots, A^s$  вычислены, то правило Горнера можно применить к сумме (11.2.5). Окончательный результат состоит в том, что матрица  $p(A)$  может быть вычислена за  $s+r-1$  матричных умножений. Число матричных умножений примерно минимизирует  $s = [\sqrt{q}]$ . Этот подход обсуждается в работе [Paterson, Stockmeyer, 1973]. Ван Лоан [Van Loan, 1978b] показал, как эту процедуру можно применять, не заводя массивов для хранения матриц  $A^2, \dots, A^s$ .

### 11.2.5. Вычисление степеней матрицы

Задача возвведения матрицы в заданную степень заслуживает особого внимания. Пусть требуется вычислить  $A^{13}$ . Замечая, что  $A^4 = (A^2)^2$ ,  $A^8 = (A^4)^2$  и  $A^{13} = A^8 A^4 A$ , мы видим, что это можно выполнить всего за 5 матричных умножений. В общем случае имеем

**Алгоритм 11.2.2 (Двоичное возвведение в степень).** Для заданных положительного целого  $s$  и матрицы  $A \in \mathbb{R}^{n \times n}$  следующий алгоритм вычисляет матрицу  $F = A^s$ .

```

Пусть  $s = \sum_{k=0}^t \beta_k 2^k$  – двоичное представление числа  $s$  с  $\beta_t \neq 0$ 
 $Z = A; q = 0$ 
while  $\beta_q = 0$ 
     $Z = Z^2; q = q + 1$ 
end
 $F = Z$ 
for  $k = q + 1 : t$ 
     $Z = Z^2$ 
    if  $\beta_k \neq 0$ 
         $F = FZ$ 
    end
end
```

Этот алгоритм требует не более  $2 \lfloor \log_2(s) \rfloor$  матричных умножений. Если  $s$  – степень двойки, то необходимо только  $\log_2(s)$  матричных умножений.

<sup>1)</sup> Здесь и далее  $[x]$  – ближайшее целое большее либо равное  $x$ . – Прим. перев.

### 11.2.6. Интегральные функции от матрицы

Мы закончим этот раздел некоторыми замечаниями об интегральных функциях от матрицы. Пусть матрица  $f(At)$  определена при всех  $t \in [a, b]$  и мы хотим вычислить матрицу

$$F = \int_a^b f(At) dt.$$

Как и в (11.1.1), интегрирование является поэлементным.

Для вычисления матрицы  $F$  можно применять обычные квадратурные формулы. Например, применяя формулу Симпсона, имеем

$$F \approx \tilde{F} = \frac{h}{3} \sum_{k=0}^m w_k f(A(a + kh)), \quad (11.2.6)$$

где  $m$  четное,  $h = (b - a)/m$  и

$$w_k = \begin{cases} 1 & k = 0, m, \\ 4 & k \text{ нечетное}, \\ 2 & k \text{ четное, } k \neq 0, m. \end{cases}$$

Если производная  $(d^4/dz^4)f(zt) = f^{(4)}(zt)$  непрерывна при  $t \in [a, b]$  и если матрица  $f^{(4)}(At)$  определена на том же самом интервале, то можно показать, что  $\tilde{F} = F + E$ , где

$$\|E\|_2 \leq \frac{nh^4(b-a)}{180} \max_{a \leq t \leq b} \|f^{(4)}(At)\|_2. \quad (11.2.7)$$

Пусть  $f_{ij}$  и  $e_{ij}$  означают элементы  $(i, j)$  матриц  $F$  и  $E$  соответственно. При сделанных выше предположениях мы можем применить стандартную оценку погрешности формулы Симпсона и получить, что

$$|e_{ij}| \leq \frac{h^4(b-a)}{180} \max_{a \leq t \leq b} |e_i^T f^{(4)}(At) e_j|.$$

Неравенство (11.2.7) следует из неравенства  $\|E\|_2 \leq n \max |e_{ij}|$  и

$$\max_{a \leq t \leq b} |e_i^T f^{(4)}(At) e_j| \leq \max_{a \leq t \leq b} \|f^{(4)}(At)\|_2.$$

Конечно, в практических приложениях формулы (11.2.6), функцию, оценивающую  $f(A(a + kh))$ , обычно нужно аппроксимировать. Поэтому общая погрешность включает погрешность аппроксимации матрицы  $f(A(a + kh))$  наряду с погрешностью формулы Симпсона.

#### Задачи

**11.2.1.** (а) Пусть  $G = \lambda I + E$  – жорданов блок размера  $p \times p$ , где  $E(\sigma_{i,j-1})$ . Показать, что

$$(\lambda I + E)^k = \sum_{j=0}^{\min\{p-1, k\}} \binom{k}{j} \lambda^{k-j} E^j.$$

(б) Использовать (а) и теорему 11.1.1 для доказательства теоремы 11.2.3.

**11.2.2.** Проверить (11.2.2).

**11.2.3.** Проверить, что если  $\|A\|_2 < 1$ , то  $\ln(I + A)$  существует и удовлетворяет следующей оценке  $\|\ln(I + A)\|_2 \leq \|A\|_2/(1 - \|A\|_2)$ .

**11.2.4.** Пусть  $A$  – симметричная положительно определенная матрица размера  $n \times n$ .

(а) Показать, что существует единственная симметрическая положительно определенная матрица  $X$ , такая, что  $A = X^2$ . (б) Показать, что если  $X_0 = I$  и  $X_{k+1} = (X_k + AX_k^{-1})/2$ , то  $X_k \rightarrow \sqrt{A}$  квадратично, где  $\sqrt{A}$  означает матрицу  $X$  из (а).

**11.2.5.** Специализировать алгоритм 11.2.1 для случая, когда матрица  $A$  симметричная. Сделать то же самое для случая, когда  $A$  верхняя треугольная. Для обоих случаев подсчитать число флопов.

**11.2.6.** Показать, что матрица  $X(t) = C_1 \cos(t\sqrt{A}) + C_2 \sqrt{A^{-1}} \sin(t\sqrt{A})$  решает задачу Коши  $\dot{X}(t) = -AX(t)$ ,  $X(0) = C_1$ ,  $\dot{X}(0) = C_2$ . Считать, что матрица  $A$  симметричная, положительно определенная.

**11.2.7.** Используя теорему 11.2.4, оценить погрешность аппроксимаций:

$$\sin(A) \approx \sum_{k=0}^q (-1)^k \frac{A^{2k+1}}{(2k+1)!}, \quad \cos(A) \approx \sum_{k=0}^q (-1)^k \frac{A^{2k}}{(2k)!}.$$

### Замечания и литература к § 11.2

Оптимизация схемы Горнера для оценки многочлена от матрицы обсуждается в работах

Knuth D. (1981). *The Art of Computer Programming*, vol. 2. *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Massachusetts. [Имеется перевод: Кнут Д. Искусство программирования для ЭВМ. Т. 2: Полуисленные алгоритмы. — М.: Мир, 1977.]

Paterson M. S. and Stockmeyer L. J. (1973). "On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials", SIAM J. Comp. 2, 60–66.

Оценка многочлена от матрицы по схеме Горнера анализируется в работе

Van Loan C. F. (1978b). "A Note on the Evaluation of Matrix Polynomials", IEEE Trans. Auto. Cont. AC-24, 320–21.

Представления матрицы  $f(A)$  многочленами Ньютона и Лагранжа и их связь с другими определениями функции от матрицы обсуждается в

Rinehart R. F. (1955). "The Equivalence of Definitions of a Matric Function", Amer. Math. Monthly 62, 395–414.

Метод двойного угла для вычисления косинуса матрицы анализируется в работе

Serbin S. and Blalock S. (1979). "An Algorithm for Computing the Matrix Cosine", SIAM J. Sci. Stat. Comp. 1, 198–204.

Квадратный корень является особенно важной матричной функцией и здесь возможны несколько подходов. Смотри

Björck A. and Hammarling S. (1983). "A Schur Method for the Square Root of a Matrix", Lin. Alg. and Its Applic. 52/53, 127–140.

Higham N. J. (1986a). "Newton's Method for the Matrix Square Root", Math. Comp. 46, 537–550.

Higham N. J. (1987c). "Computing Real Square Roots of a Real Matrix", Lin. Alg. and Its Applic. 88/89, 405–430.

## 11.3. Матричная экспонента

Одной из наиболее часто вычисляемых функций от матрицы является экспонента

$$e^{At} = \sum_{k=0}^{\infty} \frac{(At)^k}{k!}.$$

Было предложено множество алгоритмов вычисления матрицы  $e^{At}$ , но, как указано в обзорной статье Молера и Ван Loана [Moler, Van Loan, 1978], большинство из них обладают сомнительными вычислительными свойствами. Для того чтобы пояснить, какие имеются вычислительные трудности, мы приведем краткий анализ

возмущения матричной экспоненты и затем используем его для оценки одного из лучших  $e^{At}$  алгоритмов.

### 11.3.1. Теория возмущения

Исходной точкой является задача Коши

$$\dot{X}(t) = AX(t), \quad X(0) = I,$$

где  $A, X(t) \in \mathbb{R}^{n \times n}$ . Эта задача, имеющая единственное решение  $X(t) = e^{At}$ , представляет собой характеристацию, которая может быть использована для установления тождества

$$e^{(A+E)t} - e^{At} = \int_0^t e^{A(t-s)} E e^{(A+E)s} ds.$$

Из этого тождества следует, что

$$\frac{\|e^{(A+E)t} - e^{At}\|_2}{\|e^{At}\|_2} \leq \frac{\|E\|_2}{\|e^{At}\|_2} \int_0^t \|e^{A(t-s)}\|_2 \|e^{(A+E)s}\|_2 ds.$$

Дальнейшие упрощения могут быть сделаны, если оценить нормы экспонент, стоящие под интегралом. Один из способов осуществить это – воспользоваться разложением Шура. Если  $Q^H A Q = \text{diag}(\lambda_i) + N$  – разложение Шура матрицы  $A \in \mathbb{C}^{n \times n}$ , то можно показать, что

$$\|e^{At}\|_2 \leq e^{\alpha(A)t} M_S(t),$$

где

$$\alpha(A) = \max \{\operatorname{Re}(\lambda) : \lambda \in \lambda(A)\}$$

и

$$M_S(t) = \sum_{k=0}^{n-1} \frac{\|Nt\|_2^k}{k!}.$$

Немного преобразованный, этот результат может быть использован для установления неравенства

$$\frac{\|e^{(A+E)t} - e^{At}\|_2}{\|e^{At}\|_2} \leq t \|E\|_2 M_S(t)^2 \exp(t M_S(t) \|E\|_2).$$

Заметим, что  $M_S(t) \equiv 1$  в том и только том случае, когда матрица  $A$  нормальная; это подсказывает, что проблема матричной экспоненты «хорошо себя ведет», если матрица  $A$  нормальная. Это наблюдение подтверждает поведение числа обусловленности матричной экспоненты  $v(A, t)$ , определенной как

$$v(A, t) = \max_{\|E\|_2 \leq 1} \left\| \int_0^1 e^{A(t-s)} E e^{As} ds \right\|_2 \frac{\|A\|_2}{\|e^{At}\|_2}.$$

Эта величина, рассмотренная в работе [Van Loan, 1977a], измеряет чувствительность отображения  $A \rightarrow e^{At}$ , поскольку для данного  $t$  существует матрица  $E$ , для которой

$$\frac{\|e^{(A+E)t} - e^{At}\|_2}{\|e^{At}\|_2} \approx v(A, t) \frac{\|E\|_2}{\|A\|_2}.$$

Следовательно, если величина  $v(A, t)$  велика, то малые изменения в матрице  $A$  могут

привести к относительно большим изменениям  $E^{At}$ . К несчастью, трудно определить точно те матрицы  $A$ , для которых величина  $v(A, t)$  большая. (Это контрастирует с системой линейных уравнений  $Ax = b$ , где плохая обусловленность матрицы  $A$  изящно описывается в терминах SVD.) Однако кое-что мы можем сказать, а именно, что  $v(A, t) \geq tA$ , где равенство имеет место для всех неотрицательных  $t$  в том и только том случае, когда матрица  $A$  нормальная.

### 11.3.2. Метод аппроксимации Паде

Продолжаем обсуждение, начатое в § 11.2: если  $g(z) \approx e^z$ , то  $g(A) \approx e^A$ . Очень полезным классом аппроксимантов для этой цели являются функции Паде, определяемые как

$$R_{pq}(z) = D_{pq}(z)^{-1} N_{pq}(z),$$

где

$$N_{pq}(z) = \sum_{k=0}^p \frac{(p+q-k)! p!}{(p+q)! k! (q-k)!} z^k$$

и

$$D_{pq}(z) = \sum_{k=0}^q \frac{(p+q-k)! p!}{(p+q)! k! (q-k)!} (-z)^k.$$

Заметим, что функция  $R_{pq}(z) = 1 + z + \dots + z^p/p!$  – это многочлен Тейлора степени  $p$ .

К несчастью, как показывает следующее тождество, хороши только первые аппроксиманты Паде:

$$e^A = R_{pq}(A) + \frac{(-1)^q}{(p+q)!} A^{p+q+1} D_{pq}(A)^{-1} \int_0^1 u^p (1-u)^q e^{A(1-u)} du. \quad (11.3.1)$$

Однако эту проблему можно обойти, используя тождество  $e^A = (e^{A/m})^m$ . В частности, мы можем масштабировать матрицу  $A$  путем деления на  $m$ , так, чтобы матрица  $F_{pq} = R_{pq}(A/m)$  достаточно точно приближала матрицу  $e^{A/m}$ . Затем мы вычислим  $F_{pq}^m$ , используя алгоритм 11.2.2. Если  $m$  – степень двойки, то этот алгоритм сводится к многократному возведению в квадрат и, таким образом, является очень эффективным. Успех всей процедуры зависит от точности аппроксимации

$$F_{pq} = \left( R_{pq} \left( \frac{A}{2^j} \right) \right)^{2^j}.$$

В работе [Moler, Van Loan, 1978] показано, что если

$$\frac{\|A\|_\infty}{2^j} \leq \frac{1}{2},$$

то существует матрица  $E \in \mathbf{R}^{n \times n}$ , такая, что

$$F_{pq} = e^{A+E},$$

$$AE = EA,$$

$$\|E\|_\infty \leq \varepsilon(p, q) \|A\|_\infty,$$

$$\varepsilon(p, q) = 2^{3-(p+q)} \frac{p! q!}{(p+q)! (p+q+1)!}.$$

Эти результаты составляют основу эффективной  $e^A$  процедуры с контролем

погрешности округления. Используя приведенную выше формулу, легко установить неравенство:

$$\frac{\|e^A - F_{pq}\|_\infty}{\|e^A\|_\infty} \leq \epsilon(p, q) \|A\|_\infty e^{\epsilon(p, q)} \|A\|_\infty.$$

Параметры  $p$  и  $q$  могут быть определены, исходя из некоторой заданной относительной погрешности. Заметим, что вычисление матрицы  $F_{pq}$  требует примерно  $j + \max(p, q)$  матричных умножений, поэтому имеет смысл положить  $p = q$ , так как этот выбор минимизирует  $\epsilon(p, q)$  при заданном числе операций. Собирая вместе эти идеи, мы получим

**Алгоритм 11.3.1.** Для заданного  $\delta > 0$  и матрицы  $A \in \mathbb{R}^{n \times n}$  следующий алгоритм вычисляет матрицу  $F = e^{A+E}$ , где  $\|E\|_\infty \leq \delta \|A\|_\infty$ .

$$\begin{aligned} j &= \max(0, 1 + \lfloor \log_2(\|A\|_\infty) \rfloor), \\ A &= A/2^j. \end{aligned}$$

Пусть  $q$  – наименьшее неотрицательное целое, такое, что  $D = I$ ;  $N = I$ ;  $X = I$ ;  $c = 1$

```
for k = 1 : q
    c = c(q - k + 1)/[(2q - k + 1)k]
    X = AX; N = N + cX; D = D + (-1)^k cX
end
```

Решить систему  $DF = N$  относительно  $F$ , используя исключение Гаусса.

```
for k = 1 : j
    F = F^2
end
```

Этот алгоритм требует примерно  $2(q + j + 1/3)n^3$  флоков.

Для того чтобы сделать быстрыми вычисления матриц  $D = D_{qq}(A)$  и  $N = N_{qq}(A)$ , можно обратиться к методу Горнера из § 11.2. Например, если  $q = 8$ , мы имеем  $N_{qq}(A) = U + AV$  и  $D_{qq}(A) = U - AV$ , где

$$U = c_0I + c_2A^2 + (c_4I + c_6A^2 + c_8A^4)A^4$$

и

$$V = c_1I + c_3A^2 + (c_5I + c_7A^2)A^4.$$

Понятно, что матрицы  $N$  и  $D$  можно найти за 5 матричных умножений вместо 7, требуемых алгоритмом 11.3.1.

Свойства погрешностей округления алгоритма 11.3.1, в основном, были исследованы Вардом [Ward, 1977]. Оценки, которые он вывел, удобны для вычисления и включены в фортрановскую подпрограмму.

### 11.3.3. Некоторые выводы об устойчивости

Есть несколько интересных вопросов, относящихся к устойчивости алгоритма 11.3.1. Одна из возможных проблем может возникнуть в процессе возвведения в квадрат, если матрица  $A$  является матрицей, чья экспонента растет, прежде чем начнет затухать. Если

$$G = R_{qq} \left( \frac{A}{2^j} \right) \approx e^{A/2^j},$$

то можно показать, что погрешность округления порядка

$$\gamma = \mathbf{u} \| G^2 \|_2 \| G^4 \|_2 \| G^8 \|_2 \dots \| G^{2^{j-1}} \|_2,$$

вероятно, испортит вычисленную матрицу  $G^{2^j}$ . Если  $\| e^{At} \|_2$  имеет существенный начальный рост, то это может быть именно тот случай, в котором

$$\gamma \gg \mathbf{u} \| G^{2^j} \|_2 \approx \mathbf{u} \| e^{At} \|_2,$$

что практически исключает возможность малых относительных погрешностей.

Если матрица  $A$  нормальная, то такой же будет матрица  $G$  и поэтому  $\| G^n \|_2 = \| G \|_2^n$  для всех положительных целых. Следовательно,  $\gamma \approx \mathbf{u} \| G^{2^j} \|_2 \approx \mathbf{u} \| e^{At} \|_2$  и, таким образом, проблема начального роста отпадает. По существу алгоритм может гарантировать получение малой относительной погрешности, когда матрица  $A$  нормальная. Значительно сложнее сделать выводы об этом методе, когда матрица  $A$  не является нормальной, так как связь между величиной  $v(A, t)$  и явлением начального роста не ясна. Однако, судя по некоторым экспериментам, представляется, что алгоритм 11.3.1 не в состоянии получить относительно точную матрицу  $e^{At}$ , только когда величина  $v(A, 1)$  достаточно велика.

### Задачи

**11.3.1.** Показать, что  $e^{(A+B)t} = e^{At}e^{Bt}$  для всех  $t$  в том и только том случае, когда  $AB = BA$ . (Подсказка: выразить обе части через степенные ряды по  $t$  и сравнить коэффициенты при  $t$ .)

**11.3.2.** Показать, что  $\lim_{t \rightarrow \infty} e^{At} = 0$  в том и только том случае, когда  $\alpha(A) < 0$ , где  $\alpha(A)$  определена как  $\alpha(A) = \max \{\operatorname{Re}(\lambda) : \lambda \in \lambda(A)\}$ .

**11.3.3.** Предположим, что матрица  $A$  кососимметрична. Показать, что матрица  $e^A$  и  $(1,1)$ -аппроксимация Паде  $R_{11}(A)$  ортогональные. Существуют ли какие-нибудь другие значения  $p$  и  $q$ , при которых матрица  $R_{pq}(A)$  ортогональная?

**11.3.4.** Показать, что если матрица  $A$  невырожденная, то существует матрица  $X$ , такая, что  $A = e^X$ . Единственна ли матрица  $X$ ?

**11.3.5.** Показать, что если

$$\exp\left(\begin{bmatrix} -A^T & P \\ 0 & A \end{bmatrix} z\right) = \begin{bmatrix} F_{11} & F_{12} \\ 0 & F_{22} \end{bmatrix}_n,$$

то

$$F_{11}^T F_{12} = \int_0^z e^{AT_t} Pe^{At} dt.$$

### Замечания и литература к § 11.3

Большую часть того, о чем упоминается в этом разделе, и обширную библиографию можно найти в следующей обзорной статье:

Moler C. B. and Van Loan C. F. (1978). "Nineteen Dubious Ways to Compute the Exponential of a Matrix", SIAM Review 20, 801–36.

Масштабирование и возвведение в квадрат Паде аппроксимантов (алгоритм 11.3.1) и удобная реализация метода Парлетта для разложения Шура (алгоритм 11.1.1) были найдены одними из наименее сомнительных среди девятнадцати тщательно изученных методов. Различные аспекты Паде аппроксимации матричной экспоненты обсуждаются в работах.

Fair W. and Luke Y. (1970). "Padé Approximations to the Operator Exponential", Numer. Math. 14, 379–82.

- Van Loan C. F. (1977a). "On the Limitation and Application of Padé Approximation to the Matrix Exponential", in Padé and Rational Approximation, ed. E. B. Saff and R. S. Varga, Academic Press, New York.
- Ward R. C. (1977). "Numerical Computation of the Matrix Exponential with Accuracy Estimate", SIAM J. Num. Anal. 14, 600–14.
- Wragg A. (1973). "Computation of the Exponential of a Matrix I: Theoretical Considerations", J. Inst. Math. Applic. 11, 369–75.
- Wragg A. (1975). "Computation of the Exponential of a Matrix II: Practical Considerations", J. Inst. Math. Applic. 15, 273–78.
- Доказательство уравнения (11.3.1) для скалярного случая появилось в работе
- Varga R. S. (1961). "On Higher-Order Stable Implicit Methods for Solving Parabolic Partial Differential Equations", J. Math. Phys. 40, 220–31.

В теории управления есть много приложений, требующих вычисления матричной экспоненты. Например, в задачах линейной регулярной оптимизации требуются различные интегралы, содержащие матричную экспоненту. Смотри

- Armstrong E. S. and Caglayan A. K. (1976). "An Algorithm for the Weighting Matrices in the Sample–Data Optimal Linear Regulator Problem", NASA Technical Note, TN D–8372.
- Johnson J. and Phillips C. L. (1971). "An Algorithm for the Computation of the Integral of the State Transition Matrix", IEEE Trans. Auto. Cont. AC–16, 204–5.
- Van Loan C. F. (1978a). "Computing Integrals Involving the Matrix Exponential", IEEE Trans. Auto. Cont. AC–16, 395–404.

Понимание отображения  $A \rightarrow \exp(At)$  и его чувствительности будет правильным, если оценить производительность алгоритмов вычисления матричной экспоненты. Работа в этом направлении включает

- Kågström B. (1977). "Bounds and Perturbation Bounds for the Matrix Exponential", BIT 17, 39–57.
- Van Loan C. F. (1977b). "The Sensitivity of the Matrix Exponential", SIAM J. Num. Anal. 14, 971–81.

Вычисление логарифма от матрицы – важная область, требующая слишком много работы. Эти вычисления возникают в различных задачах «распознавания систем». Смотри

- Singer B. and Spilerman S. (1976). "The Representation of Social Processes by Markov Models", Amer. J. Sociology 82, 1–54.

Helton B. W. (1968). "Logarithms of Matrices", Proc. Amer. Math. Soc. 19, 733–36.

### Добавления при переводе

Обратим внимание читателя на монографию одного из основоположников теории функций от матрицы:

Лаппо-Данилевский И. Л. Применение функций от матриц к теории линейных систем обыкновенных дифференциальных уравнений.– Москва.: Гос. изд-во технико-теоретической лит-ры. 1957.–456 с.

# Специальные разделы

В последней главе мы обсуждаем набор задач, в котором представлены важные приложения метода SVD и QR-разложения. Сначала мы рассмотрим минимизацию в смысле наименьших квадратов с ограничениями. В § 12.1 рассматриваются два типа ограничений – квадратичные неравенства и линейные равенства. Следующие два раздела связаны с вариантами стандартной LS-задачи. В § 12.2 мы рассмотрим, как может быть аппроксимирован вектор  $b$  некоторым подмножеством столбцов матрицы  $A$ . Этот способ иногда применяют, если матрица  $A$  неполного ранга. В § 12.3 мы рассматриваем вариант обычной регрессии, известной как общая задача наименьших квадратов, которая возникает, если матрица  $A$  искажена ошибками. Другие приложения метода SVD обсуждаются в § 12.4, где рассмотрены различные подобласти вычислений. В § 12.5 обсуждаются некоторые варианты симметричной проблемы собственных значений. В § 12.6 мы исследуем изменение QR-разложения, когда к матрице  $A = QR$  добавляется матрица ранга один.

## 12.1. Задача наименьших квадратов с ограничениями

Задачу наименьших квадратов иногда естественно рассматривать как минимизацию величины  $\|Ax - b\|_2$  на некотором подмножестве  $\mathbf{R}^n$ . Например, мы хотим определить вектор  $b$ , как наилучшее приближение для  $Ax$ , с ограничением, что  $x$  является единичным вектором. Или, возможно, решение определяет подходящую функцию  $f(t)$ , которая описывается значениями в конечном числе узлов. Это может привести к задаче наименьших квадратов с ограничениями типа равенств. В этом разделе мы покажем, как эти задачи могут быть решены с использованием QR-разложения и SVD.

### 12.1.1. Задача LSQI

Минимизация в смысле наименьших квадратов с ограничениями типа квадратных неравенств – задача *LSQI* – это подход, который можно использовать, когда решение обычной LS-задачи должно быть *регуляризовано*. Простой задачей LSQI, которая возникает при стремлении приблизить функцию с искаженными данными, является следующая:

$$\text{минимизировать } \|Ax - b\|_2 \text{ при условии } \|Bx\|_2 \leq a, \quad (12.1.1)$$

где  $A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^m$ ,  $B \in \mathbf{R}^{n \times n}$  (не вырождена) и  $a \geq 0$ . Ограничения определяют гиперэллипсоид в  $\mathbf{R}^n$  и обычно выбираются так, чтобы уменьшить сильные колебания в подходящей функции. Это может быть достигнуто, например, если матрица  $B$  является дискретным оператором второй производной.

В более общем случае мы имеем задачу

$$\text{минимизировать } \|Ax - b\|_2 \text{ при условии } \|Bx - d\|_2 \leq \alpha, \quad (12.1.2)$$

где  $A \in \mathbf{R}^{m \times n}$  ( $m \geq n$ ),  $b \in \mathbf{R}^m$ ,  $B \in \mathbf{R}^{p \times n}$ ,  $d \in \mathbf{R}^p$  и  $\alpha \geq 0$ . Обобщение сингулярного разложения в § 8.7.3 отвечает на вопрос о разрешимости задачи (12.1.2). Действительно, если

$$\begin{aligned} U^T A X &= \text{diag}(\alpha_1, \dots, \alpha_n), & U^T U &= I_m, \\ V^T B X &= \text{diag}(\beta_1, \dots, \beta_q), & V^T V &= I_p, & q &= \min\{p, n\}, \end{aligned} \quad (12.1.3)$$

является обобщением сингулярного разложения матриц  $A$  и  $B$ , то (12.1.2) преобразуется к виду

$$\text{минимизировать } \|D_A y - \tilde{b}\|_2 \text{ при условии } \|D_B y - \tilde{d}\|_2 \leq \alpha,$$

где  $\tilde{B} = U^T b$ ,  $\tilde{d} = V^T d$  и  $y = X^{-1}x$ . Простой вид целевой функции

$$\|D_A y - \tilde{b}\|_2^2 = \sum_{i=1}^n (\alpha_i y_i - \tilde{b}_i)^2 + \sum_{i=n+1}^m \tilde{b}_i^2 \quad (12.1.4)$$

и уравнений ограничения

$$\|D_B y - \tilde{d}\|_2^2 = \sum_{i=1}^r (\beta_i y_i - \tilde{d}_i)^2 + \sum_{i=r+1}^p \tilde{d}_i^2 \leq \alpha^2 \quad (12.1.5)$$

облегчает анализ задачи LSQI. Здесь  $r = \text{rank}(B)$  и мы предполагаем, что  $\beta_{r+1} = \dots = \beta_q = 0$ .

Во-первых, задача имеет решение тогда и только тогда, когда

$$\sum_{i=r+1}^p \tilde{d}_i^2 \leq \alpha^2.$$

Если в этом выражении имеется равенство, рассмотрение формул (12.1.4) и (12.1.5) показывает, что вектор, определяемый соотношением

$$y_i = \begin{cases} \tilde{d}_i / \beta_i & i = 1 : r, \\ \tilde{b}_i / \alpha_i & i = r + 1 : n, \alpha_i \neq 0, \\ 0 & i = r + 1 : n, \alpha_i = 0 \end{cases} \quad (12.1.6)$$

разрешает задачу LSQI. В противном случае

$$\sum_{i=r+1}^p \tilde{d}_i^2 < \alpha^2 \quad (12.1.7)$$

и мы имеем несколько альтернатив для продолжения. В этой ситуации вектор  $y \in \mathbf{R}^n$ , определяемый соотношением

$$y_i = \begin{cases} \tilde{b}_i / \alpha_i & \alpha_i \neq 0, \\ \tilde{d}_i / \beta_i & \alpha_i = 0, \end{cases} \quad i = 1 : n$$

будет минимизатором выражения  $\|D_A y - \tilde{b}\|_2$ . Если такой вектор существует, то мы имеем решение задачи (12.1.2). (Однако, это не является необходимым условием минимума 2-нормы.) Поэтому мы допускаем, что

$$\sum_{\substack{i=1 \\ \alpha_i \neq 0}}^q \left( \beta_i \frac{\tilde{b}_i}{\alpha_i} - \tilde{d}_i \right)^2 + \sum_{i=q+1}^p \tilde{d}_i^2 > \alpha^2. \quad (12.1.8)$$

Отсюда следует, что решение задачи LSQI имеет место на границе допустимого

множества. Таким образом, наша окончательная цель

минимизировать  $\|D_A y - \tilde{b}\|_2$  при условии  $\|D_B y - \tilde{d}\|_2 = a$ .

Для решения этой задачи используем метод лагранжевых множителей. Определив

$$h(\lambda, y) = \|D_A y - \tilde{b}\|_2^2 + \lambda (\|D_B y - \tilde{d}\|_2^2 - a^2),$$

мы видим, что уравнения  $0 = \partial h / \partial y_i$ ,  $i = 1 : n$ , приводят к линейной системе

$$(D_A^T D_A + \lambda D_B^T D_B) y = D_B^T \tilde{b} + \lambda D_B^T \tilde{d}.$$

В предположении, что матрица коэффициентов невырождена, система имеет решение  $y(\lambda)$ , где

$$y_i(\lambda) = \begin{cases} \frac{a_i \tilde{b}_i + \lambda \beta_i \tilde{d}_i}{a_i^2 + \lambda \beta_i^2}, & i = 1 : q, \\ \tilde{b}_i / a_i, & i = q + 1 : n. \end{cases}$$

Для определения множителей Лагранжа обозначим

$$\phi(\lambda) \equiv \|D_B y(\lambda) - \tilde{d}\|_2^2 = \sum_{i=1}^r a_i \frac{\beta_i \tilde{b}_i - a_i \tilde{d}_i^2}{a_i^2 + \lambda \beta_i^2} + \sum_{i=r+1}^n \tilde{d}_i^2$$

и потом найдем решения уравнения  $\phi(\lambda) = a^2$ . Уравнения такого вида называются *секулярными уравнениями*. (См. § 8.6.) Из формулы (12.1.8) мы видим, что  $\phi(0) > a^2$ . Функция  $\phi(\lambda)$  является монотонно убывающей при  $\lambda > 0$ , и поэтому из формулы (12.1.8) вытекает существование единственного положительного  $\lambda^*$ , для которого  $\phi(\lambda^*) = a^2$ . Легко показать, что это необходимый корень. Он может быть найден применением какого-либо из стандартных методов нахождения корней, таких, как метод Ньютона. Тогда решением исходной задачи LSQI является вектор  $x = X y(\lambda^*)$ .

## 12.1.2. LS-минимизация на сфере

Для возможного случая минимизации на сфере мы имеем следующую процедуру ( $B = I_n$ ,  $d = 0$ ).

**Алгоритм 12.1.1.** Для данной матрицы  $A \in \mathbb{R}^{m \times n}$ , где  $m \geq n$ , вектора  $b \in \mathbb{R}^m$  и параметра  $a > 0$  следующий алгоритм вычисляет вектор  $x \in \mathbb{R}^n$ , такой, что  $\|Ax - b\|_2$  достигает минимума при ограничении  $\|x\|_2 \leq a$ .

Вычислить SVD  $A = U \Sigma V^T$  и запомнить  $V = [v_1, \dots, v_n]$ .

$b = U^T b$ ;  $r = \text{rank}(A)$

**if**  $\sum_{i=1}^r \left( \frac{b_i}{\sigma_i} \right)^2 > a^2$

Найти  $\lambda^*$ , такое, что  $\sum_{i=1}^r \left( \frac{\sigma_i b_i}{\sigma_i^2 + \lambda^*} \right)^2 = a^2$ .

$x = \sum_{i=1}^r \left( \frac{\sigma_i b_i}{\sigma_i^2 + \lambda^*} \right) v_i$

**else**  $x = \sum_{i=1}^r \left( \frac{b_i}{\sigma_i} \right) v_i$

**end**

На процедуру SVD приходится преобладающая часть вычислений в этом алгоритме.

**Пример 12.1.1.** Характеристическое уравнение задачи

$$\min_{\|x\|_2=1} \left\| \begin{bmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix} \right\|_2$$

задается выражением

$$\left( \frac{8}{\lambda + 4} \right)^2 + \left( \frac{2}{\lambda + 1} \right)^2 = 1.$$

Для этой задачи находим  $\lambda^* = 4.57132$  и  $x = (0.93334, 0.35898)^T$ .

### 12.1.3. Гребневая регрессия

Задача, решаемая алгоритмом 12.1.1, эквивалентна проблеме множителей Лагранжа для определения  $\lambda > 0$ , такого, что

$$(A^T A + \lambda I)x = A^T b \quad (12.1.9)$$

и  $\|x\|_2 = a$ . Здесь формула (12.1.9) является точным нормальным уравнением, сформулированным для задачи гребневой регрессии

$$\min_x \left\| \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 = \min_x \|Ax - b\|_2^2 + \lambda \|x\|_2^2.$$

Для обобщенной задачи гребневой регрессии есть некоторый способ выбора гребневого параметра  $\lambda$ , например  $\|x(\lambda)\|_2 = a$  при некотором заданном  $a$ . Мы опишем процедуру  $\lambda$ -выбора, которая обсуждается в работе Golub, Heath и Wahba (1979).

Пусть  $D_k = I - e_k e_k^T = \text{diag}(1, \dots, 1, 0, 1, \dots, 1) \in \mathbb{R}^{m \times m}$ , и пусть  $x_k(\lambda)$  разрешает задачу

$$\min_x \|D_k(Ax - b)\|_2^2 + \lambda \|x\|_2^2. \quad (12.1.10)$$

Таким образом,  $x_k(\lambda)$  – это решение задачи гребневой регрессии при вычеркнутых  $k$ -й строки матрицы  $A$  и  $k$ -й компоненты вектора  $b$ , т. е.  $k$ -й эксперимент игнорируется. Теперь рассмотрим выбор такого  $\lambda$ , которое минимизирует квадрат ошибки кросс-валидации с весами  $C(\lambda)$ <sup>1)</sup>, задаваемый формулой

$$C(\lambda) = \frac{1}{m} \sum_{k=1}^m \omega_k (a_k x_k(\lambda) - b_k)^2.$$

Здесь  $\omega_1, \dots, \omega_m$  – неотрицательные веса и  $a_k^T$  – это  $k$ -я строка матрицы  $A$ . Замечая, что

$$\|Ax_k(\lambda) - b\|_2^2 = \|D_k(Ax_k(\lambda) - b)\|_2^2 + (a_k^T x_k(\lambda) - b_k)^2$$

мы видим, что слагаемое  $[a_k^T x_k(\lambda) - b_k]^2$  увеличивает результирующую сумму квадратов, если  $k$ -я строка «восстановлена». Минимизация  $C(\lambda)$  равносильна выбору такого  $\lambda$ , что окончательная модель не зависит от какого-либо одного эксперимента.

Более строгий анализ может сделать это утверждение точным и предложить также метод для минимизации  $C(\lambda)$ . При условии что  $\lambda > 0$ , алгебраические

<sup>1)</sup> Кросс-валидация называется также *перекрестной проверкой* или *перекрестным экзаменом*. – Прим. перев.

преобразования дают формулу

$$x_k(\lambda) = x(\lambda) + \frac{a_k^T x(\lambda) - b_k}{1 - z_k^T a_k} z_k, \quad (12.1.1)$$

где  $z_k = (A^T A + \lambda I)^{-1} a_k$  и  $x(\lambda) = (A^T A + \lambda I)^{-1} A^T b$ . Умножение формулы (12.1.11) на  $-a_k^T$  и прибавление  $b_k$  к каждой части текущего уравнения дают

$$b_k - a_k x_k(\lambda) = \frac{e_k^T (I - A(A^T A + \lambda I)^{-1} A^T) b}{e_k^T (I - A(A^T A + \lambda I)^{-1} A^T) e_k}. \quad (12.1.12)$$

Замечая, что невязка  $r = (r_1, \dots, r_m)^T = b - Ax(\lambda)$  задается формулой  $r = [I - A(A^T A + \lambda I)^{-1} A^T] b$ , мы видим, что

$$C(\lambda) = \frac{1}{m} \sum_{k=1}^m \omega_k \left( \frac{r_k}{\partial r_k / \partial b_k} \right)^2.$$

Отношение  $r_k / (\partial r_k / \partial b_k)$  может рассматриваться как обратная величина «попадания»  $k$ -го наблюдения  $b_k$  на модель. Когда  $\partial r_k / \partial b_k$  мало, это говорит о том, что ошибка в модельном предсказании  $b_k$  является в некотором смысле независимой от  $b_k$ .

Практическое определение параметра  $\lambda^*$  осуществляется простым вычислением SVD-матрицы  $A$ . Действительно, если  $U^T A V = \text{diag}(\sigma_1, \dots, \sigma_n)$  при  $\sigma_1 \geq \dots \geq \sigma_n$  и  $\tilde{b} = U^T b$ , тогда можно показать с учетом формулы (12.1.12), что

$$C(\lambda) = \frac{1}{m} \sum_{k=1}^m \omega_k \left[ \frac{\tilde{b}_k - \sum_{j=1}^r u_{kj} \tilde{b}_j \left( \frac{\sigma_j^2}{\sigma_j^2 + \lambda} \right)}{1 - \sum_{j=1}^r u_{kj}^2 \left( \frac{\sigma_j^2}{\sigma_j^2 + \lambda} \right)} \right]^2.$$

Минимизация этого выражения обсуждается в работе Golub, Heath и Wahba (1979).

#### 12.1.4. Задача наименьших квадратов с ограничениями типа равенств

Мы завершаем раздел рассмотрением задачи наименьших квадратов с ограничениями типа линейных равенств:

$$\min_{Bx=d} \|Ax - b\|_2. \quad (12.1.13)$$

Здесь  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{p \times n}$ ,  $b \in \mathbb{R}^m$ ,  $d \in \mathbb{R}^p$  и  $\text{rank}(B) = p$ . Мы называем задачу (12.1.13) задачей LSE. Полагая  $a = 0$  в задаче (12.1.2), мы видим, что задача LSE является частным случаем задачи LSQI. Однако проще решать задачу LSE непосредственно, а не через множители Лагранжа.

Предположим для ясности, что обе матрицы  $A$  и  $B$  имеют полный ранг. Пусть

$$Q^T B^T = \begin{bmatrix} R \\ 0 \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}$$

—QR-разложение матрицы  $B^T$ , и пусть

$$AQ = [A_1 \ A_2] \begin{matrix} p \\ n-p \end{matrix}, \quad Q^T x = \begin{bmatrix} y \\ z \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}.$$

Очевидно, что с учетом этих преобразований формула (12.1.3) переходит в соотношение

$$\min_{R^T y = d} \|A_1 y + A_2 z - b\|_2.$$

Таким образом, вектор  $y$  определяется из уравнения ограничений  $R^T y = d$ , а вектор  $z$  получается решением задачи LS без ограничений

$$\min_z \|A_2 z + (b - A_1 y)\|_2.$$

Комбинируя полученное выше, видим, что вектор  $x = Q \begin{bmatrix} y \\ z \end{bmatrix}$  разрешает задачу (12.1.13).

**Алгоритм 12.1.2.** Пусть  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{p \times n}$ ,  $b \in \mathbb{R}^m$  и  $d \in \mathbb{R}^p$ . Если  $\text{rank}(A) = n$  и  $\text{rank}(B) = p$ , тогда следующий алгоритм минимизирует величину  $\|Ax - b\|_2$  при ограничении  $Bx = d$ .

$$B^T = QR \quad (\text{QR-разложение})$$

Решить  $R(1:p, 1:p)^T y = d$  относительно  $y$ .

$$A = AQ$$

Найти  $z$ , такое, что  $\|A(:, p+1:n)z - (b - A(:, 1:p)y)\|_2$  является минимальным  
 $x = Q(:, 1:p)y + Q(:, p+1:n)z$

Отметим, что такой подход к задаче LSE включает два разложения и матричное умножение.

### 12.1.5. Метод взвешивания

Интересным способом получения приближенного решения задачи (12.1.13) является решение задачи LS без ограничений

$$\min_x \left\| \begin{bmatrix} A \\ \lambda B \end{bmatrix} x - \begin{bmatrix} b \\ \lambda d \end{bmatrix} \right\|_2 \quad (12.1.14)$$

для больших  $\lambda$ . Обобщенное сингулярное разложение из § 8.7.3 отвечает на вопрос о качестве приближения. Пусть

$$U^T AX = \text{diag}(\alpha_1, \dots, \alpha_n) = D_A \in \mathbb{R}^{m \times n},$$

$$V^T BX = \text{diag}(\beta_1, \dots, \beta_p) = D_B \in \mathbb{R}^{p \times n}$$

суть GSVD пары матриц  $(A, B)$ , и предположим для ясности, что обе матрицы имеют полный ранг. Если  $U = [u_1, \dots, u_m]$ ,

$$V = [v_1, \dots, v_p], \quad X = [x_1, \dots, x_n],$$

то легко показать, что

$$x = \sum_{i=1}^p \frac{v_i^T d}{\beta_i} x_i + \sum_{i=p+1}^n \frac{u_i^T b}{\alpha_i} x_i \quad (12.1.15)$$

является точным решением для задачи (12.1.13), тогда как

$$x(\lambda) = \sum_{i=1}^p \frac{\alpha_i u_i^T b + \lambda^2 \beta_i^2 v_i^T d}{\alpha_i^2 + \lambda^2 \beta_i^2} x_i + \sum_{i=p+1}^n \frac{u_i^T b}{\alpha_i} x_i \quad (12.1.16)$$

разрешает (12.1.14). Так как

$$x(\lambda) - x = \sum_{i=1}^p \frac{\alpha_i (\beta_i u_i^T b - \alpha_i v_i^T d)}{\beta_i (\alpha_i^2 + \lambda^2 \beta_i^2)} x_i,$$

отсюда следует, что  $x(\lambda) \rightarrow x$  при  $\lambda \rightarrow \infty$ .

Привлекательность этого подхода к задаче LSE состоит в том, что не требуются специальные подпрограммы: используется только подпрограмма решения обычной LS-задачи. Однако при больших значениях  $\lambda$  могут возникнуть численные проблемы и необходимо быть осторожным. См. Powell и Reid (1968) и Van Loan (1982a).

**Пример 12.1.2.** Задача

$$\min_{x_1 = x_2} \left\| \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 7 \\ 1 \\ 3 \end{bmatrix} \right\|_2,$$

имеет решение  $x = (0,3407821, 0,3407821)$ . Оно может быть приближено решением задачи

$$\min \left\| \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 1000 & -1000 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 7 \\ 1 \\ 3 \\ 0 \end{bmatrix} \right\|_2,$$

которое имеет решение  $x = (0,3407810, 0,3407829)$ .

### Задачи

**12.1.1.** (a) Показать, что если  $\text{null}(A) \cap \text{null}(B) \neq \{0\}$ , то (12.1.2) не может иметь единственное решение.

(b) Построить пример, который показывает, что обратное неверно. (Подсказка: допустимо  $A^+ b$ .)

**12.1.2.** Пусть  $p_0(x), \dots, p_n(x)$  – заданные полиномы,  $(x_0, y_0), \dots, (x_m, y_m)$  заданное множество координатных пар, где  $x_i \in [a, b]$ . Требуется найти полином  $p(x) = \sum_{k=0}^n a_k p_k(x)$ , такой, что  $\sum_{i=0}^m (p(x_i) - y_i)^2$  является минимальной при ограничении

$$\int_a^b [p''(x)]^2 dx \approx h \sum_{i=0}^N \left( \frac{p(z_{i-1}) - 2p(z_i) + p(z_{i+1})}{h^2} \right)^2 \leq a^2,$$

где  $z_i = a + ih$  и  $b = a + Nh$ . Показать, что такая постановка приводит к задаче LSQI вида (12.1.1).

**12.1.3.** Пусть матрица  $Y = [y_1, \dots, y_k] \in R^{m \times k}$  обладает следующим свойством

$$Y^T Y = \text{diag}(d_1^2, \dots, d_k^2), \quad d_1 \geq d_2 \geq \dots \geq d_k > 0.$$

Показать, что если  $Y = QR$  является QR-разложением матрицы  $Y$ , тогда  $R$  – диагональная матрица, причем  $|r_{ii}| = d_i$ .

**12.1.4.** (a) Показать, что если  $(A^T A + \lambda I)x = A^T b$ ,  $\lambda > 0$  и  $\|x\|_2 = a$ , то вектор  $z = (Ax - b)/\lambda$  разрешает двойственные уравнения  $(AA^T + \lambda I)z = -b$ , где  $\|A^T z\|_2 = a$ .

(b) Показать, что если  $(AA^T + \lambda I)z = -b$ ,  $\|A^T z\|_2 = a$ , то вектор  $x = -A^T z$  удовлетворяет уравнению  $(A^T A + \lambda I)x = A^T b$ ,  $\|x\|_2 = a$ .

**12.1.5.** Пусть  $A$  – это  $m \times 1$  матрица из единиц, и пусть  $b \in R^m$ . Показать, что метод кросс-валидации с единичными весами предписывает оптимальное  $\lambda$ , задаваемое соотношением

$$\lambda = \left( \left( \frac{\tilde{b}}{s} \right)^2 - \frac{1}{m} \right)^{-1},$$

где  $\tilde{B}^T = (b_1 + \dots + b_m)/m$  и  $s = \sum_{i=1}^m (b_i - \tilde{B})^2/(m-1)$ .

**12.1.6.** Вывести уравнение (12.1.15) из формулы (12.1.17).

**12.1.7.** Получить SVD-версию алгоритма 12.1.2, которая может работать с матрицами  $A$  и  $B$  неполного ранга.

**12.1.8.** Пусть

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix},$$

где  $A_1 \in \mathbb{R}^{n \times n}$  является невырожденной и  $A_2 \in \mathbb{R}^{(m-n) \times n}$ . Показать, что

$$\sigma_{\min}(A) \geq \sqrt{1 + \sigma_{\min}(A_2 A_1^{-1})^2} \sigma_{\min}(A_1).$$

**12.1.9.** Рассмотрим задачу

$$\min_{\substack{x^T B x = \beta^2 \\ x^T C x = \gamma^2}} \|Ax - b\|_2, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m, \quad B, C \in \mathbb{R}^{n \times n}.$$

Предположим, что матрицы  $B$  и  $C$  положительно определенные и что  $z \in \mathbb{R}^{n \times n}$  невырожденная матрица, удовлетворяющая условию  $Z^T B Z = \text{diag}(\lambda_1, \dots, \lambda_n)$  и  $Z^T C Z = I_n$ . И пусть  $\lambda_1 \geq \dots \geq \lambda_n$ .  
(a) Показать, что множество допустимых  $x$  пусто, если не выполнено условие  $\lambda_n \leq \beta^2/\gamma^2 \leq \lambda_1$ .  
(b) Используя  $Z$ , показать, как задача с двумя ограничениями может быть сведена к задаче с одним ограничением вида

$$\min_{y^T W y = \beta^2 - \lambda_n \gamma^2} \|\tilde{A}x - b\|_2,$$

где  $W = \text{diag}(\lambda_1, \dots, \lambda_n) - \lambda_n I$ .

### Замечания и литература к § 12.1

Грубо говоря, регуляризация – это способ преобразования плохо обусловленной задачи к устойчивой. Важным примером является задача наименьших квадратов с квадратичными ограничениями. См. работы

Eldén L. (1977a). "Algorithms for the Regularization of III - Conditioned Least Squares Problems", BIT 17, 134–45.

Eldén L. (1977b). "Numerical Analysis of regularization and Constrained Least Square Methods", Ph. D. thesis, Linköping Studies in Science and Technology, Dissertation no. 20, Linköping, Sweden.

Литература по кросс-валидации содержится в работах

Eldén L. (1985). "A Note on the Computation of the generalized Cross-Validation Function for III-Conditioned Least Squares Problems", BIT 24, 467–472.

Golub G. H., Heath M., and Wahba G. (1979). "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter", Technometrics 21, 215–23.

Задача LSQI обсуждается в статьях

Eldén L. (1980). "Perturbation Theory for the Least Squares Problem with Linear Equality Constraints", SIAM J. Num. Anal. 17, 338–50.

Eldén L. (1983). "A Weighted Pseudoinverse, Generalized Singular Values, and Constrained Least Squares Problems", BIT 22, 487–502.

Forsythe G. E. and Golub G. H. (1965). "On the Stationary Values of a Second - Degree Polynomial on the Unit Sphere", SIAM J. App. Math. 14, 1050–68.

Gander W. (1981). "Least Squares with a Quadratic Constraint", Numer. Math. 36, 291–307.

Stewart G. W. (1984). "On the Asymptotic Behavior of Scaled Singular Value and QR Decompositions", Math. Comp. 43, 483–490.

Другие вычислительные аспекты задачи LSQI включают обработку ленточных и разреженных задач. См.

- Björck A. (1984). "A General Updating Algorithm for Constrained Linear Least Squares Problems", SIAM J. Sci. and Stat. Comp. 5, 394–402.  
 Eldén L. (1984). "An Algorithm for the Regularization of Ill-Conditioned, Banded Least Squares Problems", SIAM J. Sci. and Stat. Comp. 5, 237–254.  
 O'Leary D. P. and Simmons J. A. (1981). "A Bidiagonalization–Regularization Procedure for Large Scale Discretizations of Ill–Posed Problems", SIAM J. Sci. and Stat. Comp. 2, 474–89.  
 Schittkowski K. and Stoer J. (1979). "A Factorization Method for the Solution of Constrained Linear Least Squares Problems Allowing for Subsequent Data Changes", Numer. Math. 31, 431–63.

Задача LSE обсуждается в работе Lawson и Hansen (SLE, гл. 22). Задачи, связанные с методом взвешивания, анализируются в статьях

- Barlow J. L., N. K. Nichols, and R. J. Plemmons (1988). "Iterative Methods for Equality Constrained Least Squares Problems", SIAM J. Sci. and Stat. Comp. 9, 892–906.  
 Powell M. J. D. and Reid J. K. (1968). "On Applying Householder's Method to Linear Least Squares Problems", Proc. IFIP Congress, pp. 122–26.  
 Van Loan C. (1985). "On the Method of Weighting for Equality Constrained Least Squares Problems", SIAM Numer. Anal. 22, 851–864.  
 GSVD также полезно для изучения обобщенных задач наименьших квадратов; см.  
 Paige C. C. (1985). "The General Limit Model and the Generalized Singular Value Decomposition", Lin. Alg. and Its Applic. 70, 269–284.

## 12.2. Выбор подмножества при помощи SVD

Как упоминалось в § 5.5, LS-задача неполного ранга  $\min \|Ax - b\|_2$  может быть заменена аппроксимацией решения минимальной нормы

$$x_{LS} = \sum_{i=1}^r \frac{u_i^T b}{\sigma_i} v_i, \quad r = \text{rank}(A),$$

с вектором

$$x_{\tilde{r}} = \sum_{i=1}^{\tilde{r}} \frac{u_i^T b}{\sigma_i} v_i, \quad \tilde{r} \leq r,$$

где

$$A = U \Sigma V^T = \sum_{i=1}^r \sigma_i u_i v_i^T. \quad (12.2.1)$$

SVD-разложение матрицы  $A$  и  $\tilde{r}$  – вычисленная оценка для  $r$ . Отметим, что вектор  $x_{\tilde{r}}$  минимизирует  $\|A_{\tilde{r}}x - b\|_2$ , где матрица

$$A_{\tilde{r}} = \sum_{i=1}^{\tilde{r}} \sigma_i u_i v_i^T$$

является приближением матрицы  $A$  ранга  $\tilde{r}$ . См. теорему 2.5.2.

Замена матрицы  $A$  на матрицу  $A_{\tilde{r}}$  в LS-задаче выполняется для фильтрации малых сингулярных значений и имеет большой смысл в ситуациях, когда матрица  $A$  получается из возмущенных данных. Однако в других приложениях неполнота ранга означает избыточность факторов, которые содержит лежащая в основе модель. В этом случае для создателя модели может быть неинтересным такое приближение, как  $A_{\tilde{r}}x_{\tilde{r}}$ , которое учитывает все  $n$  избыточных факторов. Вместо этого можно найти приближение  $Ay$ , где  $y$  имеет около  $\tilde{r}$  ненулевых компонент.

Позиции ненулевых элементов вектора определяют некоторые столбцы матрицы  $A$ , т. е. некоторые факторы модели, которые используются для приближения вектора наблюдений  $b$ . Задача выбора этих столбцов называется *выбором подмножества* и составляет содержание данного раздела.

### 12.2.1. Метод QR со столбцовым выбором

Метод QR со столбцовым выбором можно рассматривать как метод выбора независимого подмножества столбцов матрицы  $A$ , которыми приближается вектор  $b$ . Предположим, мы применяем алгоритм 5.4.1 к матрице  $A \in \mathbb{R}^{m \times n}$  и вычисляем ортогональную матрицу  $Q$  и перестановочную матрицу  $\Pi$ , такие, что матрица  $R = Q^T A \Pi$  является верхней треугольной. Если  $R(1:\tilde{r}, 1:\tilde{r})z = \tilde{b}(1:\tilde{r})$ , где  $\tilde{b} = Q^T b$ , то полагаем

$$y = \Pi \begin{bmatrix} z \\ 0 \end{bmatrix},$$

тогда вектор  $Ay$  является LS-приближением вектора  $b$ , содержащим первые  $\tilde{r}$  столбцов матрицы  $A\Pi$ .

### 12.2.2. Использование метода SVD

Хотя метод QR со столбцовым выбором является достаточно надежным способом для решения задач, близких к задачам неполного ранга, иногда по причинам, рассмотренным в § 5.5, более предпочтительным является метод SVD. Поэтому мы опишем процедуру выбора подмножества, основанного на SVD, в соответствии с тем, что предложили Golub, Klemm и Stewart (1976):

- Вычислить SVD, матрицы  $A = U\Sigma V^T$ , использовать его для определения оценки ранга  $\tilde{r}$ .
- Вычислить перестановочную матрицу  $P$ , такую, что столбцы матрицы  $B_1 \in \mathbb{R}^{m \times \tilde{r}}$  в представлении  $AP = [B_1, B_2]$  «достаточно независимы».
- Приблизить вектор  $b$  вектором  $Ay$ , где  $y = P \begin{bmatrix} z \\ 0 \end{bmatrix}$  и вектор  $z \in \mathbb{R}^{\tilde{r}}$  минимизирует  $\|B_1 z - b\|_2$ .

Второй шаг является ключевым. Так как

$$\min_{z \in B^{\tilde{r}}} \|B_1 z - b\|_2 = \|Ay - b\|_2 \geq \min_{x \in \mathbb{R}^n} \|Ax - b\|_2,$$

то можно сделать вывод, что матрицу перестановок  $P$  необходимо выбирать так, чтобы сделать невязку  $(I - B_1 B_1^+)b$  предельно малой. К несчастью, такой способ решения является неустойчивым. Например,

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 + \varepsilon & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix},$$

$\tilde{r} = 2$  и  $P = I$ , тогда  $\min_{z \in B^{\tilde{r}}} \|B_1 z - b\|_2 = 0$ , но  $\|B_1^+ b\|_2 = O(1/\varepsilon)$ . С другой стороны, любое подмножество, содержащее третий столбец матрицы  $A$ , является строго независимым, но приводит к значительно худшей невязке. Этот пример показывает, что возможно противоречие между независимостью выбранных столбцов и нормой невязки, которую они порождают. Для принятия решения при возникновении

подобных противоречий требуется дополнительный математический аппарат в виде приемлемых оценок для  $\sigma_{\tilde{r}}(B_1)$ , наименьшего сингулярного значения матрицы  $B_1$ .

**Теорема 12.2.1.** Пусть SVD матрицы  $A \in \mathbb{R}^{m \times n}$  задается соотношением (12.2.1), определим матрицу  $B_1 \in \mathbb{R}^{m \times \tilde{r}}$ ,  $\tilde{r} \leq \text{rank}(A)$  следующим образом

$$AP = \begin{bmatrix} B_1 & B_2 \\ \tilde{r} & n - \tilde{r} \end{bmatrix},$$

где  $P \in \mathbb{R}^{n \times n}$  матрица перестановок. Если

$$P^T V = \begin{bmatrix} \tilde{V}_{11} & \tilde{V}_{12} \\ \tilde{V}_{21} & \tilde{V}_{22} \end{bmatrix} \begin{matrix} \tilde{r} \\ n - \tilde{r} \end{matrix} \quad (12.2.2)$$

и  $\tilde{V}_{11}$  невырождена, то

$$\frac{\sigma_{\tilde{r}}(A)}{\|\tilde{V}_{11}^{-1}\|_2} \leq \sigma_{\tilde{r}}(B_1) \leq \sigma_{\tilde{r}}(A).$$

*Доказательство.* Верхняя оценка следует из минимаксных характеристик сингулярных значений, полученных в § 8.3.1.

Чтобы установить нижнюю оценку, разобьем диагональную матрицу сингулярных значений следующим образом:

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{matrix} \tilde{r} \\ m - \tilde{r} \end{matrix}.$$

Если  $\omega \in \mathbb{R}^{\tilde{r}}$  – единичный вектор, удовлетворяющий условию  $\|B_1\omega\|_2 = \sigma_{\tilde{r}}(B_1)$ , то

$$\begin{aligned} \sigma_{\tilde{r}}(B_1)^2 &= \|B_1\omega\|_2^2 = \left\| U\Sigma V^T P \begin{bmatrix} \omega \\ 0 \end{bmatrix} \right\|_2^2 \\ &= \|\Sigma_1 \tilde{V}_{11}^T \omega\|_2^2 + \|\Sigma_2 \tilde{V}_{12}^T \omega\|_2^2. \end{aligned}$$

Теорема верна, потому что  $\|\Sigma_1 \tilde{V}_{11}^T \omega\|_2 \geq \sigma_{\tilde{r}}(A)/\|\tilde{V}_{11}^{-1}\|_2$ .  $\square$ .

Этот результат предполагает, что, стремясь получить достаточно независимое подмножество столбцов, мы выбираем матрицу перестановок  $P$ , такую, что результирующая подматрица  $\tilde{V}_{11}$  является предельно хорошо обусловленной. Эвристическое решение этой задачи может быть получено вычислением QR-разложения со столбцовым выбором матрицы  $[V_{11}^T V_{21}^T]$ , где

$$V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{matrix} \tilde{r} \\ n - \tilde{r} \end{matrix}$$

является разбиением матрицы  $V$  из (12.2.1). В частности, если применить QR со столбцовым выбором (алгоритм 5.4.1) для вычисления

$$Q^T [V_{11}^T V_{21}^T] P = [R_{11} \ R_{12}],$$

где матрица  $Q$  ортогональная,  $P$  перестановочная, а  $R_{11}$  верхняя треугольная, то из

(12.2.2) следует, что

$$\begin{bmatrix} \tilde{V}_{11} \\ \tilde{V}_{21} \end{bmatrix} = P^T \begin{bmatrix} V_{11} \\ V_{21} \end{bmatrix} = \begin{bmatrix} R_{11}^T & Q^T \\ R_{12}^T & Q^T \end{bmatrix}.$$

Отметим, что матрица  $R_{11}$  невырождена и  $\|\tilde{V}_{11}^{-1}\|_2 = \|R_{11}^{-1}\|_2$ . Эвристически, столбцовый выбор позволяет получить хорошо обусловленную матрицу  $R_{11}$ , и поэтому в целом процесс позволяет получить хорошо обусловленную матрицу  $\tilde{V}_{11}$ . Поэтому имеем

**Алгоритм 12.2.1.** Для заданных матрицы  $A \in \mathbb{R}^{m \times n}$  и вектора  $b \in \mathbb{R}^m$  следующий алгоритм вычисляет матрицу перестановок  $P$ , оценку ранга  $\tilde{r}$  и вектор  $z \in \mathbb{R}^{\tilde{r}}$ , такие, что первые  $\tilde{r}$  столбцов матрицы  $B = AP$  независимы и величина  $\|B(:, 1:\tilde{r})z - b\|_2$  является минимальной.

Вычислить SVD  $U^T A V = \text{diag}(\sigma_1, \dots, \sigma_n)$  и сохранить  $V$ .

Определить  $\tilde{r} \leq \text{rank}(A)$ .

Применить метод QR со столбцовым выбором:  $Q^T V(:, 1:\tilde{r})^T P = [R_{11} \ R_{12}]$  и положить  $AP = [B_1 \ B_2]$ , где  $B_1 \in \mathbb{R}^{m \times \tilde{r}}$  и  $B_2 \in \mathbb{R}^{m \times (n-\tilde{r})}$ .

Определить вектор  $z \in \mathbb{R}^{\tilde{r}}$ , такой, что  $\|b - B_1 z\|_2 = \min$ .

**Пример 12.2.1.** Пусть

$$A = \begin{bmatrix} 3 & 4 & 1 \\ 7 & 4 & -3 \\ 2 & 5 & 3 \\ -1 & 4 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

$\text{rank}(A) = 2$  и  $x_{LS} = [0.0815, 0.1545, 0.0730]^T$ . Если применяется алгоритм 12.2.1, то мы находим  $P [e_2 e_1 e_3]$  и решение  $x = [0.0845, 0.2275, 0.0000]^T$ . Заметим:  $\|b - Ax_{LS}\| \approx \|b - Ax\|_2 = 0.1966$ .

### 12.2.3. Еще раз о противоречии между независимостью столбцов и невязкой

Вернемся к обсуждению противоречия между независимостью столбцов и нормой невязки. В частности, чтобы оценить описанный выше метод выбора подмножества, необходимо оценить невязку, которую задает вектор  $y$ :  $r_y = b - Ay = b - B_1 z = (I - B_1 B_1^+)b$ . Здесь матрица  $B_1 = B(:, 1:\tilde{r})$ , где  $B = AP$ . С этой целью проведем сравнение невязок  $r_y$  и  $r_{x_{\tilde{r}}} = b - Ax_{\tilde{r}}$ , поскольку  $A$  является  $\tilde{r}$ -ранговой матрицей и вектор  $x_{\tilde{r}}$  разрешает задачу, приближенную к  $\tilde{r}$ -ранговой LS-задаче, а именно,  $\min \|Ax - b\|_2$ .

**Теорема 12.2.2.** Если невязки  $r_y$  и  $r_{x_{\tilde{r}}}$  определены как описано выше, и если  $\tilde{V}_{11}$  является ведущей  $r \times r$  главной подматрицей матрицы  $P^T V$ , то

$$\|r_{x_{\tilde{r}}} - r_y\|_2 \leq \frac{\sigma_{\tilde{r}+1}(A)}{\sigma_{\tilde{r}}(A)} \|\tilde{V}_{11}^{-1}\|_2 \|b\|_2.$$

*Доказательство.* Отметим, что  $r_{x_{\tilde{r}}} = (I - U_1 U_1^T)b$  и  $r_y = (I - Q_1 Q_1^T)b$ , где

$$U = \begin{bmatrix} U_1 & U_2 \\ \tilde{r} & m-\tilde{r} \end{bmatrix}$$

является разбиением матрицы  $U$  из (12.2.1) и где  $Q_1 = B_1 (B_1^T B_1)^{-1/2}$ . Из § 2.6 мы

получаем

$$\|r_{x\tilde{r}} - r_y\|_2 \leq \|U_1 U_1^T - Q_1 Q_1^T\|_2 \|b\|_2 = \|U_2^T Q_1\|_2 \|b\|_2,$$

в то же время из теоремы 12.2.1 можно сделать вывод, что

$$\|U_2^T Q_1\|_2 \leq \|U_2^T B_1\|_2 \|(B_1^T B_1)^{-1/2}\|_2 \leq \sigma_{\tilde{r}+1}(A) \frac{1}{\sigma_{\tilde{r}}(B_1)} \leq \frac{\sigma_{\tilde{r}+1}(A)}{\sigma_{\tilde{r}}(A)} \|\tilde{V}_{11}^{-1}\|_2$$

Замечая, что  $\|r_{x\tilde{r}} - r_y\|_2 = \|B_1 y - \sum_{i=1}^r (u_i^T b) u_i\|_2$ , мы видим из теоремы 12.2.2 насколько хорошо вектор  $B_1 y$  может приблизить «устойчивую» компоненту вектора  $b$ , т. е.  $U_1^T b$ . Попытка аппроксимировать  $U_2^T b$  может привести к решению с большой нормой. Кроме того, теорема утверждает, что если  $\sigma_{\tilde{r}+1}(A) \ll \sigma_{\tilde{r}}(A)$ , то умеренно независимое подмножество столбцов порождает очень близкую невязку. С другой стороны, если не существует хорошей разделенности между сингулярными значениями, то определение  $\tilde{r}$  затрудняется, и усложняется вся задача выбора подмножества.

### Задачи

**12.2.1.** Предположим, что матрица  $A \in \mathbb{R}^{m \times n}$  и что  $\|u^T A\|_2 = \sigma$ , где  $u^T u = 1$ . Показать, что если  $u^T(Ax - b) = 0$  для  $x \in \mathbb{R}^n$  и  $b \in \mathbb{R}^m$ , то  $\|x\|_2 \geq |u^T b|/\sigma$ .

**12.2.2.** Показать, что если матрица  $B_1 \in \mathbb{R}^{m \times k}$  содержит  $k$  столбцов матрицы  $A \in \mathbb{R}^{m \times n}$ , то  $\sigma_k(B_1) \leq \sigma_k(A)$ .

**12.2.3.** Из уравнения (12.2.2) видно, что матрица

$$P^T V = \begin{bmatrix} \tilde{V}_{11} & \tilde{V}_{12} \\ \tilde{V}_{21} & \tilde{V}_{22} \end{bmatrix} \begin{matrix} \tilde{r} \\ n - \tilde{r} \\ \tilde{r} \\ n - \tilde{r} \end{matrix}$$

является ортогональной. Поэтому  $\|\tilde{V}_{11}^{-1}\| = \|\tilde{V}_{22}^{-1}\|_2$  согласно CS-разложению (теорема 2.4.1). Показать, как можно вычислить  $P$ , применяя QR-алгоритм со столбцовым выбором главного элемента к матрице  $[\tilde{V}_{22}^T \quad \tilde{V}_{12}^T]$ . (Для  $\tilde{r} > \frac{n}{2}$  эта процедура будет более экономичной, чем способ, рассмотренный в тексте.) Учтите это замечание в алгоритме 12.2.1.

### Замечания и литература к § 12.2

Содержание этого раздела основано на работе

Golub G.H., Klema V., and Stewart G.W. (1976). “Rank Degeneracy and Least Squares Problems”, Technical Report TR-456, Department of Computer Science, University of Maryland, College Park, MD.

Процедура выбора подмножества, основанная на соответствующем методе для общей задачи наименьших квадратов из § 12.3, приводится в работе

Van Huffel S. and Vandewalle J. (1987). ‘Subset Selection Using the Total Least Squares Approach in Collinearity Problems with Errors in the Variables’, Lin. Alg. and Its Applic. 88/89, 695–714.

Литература по задаче выбора подмножества обширна, и мы отсылаем читателя к статье

Hotelling H. (1957). “The Relations of the Newer Multivariate Statistical Methods to Factor Analysis”, Brit. J. Stat. Psych. 10, 69–79.

### 12.3. Общая задача наименьших квадратов

Задача минимизации  $\|D(Ax - b)\|_2$ , где матрица  $A \in \mathbb{R}^{m \times n}$ , а матрица  $D = \text{diag}(d_1, \dots, d_m)$  является невырожденной, может быть переформулирована следующим образом:

$$\min_{\substack{b+r \in \text{range}(A)}} \|Dr\|_2, \quad r \in \mathbb{R}^m. \quad (12.3.1)$$

В этой задаче по умолчанию существует предположение, что ошибки заключены в векторе «наблюдений»  $b$ . Когда ошибка также присутствует в «данных» матрицы  $A$ , то более естественно рассмотреть задачу

$$\min_{\substack{b+r \in \text{range}(A+E)}} \|D[Er]T\|_F, \quad E \in \mathbb{R}^{m \times n}, \quad r \in \mathbb{R}^m, \quad (12.3.2)$$

где матрицы  $D = \text{diag}(d_1, \dots, d_m)$  и  $T = \text{diag}(t_1, \dots, t_{n+1})$  невырожденны. Эта задача, рассмотренная в работе Golub и Van Loan (1980), была названа *общей задачей наименьших квадратов* (TLS).

Если минимизатор  $[E_0r_0]$  может быть найден для задачи (12.3.2), то любой вектор  $x$ , удовлетворяющий системе  $(A + E_0)x = b + r_0$ , называется TLS-решением. Однако следует понимать, что задача (12.3.2) может не иметь совместного решения. Например, если

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad D = I_3, \quad T = I_3, \quad E_\epsilon = \begin{bmatrix} 0 & 0 \\ 0 & \epsilon \\ 0 & \epsilon \end{bmatrix},$$

то для всех  $\epsilon > 0$   $b \in \text{range}(A + E_\epsilon)$ . Тем не менее не существует наименьшего значения  $\|[Er]\|_F$ , для которого  $b + r \in \text{range}(A + E)$ .

Обобщение задачи (12.3.2) получается, если мы допускаем наличие многих правых частей. В частности, если матрица  $B \in \mathbb{R}^{m \times k}$ , то мы имеем задачу

$$\min_{\substack{\text{range}(B+R) \subseteq \text{range}(A+E)}} \|D[ER]T\|_F, \quad (12.3.3)$$

где  $E \in \mathbb{R}^{m \times n}$  и  $R \in \mathbb{R}^{m \times k}$ , а матрицы  $D = \text{diag}(d_1, \dots, d_m)$  и  $T = \text{diag}(t_1, \dots, t_{n+k})$  невырожденны. Если  $[E_0R_0]$  разрешает задачу (12.3.3), то любой  $X \in \mathbb{R}^{m \times k}$ , удовлетворяющий системе  $(A + E_0)X = (B + R_0)$ , называется TLS-решением для задачи (12.3.3). В этом разделе мы рассмотрим некоторые из математических свойств общей задачи наименьших квадратов и покажем, как она может быть решена с использованием SVD.

#### 12.3.1. Математическая основа

Следующая теорема дает условия существования и единственности TLS-решения в задаче со многими правыми частями.

**Теорема 12.3.1.** Пусть  $A$ ,  $B$ ,  $D$  и  $T$  – это матрицы, определенные выше, и пусть  $m \geq n+k$ . Предположим, что матрица

$$C = D[AB]T = \begin{bmatrix} C_1 & C_2 \\ n & k \end{bmatrix}$$

имеет SVD вида  $U^T C V = \text{diag}(\sigma_1, \dots, \sigma_{n+k}) = \Sigma$ , где матрицы  $U$ ,  $V$  и  $\Sigma$  разбиты

следующим образом

$$U = \begin{bmatrix} U_1 & U_2 \\ n & k \end{bmatrix}, \quad V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} \begin{matrix} n \\ k \end{matrix},$$

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{matrix} n \\ k \end{matrix}.$$

Если  $\sigma_n(C_1) > \sigma_{n+1}(C)$ , то матрица  $[E_0 R_0]$ , заданная соотношением

$$D [E_0 R_0] T = -U_2 \Sigma_2 [V_{12}^T V_{22}^T], \quad (12.3.4)$$

разрешает задачу (12.3.3). Если  $T_1 = \text{diag}(t_1, \dots, t_n)$  и  $T_2 = \text{diag}(t_{n+1}, \dots, t_{n+k})$ , то матрица

$$X_{\text{TLS}} = -T_1 V_{12} V_{22}^{-1} T_2^{-1}$$

существует и является единственным решением системы  $(A + E_0) X = B + R_0$ .

**Доказательство.** Сначала установим два результата, вытекающие из предположения  $\sigma_n(C_1) > \sigma_{n+1}(C)$ . Из уравнения  $CV = U\Sigma$  имеем  $C_1 V_{12} + C_2 V_{22} = U_2 \Sigma_2$ . Мы хотим показать, что матрица  $V_{22}$  невырождена. Пусть  $V_{22}x = 0$  для некоторого единичного в 2-норме вектора  $x$ . Из соотношения  $V_{12}^T V_{12} + V_{22}^T V_{22} = I$  следует, что  $\|V_{12}x\|_2 = 1$ . Но тогда

$$\sigma_{n+1}(C) \geq \|U_2 \Sigma_2 x\|_2 = \|C_1 V_{12} x\|_2 \geq \sigma_n(C_1),$$

получаем противоречие. Следовательно, подматрица  $V_{22}$  является невырожденной.

Другой факт, следующий из предположения  $\sigma_n(C_1) > \sigma_{n+1}(C)$ , касается строгой разделенности  $\sigma_n(C)$  и  $\sigma_{n+1}(C)$ . Из следствия 8.3.3 имеем  $\sigma_n(C) \geq \sigma_n(C_1)$ , и поэтому  $\sigma_n(C) \geq \sigma_n(C_1) > \sigma_{n+1}(C)$ .

Теперь мы можем доказать теорему. Если  $\text{range}(B + R) \subset \text{range}(A + E)$ , то существует  $X(n \times k)$ , такой, что  $(A + E)X = B + R$ , т. е.

$$\{D[AB]T + D[ER]T\}T^{-1} \begin{bmatrix} X \\ -I_k \end{bmatrix} = 0. \quad (12.3.5)$$

Поэтому матрица в фигурных скобках имеет, самое большое, ранг  $n$ . Следуя доказательству теоремы 2.5.2, можно показать, что

$$\|D[ER]T\|_F \geq \sum_{i=n+1}^{n+k} \sigma_i(C)^2$$

и что нижняя оценка достигается на множестве  $[ER] = [E_0 R_0]$ . Неравенство  $\sigma_n(C) > \sigma_{n+1}(C)$  гарантирует, что  $[E_0 R_0]$  является единственным минимизатором. Ядро матрицы

$$\{D[AB]T + D[E_0 R_0]T\} = U_1 \Sigma_1 [V_{11}^T V_{21}^T]$$

задается множеством  $\begin{bmatrix} V_{12} \\ V_{22} \end{bmatrix}$ . Поэтому из (12.3.5) имеем

$$T^{-1} \begin{bmatrix} X \\ -I_k \end{bmatrix} = \begin{bmatrix} V_{12} \\ V_{22} \end{bmatrix} S$$

для некоторой  $k \times k$  матрицы  $S$ . Из уравнений  $T_1^{-1}X = V_{12}S$  и  $-T_2^{-1} = V_{22}S$  видно, что  $S = -V_{22}^{-1}T_2^{-1}$ . Поэтому мы должны получить

$$X = T_1 V_{12} S = -T_1 V_{12} V_{22}^{-1} T_2^{-1} = X_{\text{TLS}}.$$

Если  $\sigma_n(C) = \sigma_{n+1}(C)$ , то TLS-задача может иметь решение, хотя оно может и не быть единственным. В этом случае, желательно выбрать решение «минимальной нормы». С этой целью рассмотрим  $\tau$ -норму, определенную на  $\mathbf{R}^{n \times k}$  следующим образом:  $\|Z\|_\tau = \|T_1^{-1}ZT_2\|_2$ . Если  $X$  задан соотношением (12.3.5), то из теоремы 2.4.1 имеем

$$\|X\|_\tau^2 = \|V_{12}V_{22}^{-1}\|_2^2 = \frac{1 - \sigma_k(V_{22})^2}{\sigma_k(V_{22})^2}.$$

Это предполагает такой выбор матрицы  $V$  в теореме 12.3.1, чтобы  $\sigma_k(V_{22})$  было максимальным.

### 12.3.2. Вычисления для случая $k = 1$

Покажем для важного случая  $k = 1$  как максимизировать  $V_{22}$ . Пусть сингулярные значения матрицы  $C$  удовлетворяют условию  $\sigma_{n-p} > \sigma_{n-p+1} = \dots = \sigma_{n+1}$ , и пусть  $V = [v_1, \dots, v_{n+1}]$  – столбцовое разбиение матрицы  $V$ . Если  $\tilde{Q}$  является матрицей Хаусхолдера, такой, что

$$V(:, n+1-p:n+1)\tilde{Q} = \begin{bmatrix} W & z \\ 0 & a \end{bmatrix},$$

$$p \quad 1$$

то вектор  $\begin{bmatrix} z \\ a \end{bmatrix}$  имеет наибольшую  $(n+1)$ -ю компоненту для всех векторов из  $\text{span}\{v_{n+1-p}, \dots, v_{n+1}\}$ . Если  $a = 0$ , то задача TLS не имеет решения. В противном случае  $x_{\text{TLS}} = -T_1 z / (t_{n+1} a)$ . Более того,

$$\begin{bmatrix} I_{n-1} & 0 \\ 0 & Q \end{bmatrix} U^T (D[Ab]T) V \begin{bmatrix} I_{n-p} & 0 \\ 0 & Q \end{bmatrix} = \Sigma,$$

и поэтому

$$D[E_0 r_0]T = -D[Ab]T \begin{bmatrix} z \\ a \end{bmatrix} [z^T a].$$

В целом, мы имеем следующий алгоритм.

**Алгоритм 12.3.1.** Для заданных матрицы  $A \in \mathbf{R}^{m \times n}$  ( $m > n$ ), вектора  $b \in \mathbf{R}^m$  и невырожденных матриц  $D = \text{diag}(d_1, \dots, d_m)$  и  $T = \text{diag}(t_1, \dots, t_{n+1})$  следующий алгоритм вычисляет (если возможно) вектор  $X_{\text{TLS}} \in \mathbf{R}^n$ , такой, что  $(A + E_0)x = (b + r_0)$  и  $\|D[E_0 r_0]T\|_F$  является минимальной.

Вычислить SVD вида  $U^T(D[Ab]T)V = \text{diag}(\sigma_1, \dots, \sigma_{n+1})$

Сохранить  $V$ .

Определить  $p$ , такое, что  $\sigma_1 \geq \dots \geq \sigma_{n-p} > \sigma_{n-p+1} = \dots = \sigma_{n+1}$

Вычислить матрицу Хаусхолдера  $P$ , такую, что если  $\tilde{V} = VP$ , то  $\tilde{V}(n+1, n-p+1:n) = 0$

```

if    $v_{n+1,n+1} \neq 0$ 
  for  $i = 1:n$ 
     $x_i = -t_i \tilde{v}_{i,n+1} / (t_{n+1} \tilde{v}_{n+1,n+1})$ 
  end
end

```

Этот алгоритм требует около  $2mn^2 + 12n^3$  флопов и большая часть из них связана с вычислением SVD.

**Пример 12.3.1.** Задача TLS

$$\min_{(a+e)x=b+r} \| [er] \|_F,$$

где  $a = [1, 2, 3, 4]^T$  и  $b = [2.01, 3.99, 5.80, 8.30]^T$  имеет решение  $x_{\text{TLS}} = [2.0212]$ ,  $e = (-0.0045, -0.0209, -0.1048, -0.0855)^T$  и  $r = (0.0022, 0.0103, 0.0519, -0.4023)^T$ . Отметим: для этих данных  $x_{\text{LS}} = 2.0197$ .

### 12.3.3. Геометрическая интерпретация

Можно показать, что TLS-решение  $x_{\text{TLS}}$  минимизирует функционал

$$\psi(X) = \sum_{i=1}^m d_i^2 \frac{|a_i^T x - b_i|^2}{x^T T_1^{-2} x + t_{n+1}^{-2}},$$

где  $a_i^T$  – это  $i$ -я строка матрицы  $A$ , а  $b_i$  –  $i$ -я компонента вектора  $b$ . Геометрическая интерпретация задачи TLS может быть сделана при помощи этих сведений. Действительно,

$$\frac{|a_i^T x - b_i|^2}{x^T T_1^{-2} x + t_{n+1}^{-2}}$$

является квадратом расстояния от вектора  $\begin{bmatrix} a_i \\ b_i \end{bmatrix} \in \mathbf{R}^{n+1}$  до ближайшей точки подпространства

$$P_x = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} : a \in \mathbf{R}^n, b \in \mathbf{R}, b = x^T a \right\};$$

здесь расстояние  $\mathbf{R}^{n+1}$  измеряется по норме  $\|z\| = \|T_z\|_2$ . Об этом способе выравнивания было много написано. См. Pearson (1901) и Madansky (1959).

### Задачи

**12.3.1.** Рассмотрим задачу TLS (12.3.2) с невырожденными матрицами  $D$  и  $T$ . (а) Показать, что если  $\text{rank}(A) < n$ , то задача (12.3.2) имеет решение в том и только в том случае, если  $b \in \text{range}(A)$ . (б) Показать, что если  $\text{rank}(A) = n$ , то задача (12.3.2) не имеет решения, если  $A^T D^2 b = 0$  и  $|t_{n+1}| \|Db\|_2 \geq \sigma_n(DAT_1)$ , где  $T_1 = \text{diag}(t_1, \dots, t_n)$ .

**12.3.2.** Показать, что если  $C = D[Ab]T = [A_1d]$  и  $\sigma_n(C) > \sigma_{n+1}(C)$ , то TLS-решение удовлетворяет системе  $(A_1^T A_1 - \sigma_{n+1}(C)^2 I)x = A_1^T d$ .

**12.3.3.** Покажите, как можно решить задачу (12.3.2) с дополнительным ограничением, что первые  $p$  столбцов минимизатора  $E$  нулевые.

### Замечания и литература к § 12.3

Этот раздел основывается на работе

Golub G. H. and Van Loan C. F. (1980). "An Analysis of the Total Least Squares Problem", SIAM J. Num. Anal. 17, 883–93.

Наиболее подробное исследование задачи TLS имеется в работе

Van Huffel S. (1987). "Analysis of the Total Least Squares Problem and Its Use in Parameter Estimation", Doctoral Thesis, Department of Electrical Engineering, K. U., Leuven.

Смотри также

Van Huffel S. (1988). "Comments on the Solution of the Nongeneric Total Least Squares Problem", Report ESAT – KUL – 88/3, Department of Electrical Engineering, K. U. Leuven.

Если некоторые столбцы матрицы  $A$  известны точно, то имеет смысл изменить матрицу TLS-возмущения  $E$  так, чтобы аналогичные столбцы в ней были нулевыми. Детали решения задачи TLS с подобным условием рассматриваются в статьях

Demmel J. W. (1987). "The smallest perturbation of a submatrix which lowers the rank and constrained total least squares problems", SIAM J. Numer. Anal. 24, 199–206.

Van Huffel S. and Vandewalle J. (1988). "The Partial Total Least Squares Algorithm", J. Comp. and App. Math. 21, 333–342.

Дополнительная литература, посвященная выравниванию методом наименьших квадратов, когда существуют ошибки в матричных данных, имеется в списке

Cochrane W. G. (1968). "Errors of Measurement in Statistics", Technometrics 10, 637–66.

Gunst R. F., Webster J. T., Mason R. L. (1976). "A Comparison of Least Squares and Latent Root Regression Estimators", Technometrics, 18, 75–83.

Linnik I. (1961). Method of Least Squares and Principles of the Theory of Observations, Pergamon Press, New York.

Madansky A. (1959). "The Fitting of Straight Lines When Both Variables Are Subject to Error", J. Amer. Stat. Assoc. 54, 173–205.

Pearson K. (1901). "On Lines and Planes of Closest Fit to Points in Space", Phil. Mag. 2, 559–72.

Stewart G. W. (1977c). "Sensitivity Coefficients for the Effects of Errors in the Independent Variables in a Linear Regression", Technical report TR-571, Department of Computer Science, University of Maryland, College Park, MD.

Van der Sluis A. and Veltkamp G. W. (1979). "Restoring Rank and Consistency by Orthogonal Projection", Lin. Alg. and Its Applic. 28, 257–78.

## 12.4. Сравнение подпространств при помощи SVD

Иногда необходимо исследовать связь между двумя заданными подпространствами. Насколько они близки? Пересекаются они или нет? Может ли одно быть «поворнутым» в другое? И так далее. В этом разделе мы покажем, как можно ответить на подобные вопросы при помощи сингулярного разложения.

### 12.4.1. Поворот подпространств

Пусть матрица  $A \in \mathbb{R}^{m \times p}$  содержит данные, полученные при обработке некоторого множества экспериментов. Если такое же множество экспериментов провести повторно, то будет получена другая матрица  $B \in \mathbb{R}^{m \times p}$ . В ортогональной задаче Прокруста возможность поворота матрицы  $B$  в матрицу  $A$  устанавливается решением следующей задачи:

$$\text{минимизировать } \|A - BQ\|_F \text{ при условии } Q^T Q = I_p. \quad (12.4.1)$$

Заметим, что если матрица  $Q \in \mathbf{R}^{p \times p}$  является ортогональной, то

$$\|A - BQ\|_F^2 = \text{trace}(A^T A) + \text{trace}(B^T B) - 2 \text{trace}(Q^T B^T A).$$

Поэтому задача (12.4.1) эквивалентна задаче максимизации  $\text{trace}(Q^T B^T A)$ .

Максимизатор  $Q$  может быть найден вычислением SVD матрицы  $B^T A$ . Действительно, если  $U^T (B^T A) V = \Sigma = \text{diag}(\sigma_1, \dots, \sigma_p)$  – это SVD данной матрицы и мы определим ортогональную матрицу  $Z$  следующим образом:  $Z = V^T Q^T U$ ; то

$$\text{trace}(Q^T B^T A) = \text{trace}(Q^T U \Sigma V^T) = \text{trace}(Z \Sigma) = \sum_{i=1}^p z_{ii} \sigma_i \leq \sum_{i=1}^p \sigma_i.$$

Очевидно, что верхняя граница достигается при  $Q = UV^T$  для  $Z = I_p$ . Это дает следующий алгоритм:

**Алгоритм 12.4.1.** Для данных матриц  $A$  и  $B$  из  $\mathbf{R}^{m \times p}$  следующий алгоритм находит ортогональную матрицу  $Q \in \mathbf{R}^{p \times p}$ , такую, что  $\|A - BQ\|_F$  минимальная.

$$C = B^T A$$

Вычислить SVD вида  $U^T C V = \Sigma$ . Сохранить  $U$  и  $V$ .

$$Q = U V^T$$

Найденная матрица  $Q$  является ортогональным полярным множителем матрицы  $B^T A$ . См. задачу 4.2.9.

**Пример 12.4.1.** Если

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 1.2 & 2.1 \\ 2.4 & 4.3 \\ 5.2 & 6.1 \\ 6.8 & 8.1 \end{bmatrix},$$

то матрица

$$Q = \begin{bmatrix} 0.9999 & -0.0126 \\ 0.0126 & 0.9999 \end{bmatrix}$$

минимизирует  $\|A - BQ\|_F$  на множестве всех  $2 \times 2$  ортогональных матриц  $Q$  и значение минимума равно 0.4661.

## 12.4.2. Пересечение ядер

Пусть заданы матрицы  $A \in \mathbf{R}^{m \times n}$  и  $B \in \mathbf{R}^{p \times n}$ ; рассмотрим задачу нахождения ортогонального базиса для  $\text{null}(A) \cap \text{null}(B)$ . Один подход состоит в вычислении ядра матрицы

$$C = \begin{bmatrix} A \\ B \end{bmatrix},$$

поскольку  $Cx = 0 \Leftrightarrow x \in \text{null}(A) \cap \text{null}(B)$ . Однако если предварительно мы докажем следующую теорему, то из нее вытекает более экономичная процедура.

**Теорема 12.4.1.** Пусть матрица  $A \in \mathbf{R}^{m \times n}$ , пусть  $\{z_1, \dots, z_t\}$  будет ортогональным базисом  $\text{null}(A)$ . Определим матрицу  $Z = [z_1, \dots, z_t]$ , и пусть  $\{w_1, \dots, w_q\}$  будет ортогональным базисом  $\text{null}(BZ)$ , где  $B \in \mathbf{R}^{p \times n}$ . Если матрица  $W = [w_1, \dots, w_q]$ , то столбцы матрицы  $ZW$  задают ортогональный базис для  $\text{null}(A) \cap \text{null}(B)$ .

**Доказательство.** Так как  $AZ = 0$  и  $(BZ)W = 0$ , то мы очевидно имеем, что  $\text{range}(ZW) \subset \text{null}(A) \cap \text{null}(B)$ . Теперь предположим, что вектор  $x$  принадлежит

и  $\text{null}(A)$ , и  $\text{null}(B)$ . Отсюда следует, что  $x = Za$  для некоторого  $0 \neq a \in \mathbb{R}^t$ . Но поскольку  $0 = Bx = BZa$ , то мы имеем  $a = Wb$  для некоторого  $b \in \mathbb{R}^q$ . Поэтому  $x = ZWb \in \text{range}(ZW)$ .

Если для вычисления ортогонального базиса в данной теореме используется SVD, то мы получаем следующую процедуру:

**Алгоритм 12.4.2.** Для данных матриц  $A \in \mathbb{R}^{m \times n}$  и  $B \in \mathbb{R}^{p \times n}$  следующий алгоритм вычисляет целое  $s$  и матрицу  $Y = [y_1, \dots, y_s]$ , имеющую ортогональные столбцы, которые задают подпространство  $\text{null}(A) \cap \text{null}(B)$ . Если пересечение тривиально, то  $s = 0$ .

Вычислить SVD вида  $U_A^T A V_A = \text{diag}(\sigma_i)$ . Сохранить  $V_A$  и положить

```
r = rank(A)
if r < n
C = BV_A(:, r + 1:n)
```

Вычислить SVD вида  $U_C^T C V_C = \text{diag}(\gamma_i)$ . Сохранить  $V_C$  и положить

```
q = rank(C)
if q < n - r
    s = n - r - q
    Y = V_A(:, r + 1:n) V_C(:, q + 1:n - r)
else
    s = 0
end
else
    s = 0
end
```

Объем работы, требующийся для этого алгоритма, зависит от относительных размеров  $m$ ,  $n$ ,  $p$  и  $r$ .

Отметим, что практическая реализация данного алгоритма требует специального подхода для решения задачи, когда вычисленное сингулярное значение  $\hat{\sigma}_i$  мало. Использование для этих целей барьера  $\delta$  (т. е.  $\hat{\sigma}_i < \delta \Rightarrow \hat{\sigma}_i = 0$ ) приводит к тому, что столбцы вычисленной матрицы  $\hat{Y}$  «почти» задают общее ядро матриц  $A$  и  $B$  в том смысле, что  $\|A\hat{Y}\|_2 \approx \|B\hat{Y}\|_2 \approx \delta$ .

**Пример 12.4.2.** Если

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 4 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 3 & 0 \end{bmatrix},$$

то  $\text{null}(A) \cap \text{null}(B) = \text{span}\{x\}$ , где  $x = (1 - 2 - 3)^T$ . Применяя алгоритм 12.4.2, находим

$$V_{2A} V_{2C} = \begin{bmatrix} -0.8165 & 0.0000 \\ -0.4082 & 0.7071 \\ 0.4082 & 0.7071 \end{bmatrix} \begin{bmatrix} 0.3273 \\ 0.9449 \end{bmatrix} \approx \begin{bmatrix} 0.2673 \\ 0.5349 \\ 0.8018 \end{bmatrix} \approx 0.2673 \begin{bmatrix} 1 \\ -2 \\ -3 \end{bmatrix}.$$

### 12.4.3. Углы между подпространствами

Пусть  $F$  и  $G$  – подпространства в  $\mathbb{R}^m$ , размерности которых удовлетворяют условию

$$p = \dim(F) \geq \dim(G) = q \geq 1.$$

Главные углы  $\theta_1, \dots, \theta_q \in [0, \pi/2]$  между  $F$  и  $G$  определяются рекурсивно следующим образом:

$$\cos(\theta_k) = \max_{u \in F} \max_{v \in G} u^T v = u^T v_k$$

при условии:

$$\begin{aligned}\|u\| &= \|v\| = 1 \\ u^T u_i &= 0 \quad i = 1 : k - 1 \\ v^T v_i &= 0 \quad i = 1 : k - 1\end{aligned}$$

Отметим, что главные углы удовлетворяют условию  $0 \leq \theta_1 \leq \dots \leq \theta_q \leq \frac{\pi}{2}$ . Векторы  $\{u_1, \dots, u_q\}$  и  $\{v_1, \dots, v_q\}$  называются *главными векторами* между подпространствами  $F$  и  $G$ .

Главные углы и векторы возникают во многих статистических приложениях. Наибольший главный угол связан с понятием расстояния между подпространствами одинаковой размерности, что мы обсуждали в § 2.6.3. Если  $p = q$ , то  $\text{dist}(F, G) = \sqrt{1 - \cos(\theta_p)^2} = \sin(\theta_p)$ .

Если столбцы матриц  $Q_F \in \mathbb{R}^{m \times p}$  и  $Q_G \in \mathbb{R}^{m \times q}$  определяют ортогональные базисы для  $F$  и  $G$  соответственно, то

$$\max_{\substack{u \in F \\ \|u\|_2=1}} \max_{\substack{v \in G \\ \|v\|_2=1}} u^T v = \max_{\substack{y \in \mathbb{R}^p \\ \|y\|_2=1}} \max_{\substack{z \in \mathbb{R}^q \\ \|z\|_2=1}} y^T (Q_F^T Q_G) z.$$

Из минимаксных характеристик собственных значений, полученных в § 8.3.1, следует, что если

$$Y^T (Q_F^T Q_G) Z = \text{diag}(\sigma_1, \dots, \sigma_q)$$

задает SVD матрицы  $Q_F^T Q_G$ , то можно определить  $u_k$ ,  $v_k$  и  $\theta_k$  следующим образом

$$\begin{aligned}[u_1, \dots, u_p] &= Q_F Y, \\ [v_1, \dots, v_q] &= Q_G Z, \\ \cos(\theta_k) &= \sigma_k, \quad k = 1 : q.\end{aligned}$$

Обычно пространства  $F$  и  $G$  задаются как области определения матриц  $A \in \mathbb{R}^{m \times p}$  и  $B \in \mathbb{R}^{m \times q}$ . В этом случае необходимые ортогональные базисы могут быть получены вычислением QR-разложений этих двух матриц.

**Алгоритм 12.4.3.** Для данных матриц  $A \in \mathbb{R}^{m \times p}$  и  $B \in \mathbb{R}^{m \times q}$  ( $p \geq q$ ) с независимыми линейными столбцами следующий алгоритм вычисляет ортогональные матрицы  $U = [u_1, \dots, u_q]$  и  $V = [v_1, \dots, v_q]$  и  $\cos(\theta_1), \dots, \cos(\theta_q)$ , такие, что  $\theta$ -главные углы между  $\text{range}(A)$  и  $\text{range}(B)$ , а  $u_k$  и  $v_k$  – соответствующие главные векторы.

Используя алгоритм 5.2.1, вычислить QR-разложение

$$\begin{aligned}A &= Q_A R_A, \quad Q_A^T Q_A = I_p, \quad R_A \in \mathbb{R}^{p \times p}, \\ B &= Q_B R_B, \quad Q_B^T Q_B = I_q, \quad R_B \in \mathbb{R}^{q \times q},\end{aligned}$$

$$C = Q_A^T Q_B$$

Вычислить SVD вида  $Y^T C Z = \text{diag}(\cos(\theta_k))$

$$Q_A Y(:, 1 : q) = [u_1, \dots, u_q]; \quad Q_B Z = [v_1, \dots, v_q].$$

Этот алгоритм требует около  $4m(q^2 + 2p^2) + 2pq(m + q) + 12q^3$  флопов.

Идея использовать SVD для вычисления главных углов и векторов предложена в работе Bjölek и Golub (1973). В этой статье также рассматривается задача для матриц  $A$  и  $B$  неполного ранга.

#### 12.4.4. Пересечение подпространств

Алгоритм 12.4.3 также может использоваться для вычисления ортогонального базиса для  $\text{range}(A) \cap \text{range}(B)$ , где  $A \in \mathbb{R}^{m \times p}$  и  $B \in \mathbb{R}^{m \times q}$ .

**Теорема 12.4.2.** Пусть  $\{\cos(\theta_k), u_k, v_k\}_{k=1}^q$  определены при помощи алгоритма 12.4.3. Если индекс  $s$  задается условием  $1 = \cos(\theta_1) = \dots = \cos(\theta_s) > \cos(\theta_{s+1})$ , то имеем

$$\text{range}(A) \cap \text{range}(B) = \text{span}\{u_1, \dots, u_s\} = \text{span}\{v_1, \dots, v_s\}.$$

*Доказательство.* Доказательство следует из того факта, что если  $\cos(\theta_k) = 1$ , то  $u_k = v_k$ .

В приближенной арифметике приходится вычислять приближенную кратность единичных косинусов алгоритма 12.4.3.

**Пример 12.4.3.** Если

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \text{и} \quad B = \begin{bmatrix} 1 & 5 \\ 3 & 7 \\ 5 & -1 \end{bmatrix},$$

то косинусы главных углов между  $\text{range}(A)$  и  $\text{range}(B)$  равны 1.000 и 0.856.

#### Задачи

**12.4.1.** Показать, что если  $A$  и  $B$   $m \times p$ -матрицы, причем  $p \leq m$ , то

$$\min_{Q^T Q = I_p} \|A - BQ\|_F^2 = \sum_{i=1}^p (\sigma_i(A)^2 - 2\sigma_i(B^T A) + \sigma_i(B)^2).$$

**12.4.2.** Обобщить алгоритм 12.4.2, так, чтобы он мог вычислить ортогональный базис для  $\text{null}(A_1) \cap \dots \cap \text{null}(A_s)$ .

**12.4.3.** Обобщить алгоритм 12.4.3, чтобы использовать его для случая, когда матрицы  $A$  и  $B$  неполного ранга.

**12.4.4.** Установить связь главных углов и векторов между  $\text{range}(A)$  и  $\text{range}(B)$  с собственными значениями и собственными векторами обобщенной задачи на собственные значения

$$\begin{bmatrix} 0 & A^T B \\ B^T A & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \sigma \begin{bmatrix} A^T A & 0 \\ 0 & B^T B \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix}.$$

#### Замечания и литература к § 12.4

Задача минимизации  $\|A - BQ\|_F$  на множестве всех ортогональных матриц возникает в психометрии. См. работы

Bar-Itzhack L. Y. (1975). "Iterative Optimal Orthogonalization of the Strapdown Matrix", IEEE Trans. Aerospace and Electronic Systems 11, 30-37.

Green B. (1952). "The Orthogonal Approximation of an Oblique Structure in Factor Analysis", Psychometrika 17, 429-440.

Hanson R. J. and Norris M. J. (1981). "Analysis of Measurements Based on the Singular Value Decomposition", SIAM J. Sci. and Stat. Comp. 2, 363-374.

Schonemann P. (1966). "A Generalized Solution of the Orthogonal Procrustes Problem", Psychometrika 31, 1-10.

Если  $B = 1$ , то данная проблема равносильна нахождению ортогональной матрицы, ближайшей к  $A$ . Это эквивалентно задаче полярного разложения (4.2.9). См. работы

Björck A. and Bowie C. (1971). "An Iterative Algorithm for Computing the Best Estimate of an Orthogonal Matrix", SIAM J. Num. Anal. 8, 358–64.

Higham N.J. and Schreiber R.S. (1988). "Fast Polar decomposition of an Arbitrary Matrix", Technical report 88–942, Department of Computer Science, Cornell University, Ithaca, New York.

Если матрица  $A$  достаточно близка к матрице, ортогональной самой себе, то метод Björck и Bowie более эффективен, чем SVD-алгоритм. Было бы интересно расширить их подход на случай, когда  $b$  не является единичной.

Задача минимизации  $\|AX - B\|_F$  при ограничении, что  $X$  – симметричная матрица, исследована в работе

Higham N.J. (1988b). "The Symmetric Procrustes Problem", BIT 28, 133–43.

Использование SVD для решения канонической корреляционной задачи было оригинально предложено в статье

Björck A. and Golub G.H. (1973). "Numerical Methods for Computing Angles Between Linear Subspaces", Math. Comp. 27, 579–94.

## 12.5. Модифицированные задачи на собственные значения

В этом разделе рассматриваются некоторые варианты стандартной задачи на собственные значения. В обсуждении широко используются матричные методы, описанные в этой книге.

### 12.5.1. Стационарные значения квадратичной формы с ограничениями

Пусть матрица  $A \in \mathbb{R}^{n \times n}$  будет симметричной. Градиент  $r(x) = x^T Ax / x^T x$  равен нулю в том и только том случае, если  $x$  является собственным вектором матрицы  $A$ . Поэтому стационарными значениями  $r(x)$  являются собственные значения матрицы  $A$ .

В некоторых приложениях необходимо находить стационарные значения  $r(x)$  при наличии ограничений вида  $C^T x = 0$ , где  $C \in \mathbb{R}^{n \times p}$  и  $n \geq p$ . Пусть

$$Q^T C Z = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} r \\ n-r \\ r-p+r \end{bmatrix}, \quad r = \text{rank}(C),$$

является полным ортогональным разложением матрицы  $C$ . Определим матрицу  $B \in \mathbb{R}^{n \times n}$  следующим образом:

$$Q^T A Q = B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \begin{bmatrix} r \\ n-r \\ r \end{bmatrix}$$

и положим

$$y = Q^T x = \begin{bmatrix} u \\ v \end{bmatrix} \begin{bmatrix} r \\ n-r \end{bmatrix}.$$

Так как условие  $C^T x = 0$  преобразуется к виду  $S^T u = 0$ , то исходная задача переходит в задачу нахождения стационарных значений  $r(y) = y^T B y / y^T y$  при ограничении  $u = 0$ . Но это равносильно просто нахождению стационарных значений (собственных значений) симметричной  $(n-r) \times (n-r)$  матрицы  $B_{22}$ .

### 12.5.2. Обратная задача на собственные значения

Рассмотрим задачу нахождения стационарных значений градиента  $x^T Ax/x^T x$  при ограничении  $c^T x = 0$ , где матрица  $A \in \mathbb{R}^{n \times n}$  симметричная, а вектор  $c \in \mathbb{R}^n$  ненулевой. С учетом сказанного выше легко показать, что необходимые стационарные значения  $\tilde{\lambda}_1, \dots, \tilde{\lambda}_{n-1}$  лежат между собственными значениями  $\lambda_i$  матрицы  $A$ .

$$\lambda_n \leq \tilde{\lambda}_{n-1} \leq \lambda_{n-1} \leq \dots \leq \lambda_2 \leq \tilde{\lambda}_1 \leq \lambda_1.$$

Теперь предположим, что матрица  $A$  имеет различные собственные значения и что нам даны значения  $\tilde{\lambda}_1, \dots, \tilde{\lambda}_{n-1}$ , удовлетворяющие условию

$$\lambda_n < \tilde{\lambda}_{n-1} < \lambda_{n-1} < \dots < \lambda_2 < \tilde{\lambda}_1 < \lambda_1.$$

Наша задача – определить единичный вектор  $c \in \mathbb{R}^n$ , такой, что  $\tilde{\lambda}_i$  являются стационарными значениями для  $x^T Ax$  при условии  $x^T x = 1$  и  $c^T x = 0$ .

Для определения свойств вектора  $c$  используем метод множителей Лагранжа. Приравнивая градиент

$$\phi(x, \lambda, \mu) = x^T Ax - \lambda(x^T x - 1) + 2\mu x^T c$$

к нулю, мы получим важное уравнение  $(A - \lambda I)x = -\mu c$ , из которого вытекает, что  $\lambda = x^T Ax$ , т. е.  $\lambda$  является стационарным значением. Поэтому матрица  $A - \lambda I$  является невырожденной и  $x = -\mu(A - \lambda I)^{-1}c$ . Умножая обе стороны равенства на  $c^T$  и подставляя спектральное разложение  $Q^T A Q = \text{diag}(\lambda_i)$ , получаем

$$0 = \sum_{i=1}^n \frac{d_i^2}{\lambda_i - \lambda},$$

где  $d = Q^T c$ , т. е.

$$p(\lambda) \equiv \sum_{i=1}^n d_i^2 \prod_{\substack{j=1 \\ j \neq i}}^n (\lambda_j - \lambda) = 0.$$

Заметим, что  $1 = \|c\|_2^2 = \|d\|_2^2 = d_1^2 + \dots + d_n^2$  является коэффициентом при  $(-\lambda)^{n-1}$ . Так как  $p(\lambda)$  – полином, имеющий нули  $\tilde{\lambda}_1, \dots, \tilde{\lambda}_{n-1}$ , получаем

$$p(\lambda) = \prod_{j=1}^{n-1} (\tilde{\lambda}_j - \lambda).$$

Из этих двух формул для  $p(\lambda)$  следует, что

$$d_k^2 = \frac{\prod_{j=1}^{n-1} (\tilde{\lambda}_j - \lambda_k)}{\prod_{\substack{j=1 \\ j \neq k}}^n (\lambda_j - \lambda_k)}.$$

Так как для  $d$  возможны два знака, то существует  $2^n$  различных решений этой задачи.

### 12.5.3. Одноранговая модификация задачи на собственные значения

При заданных  $\sigma \in \mathbb{R}$ ,  $u \in \mathbb{R}^n$  и  $D = \text{diag}(d_1, \dots, d_n)$ , удовлетворяющих условию  $d_1 \geq \dots \geq d_n$ , иногда требуется вычислить собственные значения  $\lambda_1 \geq \dots \geq \lambda_n$  матрицы

$$C = D + \sigma u u^T.$$

Мы обсуждали эту задачу в § 8.6 и сейчас высажем несколько дополнительных замечаний. Используя минимаксные характеристики собственных значений, можно показать, что

- (i) if  $\sigma \geq 0$   
 $d_i \leq \lambda_i \leq d_{i-1} \quad i = 2:n$   
 $d_2 \leq \lambda_1 \leq d_1 + \sigma u^T u$
- (ii) if  $\sigma \leq 0$   
 $d_{i+1} \leq \lambda_i \leq d_i \quad i = 1:n-1$   
 $d_n + \sigma u^T u \leq \lambda_n \leq d_n$

Один подход к вычислению  $\lambda(C)$  связан с применением метода Ньютона к функции  $p(\lambda) = \det(D + \sigma uu^T - \lambda I)$ . Можно показать, что если

$$\begin{aligned} r_1(\lambda) &= 1 \\ p_1(\lambda) &= (d_1 - \lambda) + \sigma u_1^2 \\ \text{for } k &= 2:n \\ r_k(\lambda) &= (d_{k-1} - \lambda)r_{k-1}(\lambda) \\ p_k(\lambda) &= (d_k - \lambda)p_{k-1}(\lambda) + \sigma u_k^2 r_k(\lambda) \end{aligned} \tag{12.5.2}$$

end

то  $p_n(\lambda) = p(\lambda)$ . Поэтому вычисление обоих многочленов  $p(\lambda)$  и  $p'(\lambda)$  осуществляется непосредственно. Переплетение свойств может быть использовано для получения хорошего начального значения.

Альтернативный метод для нахождения  $\lambda(C)$  приводит к обобщенной трехдиагональной задаче на собственные значения. Пусть  $P$  – перестановочная матрица, такая, что если  $v = Pu$ , то

$$v_1 = \dots = v_s = 0, \quad 0 < |v_{s+1}| \leq \dots \leq |v_n|.$$

Далее, пусть  $K$  – двухдиагональная матрица

$$K = \begin{bmatrix} 1 & r_1 & & \cdots & 0 \\ & 1 & r_2 & & \vdots \\ & & \ddots & \ddots & \\ \vdots & & & \ddots & r_{n-1} \\ 0 & \cdots & & & 1 \end{bmatrix}, \quad r_i = \begin{cases} 0 & i = 1:s-1 \\ -v_i/v_{i+1} & i = s:n-1 \end{cases}.$$

Заметим, что  $Kv = v_n e_n$  и поэтому уравнение

$$(D + \sigma uu^T)x = \lambda x$$

преобразуется к виду  $(KPDP^T K^T + v_n^2 e_n e_n^T)y = \lambda K K^T y$ , где  $x = PK^T y$ . Обе матрицы  $KPDP^T K^T + \sigma v_n^2 e_n e_n^T$  и  $KK^T$  симметричные и трехдиагональные. Для нахождения требуемых собственных значений может быть применен обобщенный метод бисекций из § 8.4.1.

### 12.5.4. Вторая обратная задача на собственные значения

Рассмотрим задачу нахождения трехдиагональной матрицы

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & & \beta_{n-1} & \alpha_n \end{bmatrix},$$

такой, что матрица  $T$  и ее ведущая  $(n - 1)$ -го порядка главная подматрица  $\tilde{T}$  имеют заданные собственные значения. Другими словами, ищем матрицу  $T$ , заданную условием  $\lambda(T) = \{\lambda_1, \dots, \lambda_n\}$  и  $\lambda(\tilde{T}) = \{\tilde{\lambda}_1, \dots, \tilde{\lambda}_{n-1}\}$ , причем

$$\lambda_1 > \tilde{\lambda}_1 > \lambda_2 > \dots > \lambda_{n-1} > \tilde{\lambda}_{n-1} > \lambda_n.$$

Для выполнения этих соотношений должна существовать ортогональная матрица  $Q$ , такая, что  $Q^T T Q = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ , и такая, что стационарные значения для

$$\frac{x^T T x}{x^T x} \text{ при условии } e_1^T x = 0$$

достигаются на  $\tilde{\lambda}_1, \dots, \tilde{\lambda}_{n-1}$ . Другими словами,  $\tilde{\lambda}_i$  являются стационарными значениями для

$$\frac{y^T \Lambda y}{y^T y} \text{ при условии } d^T y = 0,$$

где  $d = Q^T e_1$ , первый столбец матрицы  $Q$ . С учетом сказанного выше мы знаем, что

$$d_k^2 = \frac{\prod_{j=1}^{n-1} (\tilde{\lambda}_j - \lambda_k)}{\prod_{\substack{j=1 \\ j \neq k}}^{n-1} (\lambda_j - \lambda_k)}, \quad k = 1 : n.$$

Рассмотрим применение метода Ланцша (9.1.3) для матрицы  $A = \Lambda$  и  $q_1 = d$ . Если алгоритм пробегает все  $n$  итераций, мы получаем ортогональную матрицу  $Q$ , такую, что матрица  $Q^T \Lambda Q = T$  является трехдиагональной. Так как первым столбцом матрицы  $Q$  является вектор  $d$ , отсюда следует, что матрица  $T$  удовлетворяет необходимым свойствам.

#### Задачи

**12.5.1.** Пусть матрица  $A \in \mathbb{R}^{m \times n}$ , рассмотрим задачу нахождения стационарных значений для

$$R(x, y) = \frac{y^T A x}{\|y\|_2 \|x\|_2}, \quad y \in \mathbb{R}^m, x \in \mathbb{R}^n,$$

при ограничениях

$$\begin{aligned} C^T x = 0 & \quad C \in \mathbb{R}^{n \times p} & n \geq p, \\ D^T y = 0 & \quad D \in \mathbb{R}^{m \times q} & m \geq q. \end{aligned}$$

Покажите, как решить эту задачу, сначала вычислив полное ортогональное разложение матриц  $C$  и  $D$  и потом вычислив SVD некоторых подматриц преобразованной матрицы  $A$ .

**12.5.2.** Пусть матрица  $A \in \mathbb{R}^{m \times n}$  и матрица  $B \in \mathbb{R}^{p \times n}$ . Допустим, что  $\text{rank}(A) = n$  и  $\text{rank}(B) = p$ . Используя методы этого раздела, покажите, как решить задачу

$$\min_{\substack{Bx=0 \\ \|x\|_2^2 + 1}} \frac{\|b - Ax\|_2^2}{\|x\|_2^2 + 1} = \min_{Bx=0} \frac{\left\| [AB] \begin{bmatrix} x \\ -1 \end{bmatrix} \right\|_2^2}{\left\| \begin{bmatrix} x \\ -1 \end{bmatrix} \right\|_2^2}.$$

Покажите, что это задача TLS с ограничением. Всегда ли она разрешима?

**12.5.3.** Пусть матрица  $A \in \mathbb{R}^{n \times n}$  симметричная и матрица  $B \in \mathbb{R}^{p \times n}$  имеет ранг  $p$ . Пусть  $d \in \mathbb{R}^p$ . Покажите, как решить задачу минимизации  $x^T A x$  с ограничениями типа  $\|x\|_2 = 1$ ,  $Bx = d$ . Укажите, когда решение может не существовать.

### Замечания и литература к § 12.5

Многие из задач, рассмотренных в данном разделе, появились в следующих обзорных статьях:

Boley D. and Golub G. H. (1987). "A Survey of Matrix Inverse Eigenvalue Problems", *Inverse Problems* 3, 595–622.

Golub G. H. (1973). "Some Modified Matrix Eigenvalue Problems", *SIAM review* 15, 318–44.

Литература по задаче стационарных значений включена в список

Forsythe G. E. and Golub G. H. (1965). "On the Stationary Values of a Second-Degree Polynomial on the Unit Sphere", *SIAM J. App. Math.* 13, 1050–68.

Golub G. H. and Underwood R. (1970). "Stationary Values of the Ratio of Quadratic Forms Subject to Linear Constraints", *Z. Angew. math. Phys.* 21, 318–26.

Использование QR-алгоритма для определения квадратурных формул Гаусса обсуждается в работе

Golub G. H. and Welsch J. H. (1969). "Calculation of Gauss Quadrature Rules", *Math. Comp.* 23, 221–30.

Методы типа Ланцоша для различных обратных задач на собственные значения детально рассмотрены в работах

Boley D. L. and Golub G. H. (1978). "The Matrix Inverse Eigenvalue Problem for Periodic Jacobi Matrices", in *Proc. Fourth Symposium on Basic Problems of Numerical Mathematics*, Prague, pp. 63–76.

Boley D. and Golub G. H. (1984). "A Modified Method for Restructuring Periodic Jacobi Matrices", 42, 143–150.

Новые исследования по развитию алгоритмов содержатся в работах

Friedland S., Nocedal J., and Overton M. L. (1987). "The Formulation and Analysis of Numerical Methods for Inverse Eigenvalue Problems", *SIAM J. Numer. Anal.* 24, 634–667.

Gragg W. B. and Harrod W. J. (1984). "The Numerically Stable Reconstruction of Jacobi Matrices from Spectral Data", *Numer. Math.* 44, 317–336.

Kautsky J. and Golub G. H. (1983). "On the Calculation of Jacobi Matrices", *Lin. Alg. and Its Applic.* 52/53, 439–456.

Еще один важный класс обратных задач на собственные значения связан с нахождением диагональной матрицы  $D$ , такой, что матрица  $AD$  (иногда матрица  $A + D$ ) имеет заданные собственные значения. См. работу

Friedland S. (1975). "On Inverse Multiplicative Eigenvalue Problems for Matrices", *Lin. Alg. and Its Applic.* 12, 127–38.

## 12.6. Модификация QR-разложения

Во многих приложениях требуется заново вычислить разложение заданной матрицы  $A \in \mathbb{R}^{m \times n}$  после того, как матрица была незначительно изменена. Например, нам задано QR-разложение матрицы  $A$  и требуется вычислить QR-разложение матрицы  $A$ , которая получается (а) прибавлением произвольной одноранговой матрицы к  $A$ , (б) добавлением строки (столбца) к матрице  $A$ , (в) удалением строки (столбца) из матрицы  $A$ . В данном разделе мы покажем, что в ситуациях, подобных этой, гораздо эффективнее «модифицировать» QR-разложение матрицы  $A$ , чем выполнить его заново.

Перед началом упомянем, что существуют также способы для модификаций разложений  $PA = LU$ ,  $A = GG^T$  и  $A = LDL^T$ . Однако эти разложения могут быть довольно чувствительны к модификациям, потому что требуется выбор ведущего элемента и, кроме того, когда мы изменяем положительно определенную матрицу, то результат может не быть положительно определенным. См. Gill, Golub, Murray и Saunders (1974) и Stewart (1979c). Следуя этим результатам, мы кратко обсудим гиперболические преобразования и их использование в задаче модификации разложения Холецкого после удаления строки.

### 12.6.1. Одноранговые модификации

Пусть мы имеем QR-разложение  $QR = B \in \mathbb{R}^{m \times n}$  и нам надо вычислить разложение вида  $B + uv^T = Q_1 R_1$ , где  $u, v \in \mathbb{R}^n$  заданы. Заметим, что

$$B + uv^T = Q(R + wv^T), \quad (12.6.1)$$

где  $w = Q^T u$ . Предположим, что мы уже вычислили матрицы вращения  $J_{n-1}, \dots, J_2, J_1$ , такие, что

$$J_1^T \dots J_{n-1}^T w = \pm \|w\|_2 e_1.$$

Здесь каждая матрица  $J_k$  выполняет вращение в плоскости  $k$  и  $k+1$ . (Для ознакомления с деталями см. алгоритм 5.3.1.) Если эти же самые матрицы вращения Гивенса применить к матрице  $R$ , то можно показать, что матрица

$$H = J_1^T \dots J_{n-1}^T R \quad (12.6.2)$$

является верхней хессенберговой. Например, в случае  $n = 4$  мы начинаем с матрицей и вектором

$$R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}, \quad w = \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

и потом изменяем их следующим образом:

$$R = J_3^T R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}, \quad w = J_3^T w = \begin{bmatrix} \times \\ \times \\ \times \\ 0 \end{bmatrix},$$

$$R = J_2^T R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}, \quad w = J_2^T w = \begin{bmatrix} \times \\ \times \\ 0 \\ 0 \end{bmatrix},$$

$$H = J_1^T R = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}, \quad w = J_1^T w = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Соответственно матрица

$$(J_1^T \dots J_{n-1}^T)(R + wv^T) = H \pm \|w\|_2 e_1 v^T = H_1 \quad (12.6.3)$$

является также верхней хессенберговой.

В § 5.2.3 было показано, как вычислить QR-разложение верхней хессенберговой матрицы за  $O(n^2)$  флопов. В частности, можно найти матрицы вращения Гивенса  $G_k$ ,  $k = 1:n-1$ , такие, что матрица

$$G_{n-1}^T \dots G_1^T H_1 = R_1 \quad (12.6.4)$$

является верхней треугольной. Комбинируя (12.6.1) и (12.6.4), мы получаем QR-разложение  $B + uv^T = Q_1 R_1$ , где

$$Q_1 = Q J_{n-1} \dots J_1 G_1 \dots G_{n-1}.$$

Тщательная проверка объема работы показывает, что требуется около  $26n^2$  флопов. Вектор  $w = Q^T u$  требует  $2n^2$  флопов. Вычисление  $H$  и объединение матриц  $J_k$  в матрицу  $Q$  требуют  $12n^2$  флопов. Наконец, вычисление  $R_1$  и умножение матриц  $G_k$  на матрицу  $Q$  требуют  $12n^2$  флопов.

Метод легко обобщается на случай прямоугольной матрицы  $B$ . Он также может быть легко обобщен для вычисления QR-разложения матрицы  $B + UV^T$ , где  $\text{rank}(UV^T) = p > 1$ .

## 12.6.2. Добавление или удаление столбца

Допустим, что имеется QR-разложение

$$QR = A = [a_1, \dots, a_n], \quad a_i \in \mathbb{R}^m, \quad (12.6.5)$$

и разбиение верхней треугольной матрицы  $R \in \mathbb{R}^{m \times n}$  выполнено следующим образом:

$$R = \begin{bmatrix} R_{11} & v & R_{13} \\ 0 & r_{kk} & w^T \\ 0 & 0 & R_{33} \end{bmatrix} \begin{matrix} k-1 \\ 1 \\ m-k \\ k-1 & 1 & n-k \end{matrix}.$$

Теперь допустим, что мы хотим вычислить QR-разложение матрицы

$$\tilde{A} = [a_1 \dots a_{k-1} a_{k+1} \dots a_n] \in \mathbb{R}^{m \times (n-1)}.$$

Заметим, что матрица  $\tilde{A}$  такая же, как и матрица  $A$  с убранным  $k$ -м столбцом и при этом матрица

$$Q^T \tilde{A} = \begin{bmatrix} R_{11} & R_{13} \\ 0 & w^T \\ 0 & R_{33} \end{bmatrix} = H$$

является верхней хессенберговой, т. е.

$$H = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad m = 7, n = 6, k = 3.$$

Очевидно, что нежелательные поддиагональные элементы  $h_{k+1,k}, \dots, h_{n,n-1}$  могут быть обнулены последовательностью вращений Гивенса:  $G_{n-1}^T \dots G_k^T H = R_1$ . Здесь матрица  $G_i$  выполняет вращение в плоскости  $i$  и  $i+1$  для  $i = k:n-1$ . Поэтому если  $Q_1 = QG_k \dots G_{n-1}$ , то  $\tilde{A} = Q_1 R_1$  является QR-разложением матрицы  $\tilde{A}$ .

Приведенная выше процедура может быть выполнена за  $O(n^2)$  флопов и очень полезна в некоторых задачах наименьших квадратов. Например, при желании проверить важность  $k$ -го фактора в исходной модели исключением  $k$ -го столбца в соответствующих матричных данных и решением получившейся LS-задачи. Аналогично, бывает полезным умение эффективно вычислять решение задачи LS после добавления столбца к матрице  $A$ . Пусть мы имеем QR-разложение (12.6.5) и хотим вычислить QR-разложение матрицы

$$\tilde{A} = [a_1 \dots a_k z a_{k+1} \dots a_n],$$

где вектор  $z \in \mathbb{R}^m$  задан. Отметим, что если  $w = Q^T z$ , то матрица

$$Q^T A = [Q^T a_1 \dots Q^T a_k w Q^T a_{k+1} \dots Q^T a_n] = \tilde{A}$$

является верхней треугольной, исключая присутствие «клина» в  $k+1$  столбце, например

$$\tilde{A} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 & 0 \end{bmatrix}, \quad m = 7, n = 5, k = 3.$$

Можно определить матрицы вращения Гивенса  $J_{m-1}, \dots, J_{k+1}$ , такие, что

$$J_{k+1}^T \dots J_{m-1}^T w = \begin{bmatrix} w_1 \\ \vdots \\ w_{k+1} \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

где матрица  $J_{k+1}^T \dots J_{m-1}^T \tilde{A} = \tilde{R}$  верхняя треугольная. Проиллюстрируем это на

приведенном выше примере

$$H = J_6^T \tilde{A} = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H = J_5^T H = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$H = J_4^T H = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Модификация требует  $O(mn)$  флопов.

### 12.6.3. Добавление или исключение строки

Пусть у нас имеется QR-разложение  $QR = A \in \mathbb{R}^{m \times n}$ , и мы хотим получить разложение матрицы

$$\tilde{A} = \begin{bmatrix} w^T \\ A \end{bmatrix},$$

где  $w \in \mathbb{R}^n$ . Заметим, что матрица

$$\text{diag}(1, Q^T) \tilde{A} = \begin{bmatrix} w^T \\ R \end{bmatrix} = H$$

является верхней хессенберговой. Поэтому матрицы вращения Гивенса могут быть определены так, что  $J_n^T \dots J_1^T H = R_1$  является верхней треугольной. Отсюда следует, что

$$\tilde{A} = Q_1 R_1$$

является искомым QR-разложением, где  $Q_1 = \text{diag}(1, Q) J_1 \dots J_n$ .

Результат существенно не усложняется, если новая строка добавляется между строками  $k$  и  $k+1$  матрицы  $A$ . Мы просто применим приведенный выше алгоритм с заменой матрицы  $A$  на матрицу  $PA$  и матрицы  $Q$  на  $PQ$ , где

$$P = \begin{bmatrix} 0 & I_{m-k} \\ I_k & 0 \end{bmatrix}.$$

На выходе матрица  $\text{diag}(1, P^T)Q_1$  является искомым ортогональным сомножителем.

И в конце мы рассмотрим, как модифицировать QR-разложение  $QR = A \in \mathbb{R}^{m \times n}$ , когда исключена первая строка матрицы  $A$ . В частности, мы хотим вычислить QR-разложение подматрицы  $A_1$  в матрице

$$A = \begin{bmatrix} z^T \\ A_1 \end{bmatrix} \begin{matrix} 1 \\ m-1 \end{matrix}.$$

(Процедура аналогична, если исключается произвольная строка.) Пусть  $q^T$  будет первой строкой матрицы  $Q$ , и вычисленные матрицы вращения Гивенса таковы, что

$$G_1^T \dots G_{m-1}^T q = \alpha e_1,$$

где  $\alpha = \pm 1$ . Отметим, что матрица

$$H = G_1^T \dots G_{m-1}^T R = \begin{bmatrix} v^T \\ R_1 \end{bmatrix} \begin{matrix} 1 \\ m-1 \end{matrix}$$

является верхней хессенберговой и что

$$QC_{m-1} \dots G_1 = \begin{bmatrix} \alpha & 0 \\ 0 & Q_1 \end{bmatrix},$$

где матрица  $Q_1 \in \mathbb{R}^{(m-1) \times (m-1)}$  ортогональна. Поэтому

$$A = \begin{bmatrix} z^T \\ A_1 \end{bmatrix} = (QG_{m-1} \dots G_1)(G_1^T \dots G_{m-1}^T R) = \begin{bmatrix} \alpha & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} v^T \\ R_1 \end{bmatrix},$$

отсюда мы заключаем, что  $A_1 = Q_1 R_1$  есть искомое QR-разложение.

#### 12.6.4. Методы гиперболического преобразования

Напомним, что  $R$  в разложении  $A = QR$  – это есть множитель Холецкого из разложения  $A^T A = GG^T$ . Поэтому существует тесная связь между тем, что рассмотренными модификациями QR-разложения и аналогичными модификациями разложения Холецкого. Проиллюстрируем это на задаче модификации разложения Холецкого, которая соответствует удалению  $A$ -строки в QR-разложении. В задаче модификации разложения Холецкого после удаления строки, нам дано разложение Холецкого

$$GG^T = A^T A = \begin{bmatrix} z^T \\ A_1 \end{bmatrix}^T \begin{bmatrix} z^T \\ A_1 \end{bmatrix}, \quad (12.6.6)$$

где  $A \in \mathbb{R}^{m \times n}$  при  $m > n$  и  $z \in \mathbb{R}^n$ . Наша цель найти нижний треугольник  $G_1$ , такой, что  $G_1 G_1^T = A_1^T A_1$ . Существует несколько подходов к этой интересной и важной задаче. Мы представим только процедуру модификации, которая основывается на гиперболических преобразованиях, поскольку это дает удобный случай ввести несколько новых идей.

Начнем с определения. Матрица  $H \in \mathbb{R}^{m \times m}$  является псевдоортогональной относительно сигнатурной матрицы  $S = \text{diag}(\pm 1) \in \mathbb{R}^{m \times m}$ , если  $H^T S H = S$ . Теперь из (12.6.6) имеем  $A^T A = A_1^T A_1 + z z^T = GG^T$  и поэтому

$$A_1^T A_1 = A^T A - z z^T = GG^T - z z^T = [Gz] \begin{bmatrix} I_n & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} G^T \\ z^T \end{bmatrix}.$$

Определим сигнатурную матрицу

$$S = \begin{bmatrix} I_n & 0 \\ 0 & -1 \end{bmatrix} \quad (12.6.7)$$

и предположим, что мы можем найти матрицу  $H \in \mathbf{R}^{(n+1) \times (n+1)}$ , такую, что  $H^T S H = S$ , при условии, что матрица

$$H \begin{bmatrix} G^T \\ z^T \end{bmatrix} = \begin{bmatrix} G_1^T \\ 0 \end{bmatrix} \quad (12.6.8)$$

является верхней треугольной. Отсюда следует, что

$$A_1^T A_1 = [Gz] H^T S H \begin{bmatrix} G^T \\ z^T \end{bmatrix} = [G_1 0] S \begin{bmatrix} G_1 \\ 0 \end{bmatrix} = G_1 G_1^T$$

является искомым разложением Холецкого.

Теперь покажем, как построить матрицу гиперболического преобразования  $H$  из (12.6.8), использующую *гиперболические вращения*. Матрица  $2 \times 2$  гиперболического вращения имеет вид

$$H = \begin{bmatrix} \operatorname{ch}(\theta) & -\operatorname{sh}(\theta) \\ -\operatorname{sh}(\theta) & \operatorname{ch}(\theta) \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & c \end{bmatrix}.$$

Заметим, что если  $H \in \mathbf{R}^{2 \times 2}$  является матрицей гиперболического вращения, то  $H^T S H = S$ , где  $S = \operatorname{diag}(-1, 1)$ . Проводя аналогию с нашими исследованиями вращений Гивенса, давайте рассмотрим, как гиперболические вращения могут быть использованы для обнуления. Из соотношения

$$\begin{bmatrix} c & -s \\ -s & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad c^2 - s^2 = 1,$$

получаем уравнение  $c x_2 = s x_1$ . Отметим, что решения для этого уравнения не существует, если  $x_1 = x_2 \neq 0$ ; можно сделать вывод, что гиперболические вращения не так численно устойчивы, как их двойники – вращения Гивенса. Если  $x_1 \neq x_2$ , то возможно вычислить пару  $\operatorname{ch} - \operatorname{sh}$ :

```

if  $x_2 = 0$ 
     $s = 0; c = 1$ 
else
    if  $|x_2| < |x_1|$ 
         $\tau = x_2/x_1; c = 1/\sqrt{1 - \tau^2}; s = \tau t$ 
    elseif  $|x_1| < |x_2|$ 
         $\tau = x_1/x_2; s = 1/\sqrt{1 - \tau^2}; c = s t$ 
    end
end

```

(12.6.9)

Отметим, что норма матрицы гиперболического вращения, полученной при помощи этого алгоритма, настолько велика, насколько  $x_1$  близок к  $x_2$ .

Любая матрица  $H = H(p, n + 1, \theta) \in \mathbf{R}^{(n+1) \times (n+1)}$ , которая совпадает с единичной за исключением элементов  $h_{p,p} = h_{n+1,n+1} = \operatorname{ch}(\theta)$  и  $h_{p,n+1} = h_{n+1,p} = -\operatorname{sh}(\theta)$  удовлетворяет условию  $H^T S H = S$ , где  $S$  – матрица из (12.6.7). Используя алгоритм (12.6.9), мы попытаемся создать гиперболические вращения  $H_k = H(1, k, \theta_k)$  для  $k = 2:n+1$ , такие, что

$$H_n \dots H_1 \begin{bmatrix} G^T \\ z^T \end{bmatrix} = \begin{bmatrix} \tilde{G}^T \\ 0 \end{bmatrix}.$$

Это оказывается возможным, если матрица  $A$  имеет полный столбцовый ранг. Гиперболическое вращение  $H_k$  обнуляет элемент  $(k+1, k)$ . Другими словами, если матрица  $A$  имеет полный столбцовый ранг, тогда можно показать, что каждое обращение к алгоритму (12.6.9) дает пару ch – sh. См. Alexander, Pan и Plemnons (1988).

### Задачи

**12.6.1.** Пусть нам известно QR-разложение матрицы  $A \in \mathbb{R}^{m \times n}$  и требуется минимизировать  $\|(A + Uv^T)x - b\|_2$ , где  $u, b \in \mathbb{R}^m$  и  $v \in \mathbb{R}^n$  заданы. Предложить алгоритм для решения этой задачи, требующий  $O(mn)$  флопов. Предположить, что матрица  $Q$  должна модифицироваться.

**12.6.2.** Пусть нам известно QR-разложение  $QR = A \in \mathbb{R}^{m \times n}$ . Предложить алгоритм для вычисления QR-разложения матрицы  $A$ , полученной вычеркиванием  $k$ -й строки из  $A$ . Ваш алгоритм должен требовать  $O(mn)$  флопов.

**12.6.3.** Пусть матрица  $T \in \mathbb{R}^{n \times n}$  является трехдиагональной и симметричной, и пусть вектор  $v \in \mathbb{R}^n$ . Предложить  $O(n^2)$  метод для вычисления ортогональной матрицы  $Q \in \mathbb{R}^{n \times n}$ , такой, что матрица  $Q^T(T + vv^T)Q = \tilde{T}$  является также трехдиагональной.

**12.6.4.** Пусть

$$A = \begin{bmatrix} c^T \\ B \end{bmatrix}, \quad c \in \mathbb{R}^n, \quad B \in \mathbb{R}^{(m-1) \times n},$$

где  $m > n$ . Используя формулу Sherman–Morrison–Woodbury, показать что

$$\frac{1}{\sigma_{\min}(B)} \leq \frac{1}{\sigma_{\min}(A)} + \frac{\|(A^T A)^{-1} c\|_2^2}{1 - c^T (A^T A)^{-1} c}.$$

**12.6.5.** Каково поведение 2-нормы матрицы гиперболического вращения, полученной алгоритмом (12.6.9) как функции от  $x_1$  и  $x_2$ ?

**12.6.6.** Показать, что гиперболическая редукция из § 12.6.4 не обрывается, если матрица  $A$  имеет полный столбцовый ранг.

### Замечания и литература к § 12.6

Численные аспекты задачи модификации представлены в работе

Gill P. E., Golub G. H., Murray W., and Saunders M. A. (1974). “Methods for Modifying Matrix Factorizations”, Math. Comp. 28, 505–35.

Дополнительная литература по этой теме

Bartels R. H. (1971). “A Stabilization of the Simplex Method”, Numer. Math. 16, 414–434.

Daniel J., Gragg W. B., Kaufman L., and Stewart G. W. (1976). “Reorthogonalization and Stable Algorithms for Updating the Gram–Schmidt QR Factorization”, Math. Comp. 30, 772–95.

Gill P. E., Murray W., and Saunders M. A. (1975). “Methods for Computing and Modifying the LDV Factors of a Matrix”, Math. Comp. 29, 1051–77.

Goldfarb D. (1976). “Factored Variable Metric Methods for Unconstrained Optimization”, Math. Comp. 30, 796–811.

Фортран-программы для модификации QR-разложения и разложения Холецкого включены в Linpack (гл. 10). Устойчивость разложения Холецкого анализируется в работе

Stewart G. W. (1979c). “The Effects of Rounding Error on an Algorithm for Downdating a Cholesky Factorization”, J. Inst. Math. Applic. 23, 203–13.

Гиперболические преобразования аналогичны преобразованиям Гивенса, но основаны на ch

и  $\sinh$  вместо  $\cos$  и  $\sin$ . Использование гиперболических преобразований для модификации разложений описано в статьях

Alexander S. T., Pan C. T., and Plemmons R. J. (1988). "Analysis of a Recursive Least Squares Hyperbolic Rotation Algorithm for Signal Processing", *Lin. Alg. and Its Applic.* 98, 3–40.

Bojanczyk A. W., Brent R. P., Van Dooren P., and de Hoog F. R. (1987). "A Note on Downdating the Cholesky Factorization", *SIAM J. Sci. and Stat. Comp.* 8, 210–221.

Golub G. H. (1969). "Matrix Decompositions and Statistical Computation", in *Statistical Computation*, ed., Milton R. C. and Nelder J. A., Academic Press, New York, pp. 365–97.

Работа по распараллеливанию процедур модификации с пополнением и модификации с удалением только началась. Смотри

Henkel C. S., Heath M. T., and Plemmons R. J. (1988). "Cholesky Downdating on a Hypercube", in Fox G. (1988), 1592–1598.

Роль методов модификации для нелинейных уравнений и в задачах оптимизации рассматривается в работе

Dennis J. E. and Schnabel R. B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

# Предметный указатель

- Алгебраическая кратность (algebraic multiplicity) 305  
Алгоритмы: (algorithms)  
    Аазена (Aassen's) 155  
    барьерный метод Якоби (threshold Jacobi) 403  
    бисекция (bisection) 393  
    блочная циклическая редукция (block cyclic reduction) 162–164  
    блочный Ланцоша (block Lanczos) 438  
    быстрое вращение Гивенса (fast Givens rotation) 210  
    возвведение в квадрат  $\exp(A)$  (squaring) 498, 499  
Гаусса – Зейделя (Gauss – Seidel) 454  
двоичное возвведение в степень (binary powering) 493  
двудиагонализация Хаушольдера (Hausholder bidiagonalization) 218  
диагонального выбора (diagonal pivoting) 156  
Дулилла (Doolittle reduction) 99  
жорданово разложение (Jordan decomposition) 350  
исключение Гаусса (Gauss elimination)  
    блочный (block) 113  
    полный выбор ведущего элемента (complete pivoting) 114  
    с внешним произведением (outer product) 96  
    с частичным выбором: версия с внешним произведением (outer product with pivoting) 109  
    с частичным выбором: gaxpy-версия (gaxpy with pivoting) 111  
    gaxpy-версия LU-разложения (gaxpy LU) 97  
итерации с отношением Релея (Rayleigh quotient iteration) 440  
итерации Якоби (Jacobi iteration) 453  
классический метод Грама – Шмидта (classical Gram – Schmidt) 201  
ланцошева двудиагонализация (Lanczos bidiagonalization) 446, 447  
Левинсона (Levinson) 175  
ленточный метод Холецкого (band Cholesky) 146  
ленточная обратная подстановка (band backward substitution) 143  
ленточная прямая подстановка (band forward substitution) 143  
матричная операция gaxpy (matrix-matrix gaxpy operation) 22  
метод Арнольди (Arnoldi method) 448  
метод Бартельса – Стюарта (Bartels – Stewart method) 486  
метод оценки обусловленности (condition estimator) 124  
метод gaxpy с обратным ходом (backward gaxpy) 458  
неявный симметричный QR-шаг со сдвигом Уилкинсона (implicit symmetric QR step with Wilkinson shift) 380  
нормальные уравнения (normal equations) 207  
обратная подстановка (backward substitution) 88, 89  
ортогонализация Грама – Шмидта (Gram – Schmidt orthogonalization) 202  
ортогональные итерации (orthogonal iterations) 318  
Партлетта – Рида (Partlett – Reid) 151  
параллельная кольцевая факторизация (parallel ring factorization) 277  
пересечение подпространств (intersection of subspaces) 522  
пересечение ядер (null-space intersection) 519  
последовательная верхняя релаксация (sequential over-relaxation SOR) 456  
процедура Ланцоша (Lanczos) 433  
прямая подстановка (forward substitution) 87, 89  
прямое накопление (Householder matrix accumulation) 184  
решение блочно-диагональной системы (block diagonal system solving) 160–161  
решение задачи Прокруста (Procrust) 518  
решение симметричной трехдиагональной

- положительно определенной системы (positive definite tridiagonal system solver) 147
- решение систем Вандермонда (Vandermonde system solving) 169
- симметричный метод SOR (symmetric successive over-relaxation) 459
- скалярное произведение (dot product) 18
- скалярное произведение матриц (matrix-matrix dot) 23, 25
- сопряженные градиенты с предобусловлением (preconditioned conjugate gradients) 472, 473
- стационарные значения квадратичной формы с ограничениями (stationary values with constraints) 523
- степенные итерации (power iterations) 316
- Тренча (Trench) 177
- углы между подпространствами (angles between subspaces) 521
- умножение матриц с использованием внешних произведений (matrix-matrix outer product) 25, 26
- умножение матрицы на вектор (matrix-vector row) 19
- умножение на матрицу Гивенса (Givens rotation times matrix) 188
- умножение на матрицу Хаусхолдера слева (Hausholder reflection times matrix) 184
- умножение треугольных матриц (triangular matrix multiplication) 30
- функция от треугольной матрицы (function of triangular matrix) 486
- Хаусхолдера трехдиагонализация (Hausholder triangularization) 377
- Хаусхолдера QR-разложение с выбором верхнего столбца (Hausholder QR) 216
- хессенбергово-треугольное преобразование (Hessenberg-triangular reduction) 356–358
- Холецкого параллельная реализация (Cholesky parallel) 273, 275, 284, 285
- Холецкого разложение  $A - xB$  (Cholesky decomposition) 420
- Холецкого сопряженных направлений (conjugate gradients) 445, 467
- циклический метод Якоби (cyclic Jacobi) 402
- чебышёвские полуитерации (Chebyshev semi-iterative) 457
- Штрассена (Strassen) 43
- экспонента матрицы (matrix exponentiation) 559
- gaxpy** 22
- LDL<sup>T</sup>-разложение (LDL<sup>T</sup>) 128, 130, 131
- LDM<sup>T</sup>-разложение (LDM<sup>T</sup>) 127, 129
- LU-разложение матрицы Хессенберга (LU Hessenberg) 145
- QR-несимметричный (QR unsymmetric) 340
- QR-симметричный (QR symmetric) 380, 381
- QR-итерации с одинарным сдвигом (single shift QR iteration) 336
- QR-разложение: преобразование Гивенса (QR Givens) 198
- QR-разложение: преобразование Хаусхольдера (QR Hausholder) 196
- QR-шаг Френсиса (Francis QR step) 340
- QZ-шаг (QZ step) 361–362
- S-шаг (S-step Lanczos) 440
- saxpy** 18
- SVD 390
- SVD-процедура Якоби (SVD procedure Jacobi) 408, 409
- SVD-шаг Голуба–Кахана (SVD Golub-Kahan step) 389
- TLS 516
- Анализ ошибок (error analysis)
- обратный (backward) 70
  - прямой (forward) 70
- Аппроксимация матричной функции (approximation of a matrix function) 488
- Арифметика с округлением (rounded arithmetic) 66
- с отбрасыванием разрядов (shopped arithmetic) 66
- База (base) 66
- Базис (basis) 56
- Бисекция (bisection) 393
- Блочная матрица (block matrix) 36, 37
- Блочное диагональное доминирование (block diagonal dominance) 161
- Блочные алгоритмы (block algorithms) 36, 53
- двуихдиагональные системы (bidiagonal systems) 161
- Ланцоса (Lanczos) 438
- повторное использование данных (data re-use) 53
- тридиагональные системы (tridiagonal systems) 159
- циклическая редукция (cyclic reduction) 163, 164
- Якоби (Jacobi) 407, 408
- Быстрое преобразование Гивенса (fast Givens transformation) 192, 193
- Вектор (vector)**
- Гаусса (Gauss) 93
  - Хаусхолдера (Hausholder) 183
- Векторно-конвейерные вычисления (vector pipeline computing) 45
- Векторно-конвейерный компьютер (vector pipeline computer) 45

- Векторные вычисления (vector computing) 46, 47
  - конвейерные (pipelining) 40, 46, 47
  - регистры (registers) 47
  - нормы (vector norms) 59, 60
  - обмены (vector touch) 52
  - обозначения (notation) 18
  - операции (operations) 18
- Векторы Ланцоша 428
  - Шура (Schur vectors) 303
- Взаимные перестановки (interchange permutations) 107
- Взвешивание по столбцам (column weighting) 230
  - строкам (row weighting) 230
- Внедиагональный элемент (off-diagonal elements) 399
- Внешнее произведение (outer product) 17
  - блочное (block) 40
- Вращения Гивенса (Givens rotations) 187, 188, 198, 199
  - Якоби (Jacobi) 400
- Выбор ведущего элемента (pivoting) в методе Азена (Aasen) 155
  - полный (complete) 114
  - симметричный (symmetric matrices) 139, 157
    - столбец (column) 214, 215
    - частичный (partial) 108
  - подмножества (subset selection) 509
- Выборочная ортогонализация (selective orthogonalization) 437
- Вычисление ортонормированного базиса (orthonormal basis computation) 203
  - скалярного произведения с накоплением (dot product accumulation) 69
  - погрешности округления (roundoff errors) 66
- Гауссово исключение (Gaussian elimination) 23
  - ошибки округления (roundoff errors) 102
- Геометрическая кратность (geometric multiplicity) 305
- Гиперболические преобразования (hyperbolic transformations) 532, 533
- Главные векторы и углы между подпространствами (principal angles and vectors) 521
- Гребневая регрессия (ridge regression) 504
- Двоеточие (colon) 21, 32
- Двоичное возведение в степень (binary powering) 493
- Двойная точность (double precision) 69
- Двухдиагонализация (bidiagonalization) 218
  - метод Ланцоша (Lanczos) 446
    - с приведением матрицы к верхнедиагональному виду (upper triangularizing first) 219
- Двухдиагональная матрица (bidiagonal matrix) 30
- Декомпозиция области (domain decomposition) 475, 476
- Детерминант (determinant) 51, 52, 58
  - , вырожденность (singularity) 83
- Диагональная форма (diagonal form) 305
- Динамическое распределение работы (dynamically scheduled algorithms) 257, 261
- Динамический резерв задач (pool of task) 256, 270
- Дифференцирование матрицы (differentiation) 58
- Длина вектора (vector length) 47
- Добавление или исключение строки (row addition or deletion) 531
- Доминирующее собственное значение (dominant eigenvalue) 317
- Доминирующий собственный вектор (dominant eigenvector) 317, 318
- Дополнение Шура (Schur complement) 101
- Дробление вычислений (granularity) 247, 264, 265
  - систолическая модель (systolic model) 266
- Единичная матрица (identity matrix) 57
  - ошибка (unit roundoff) 66
- Единичный шаг выборки (unit stride) 50
- Жорданово представление функции от матрицы (Jordan decomposition of matrix function) 488
  - разложение (decomposition) 305, 306
    - вычисление (computation) 350
  - Жордановы блоки (Jordan blocks) 306
- Задача наименьших квадратов (least square (LS) problem) 221
  - невязка (residual) 206
  - неполного ранга (rank deficient) 221, 222
  - основные решения (basic solutions) 224
  - полного ранга (full rank) 224
  - решение с минимальной нормой (minimum norm solution) 222
  - с ограничениями (constrained least squares) 501, 503
    - -- -- типа равенств (equality constrained LS) 505
    - -- -- чувствительность к возмущениям (perturbation sensitivity) 223
  - Прокруста (Procrust problem) 518, 519

- Юла – Уолкера (Yule – Walker problem) 172
- Закон Амдаля (Amdahl's law) 52
- инерции Сильвестра (Sylvester law of inertia) 374
  - – теорема 374
- Защищенные переменные (protected variables) 258
  
- Иерархическая память (hierarchical memory)** 49
- Инвариантное подпространство (invariant subspace) 300, 305
  - – доминирование (dominating) 318
  - – вектор Шура (Schur vector) 303
  - – возмущения (perturbation) 371
  - – прямая сумма (direct sum of) 305
  - – чувствительность (sensitivity) 371, 372
- Индекс профиля (profile index) 149
- Инерция симметрической матрицы (inertia of symmetric matrix) 374
- Инициализация (initialization) 258
- Интервал значений показателя (exponent range) 66
- Исчерпывание (deflation) 335
  - хессенбергово-треугольной формы (Hessenberg-triangular form) 358
  - QR-алгоритм (QR) 335
- Итерации Гаусса – Зейделя (Gauss – Seidel iterations) 507
  - – предобусловливатель (preconditioner) 477, 478
  - – – при решении уравнения Пуассона (solving Poisson equation) 456
  - с одинарным сдвигом (single shift iterations) 377
  - – отношением Релея (Rayleigh quotient iterations) 395
    - – – QR-алгоритм (QR) 396
    - – – симметрично-определенный пучок (symmetric-definite pencil) 421
- Итерационная матрица (iteration matrix) 455
- Итерационное уточнение (iterative improvement) 232
  - – для метода наименьших квадратов (for LS) 232
  - – линейных систем (for linear systems) 121, 122
  - – с фиксированной точностью (fixed precision) 123
  - – со смешанной точностью (mixed precision) 122
- Итерационные методы (iterative methods) 452
- Итерация Якоби для метода (Jacobi iteration for SVD) 408, 409
  - – – симметричной задачи собственных значений (symmetric eigenproblem) 399
  
- Квадратичная форма (quadratic form) 523
- Квадратный корень матрицы (square root of a matrix) 494
- Классические итерации Якоби для собственных значений (classical Jacobi iterations for eigenvalues) 401
- Кольцевые процедуры (ring algorithms) 273
  - – другие (others) 276
  - – Холецкого (Cholesky) 273, 274
  - – Якоби (Jacobi) 405
- Кольцо (ring) 240
- Комплексная матрица (complex matrix) 27
- Конвейеризация (pipelining) 45
  - сложения (addition) 45, 46
  - гахпу 46
- Конфлюэнтные системы Вандермонда (confluent Vandermonde matrix) 170
- Косинус матрицы (cosine of a matrix) 490
- Кратность алгебраическая (algebraic multiplicity) 305
  - геометрическая (geometric) 305
  - собственного значения (multiplicity of eigenvalues) 305
- Кратные собственные значения (multiple eigenvalues) 305
  - – матричная функция 486
  - – – триангуляции Ланцша 439
- Критический разрез алгоритма (critical section) 257
- Кросс-валидации (cross-validation) 504
- Круговое распределение (wrap mapping) 274
- Крылова матрица (Krylov matrix) 330
  - подпространство (Krylov subspace) 426, 427
  
- Ленточное LU-разложение (band LU factorization) 142, 143
- Ленточные алгоритмы (band algorithms)
  - исключение Гаусса (Gauss elimination) 144
  - ленточный метод Холецкого (band Cholesky) 146
  - решение треугольных систем (triangular system) 142
- LU-разложение матрицы Хессенберга (LU Hessenberg) 144, 145
- Ленточный метод Холецкого (band Cholesky) 146
  - параллельный гахпу (parallel gahpu) 291
  - разложение  $A - xB$  420, 421
  - – с внешним произведением (outer product) 290
- Логарифм матрицы (log of matrix) 490
- Локальная программа (nodepogram) 239
  
- Масштабирование (balancing) 342
- Масштабирование по столбцам (column sca-

- ling) 30, 121
- строкам (row scaling) 30, 120
- Матрица (matrix)**
- блочная (block) 36
- верхняя двухдиагональная (upper bidiagonal) 29
- треугольная (upper triangular) 29
- хессенбергова (upper Hessenberg) 29
- вырожденная (singular) 57
- диагональная (diagonal) 29, 30
- ленточная (band) 29
- невырожденная (nonsingular) 57
- нижняя двухдиагональная (lower bidiagonal) 29
  - треугольная (lower triangular) 29
  - хессенбергова (lower Hessenberg) 29
- нормальная (normal) 303
- плохо обусловленная (ill-conditioned) 82
- преобразования Гаусса (Gauss transformation) 93
- простая (nonderogatory) 331
- симметричная (symmetric) 33
- строго диагонально доминирующая (diagonal dominance) 116
- треугольная (triangular) 30, 91
- умножение (multiplication) 92
- трехдиагональная (tridiagonal) 29
- унитреугольная (unit triangular) 91
- Хаусхолдера (Hausholder) 181, 183
- хорошо обусловленная (well-conditioned) 82
- эрмитова (Hermitian) 36
- Матрична норма (matrix norm)** 61, 62
- согласованность (consistency) 62
- соотношения (relations between) 64
- Фробениуса (Frobenius) 62
- экспонента (exponential matrix) 495, 496
- аппроксимации Паде (Padé approximation) 497
- чувствительность (sensitivity) 495, 496
- Матричные операции (operations with matrices)** 17, 18
- Машинная точность (machine precision) 67
- Метод см. также *Алгоритм*
- Метод Аасена (Aasen's method) 152–156
- Арнольди (Arnoldi) 448
- Грама–Шмидта классический (classical Gram–Schmidt) 202
- модифицированный (modified) 202
- конечных элементов (finite element method) 284
- оценка погрешности (error estimation) 293, 294
- Ланцоша (Lanczos) 445–449
- нормальных уравнений (normal equations) 206
- с диагональным выбором (diagonal pivoting method) 156–158
- сопряженных градиентов (conjugate gradient method) 464–467
- связь с алгоритмом Ланцоша 468
- Холецкого блочный (block) 136
- с внешним произведением (outer product) 135
- Холецкого: *gaxpy*-версия 134
- чебышевских полуитераций (Chebyshev semiaffteration method) 457, 458
- Штрассена (Strassen) 42, 43
- Якоби для линейных систем (Jacobi) 453
- Методы для неопределенных систем (indefinite system methods)** 127
- линейного уравнения для блочных трехдиагональных систем (linear equation method for block tridiagonal systems) 159
  - ленточных систем (band systems) 141, 142
  - положительно определенных систем (positive definite systems) 132
  - симметричных неопределенных систем (symmetric indefinite systems) 150
  - систем Вандермонда (Vandermonde) 166
  - тёплицевых систем (Toeplitz) 171
  - треугольных систем (triangular systems) 86
- Минимальная невязка (minimal residual)** 206
- Многочлен Чебышёва (Chebyshev polynomial)** 206
- Множитель Лагранжа (Lagrange multiplier)** 503
- Модель с распределенной памятью (distributed memory model)** 238, 239
- Модификации Краута и Дулитла (Craut and Doolittle)** 99
- Модификация (update)** 21
- QR-разложения (updating QR factorization) 528
- Мониторы (monitors)** 257, 258
- определение 257
- **cholij** 295
- **block.chol** 296
- **nexti** 257
- **nextij** 270
- **gax** 261
- **prod** 271
  
- Наибольшее и наименьшее сингулярные числа матрицы (largest and smallest singular matrix numbers)** 385
- Наискорейший спуск и сопряженные направления (steepest descent and conjugate gradients)** 461
- Направления спуска (search directions)** 462, 463
- Невязки метода сопряженных направлений**

- (residuals of conjugate gradient method) 465
- выбор подмножества (subset selection) 512, 513
- задачи наименьших квадратов (LS problem) 512, 513
- Недоопределенные системы (underdeterminate systems) 234
- Независимость линейная (linear independence) 56
- Неотрицательно определенные системы (semi-definite systems) 137
- Непрерывность разложения собственных значений (continuity of eigenvalue decompositions) 306
- Неприводимые матрицы Хессенберга (ungeneralized Hessenberg matrices) 329
- Неравенство Коши–Шварца (Cauchy–Schwartz inequality) 60
- Несимметрична проблема собственных значений (unsymmetric eigenproblem) 299
- Несимметричный метод Ланцоса (unsymmetric Lanczos method) 449
- Неявный симметричный QR-шаг со сдвигом Уилкинсона (implicit symmetric QR step with Wilkinson shift) 380
- Норма (norm) 59
- Нормальное уравнение (normal equation) 206
- Нормальность матрицы и обусловленность собственных значений (normality and eigenvalues condition) 310
- Нормальные матрицы (normal matrices) 303
- Нормы (norms)
  - векторные (vector) 59–60
  - матричные (matrix) 61
  - степеней матрицы (powers of matrix) 321
  - Фробениуса (Frobenius) 73–75
- Обобщение сингулярного разложения (generalized singular value decomposition) 502
- Обобщенная задача наименьших квадратов с ограничениями (generalized constrained least squares) 501
- Обобщенный метод наименьших квадратов (generalized least squares) 231
- Обозначение абсолютной величины (absolute value notation) 67
- Обозначения (notation)
  - блочная матрица (block matrix) 37
  - вектор (vector) 17
  - двоеточие (colon) 32
  - матрица (matrix) 17
- Образ матрицы (range of a matrix) 57
- Обратная матрица (inverse of matrix) 57
  - случай Тёплица (Toeplitz case) 175, 176
  - подстановка (back substitution) 88
- ортогональная итерация (inverse orthogonal iteration) 322, 323
- Обратные задачи о собственных значениях (inverse eigenvalue problems) 532, 533
- Обратный метод последовательной верхней релаксации (Backward successive over-relaxation) 458
- Обращение в машинный нуль (underflow) 66
- Обусловленность (condition) 81
  - оценка (estimation) 123
  - прямоугольных матриц (condition of rectangular matrix) 206
- Одноранговая модификация диагональной матрицы (rank-one modification of a diagonal matrix) 413
  - задачи на собственные значения 542, 545
  - QR-разложений (QR factorizations) 528
- Операции над векторами (vector operations) 18
  - – матрицами (matrix operations) 17
- Описание алгоритма (specifying algorithm) 27
- Ортогональная матрица (orthogonal matrix) 73
- Ортогональное дополнение (orthogonal complement) 73
- Ортогональные векторы (orthogonal vectors) 73
- Ортогональный проектор (orthogonal projection) 77
- Ортогональное матричное представление (orthogonal matrix representation) 186
  - – вращение Гивенса (Givens rotation) 189
  - – факторизованное (factorized form) 185
- Ортогональные итерации (orthogonal iterations) 318
- Ортонормированный базис (orthonormal basis) 73
- Ортонормальность (orthonormality) 73
- Основное решение в методе наименьших квадратов (basic solution in LS) 224, 225
- Отбрасывание разрядов (cancellation) 66
- Отделенность матриц (separation of matrices) 313
- Отношение вычислительных затрат к коммуникациям (computations/communications ratio) 246
- Оценка погрешности в степенном методе (error estimation in power method) 317
- Ошибки округления (roundoff errors) 68, 70
- Параллельные вычисления (parallel computations)
  - – асинхронная торонидальная процедура (systolic torus) 269
  - – матричное умножение, кольцевой алгоритм (matrix multiplication) 273

- - общая память динамическая (shared memory dynamic) 286
- - статическая (shared memory static) 286
- - решение треугольной системы (triangular system solving) 285
- - кольцевой алгоритм 285
- - Холецкий, асинхронная сеточная процедура (mesh) 285
- - кольцевой алгоритм 273
- - систолический массив 286
- - Якоби циклический метод (cyclic Jacobi) 403
- - gaxpy 238
- - динамическое разделение памяти 259
- - передача сообщений 249
- - статическое разделение памяти 252
- - систолическая модель 243
- - QR-разложение (QR factorization) 238
- - систолический алгоритм (systolic mesh) 287
- Передача сообщений (message passing) 248, 249
- Переменные условия (condition variables) 258
- Переопределенная система (overdetermined system) 205
- Переполнение (overflow) 66
- Перестановка циклов (loop reordering) 26, 27
- Перестановочные матрицы (permutation matrices) 107
- Пересылки в общей памяти (shared memory traffic) 253, 254
- Персимметрическая матрица (persymmetric matrix) 172
- Поворот пространств (rotation of subspaces) 518
- Погрешность абсолютная (absolute error) 60
  - матричной функции (matrix function) 488, 489
  - относительная (relative) 60
- Подматрица (submatrix) 38
- Подпространство (subspace) 56
  - базис (basis) 56
  - инвариантное (invariant) 300, 301
  - ортогональная проекция (orthogonal projection) 77
  - пересечение (intersection) 522
  - ядер (null-space intersection) 519
  - поворот (rotation) 518
  - понижающее (deflating) 364
  - прямая сумма (direct sum) 56
  - расстояние между (distance) 77
  - размерность (orthogonal projection) 77
  - сингулярное (singular) 385, 386
  - угол между 520, 521
- Полиномиальный предобусловливатель (polynomial preconditioner) 477
- Полное ортогональное пространство (complete orthogonal space) 217
- Полные ортогональные разложения (complete orthogonal decomposition) 217
- Положительно определенные системы (positive definite systems) 132, 133, 443
  - алгоритм Ланцоса (Lanczos) 442
  - итерации Гаусса – Зейделя (Gauss – Seidel iterations) 455
  - несимметрические (unsymmetric) 133
  - свойства (properties) 132, 133
  - симметрические (symmetric) 134
  - $LDL^T$  134
- Полярное разложение (polar decomposition) 140
- Понижающее подпространство (deflating subspace) 364
- Последовательная верхняя релаксация (successive over-relaxation), SOR 456
  - предобусловливатель (preconditioner) 477
  - симметричная задача собственных значений (symmetric eigenproblem) 364
  - SSOR 458
  - симметрические неопределенные системы (symmetric indefinite systems) 150
- Последовательность Штурма (Sturm sequence) 393
- Потеря ортогональности (loss of orthogonality) 437
  - в методе Грама – Шмидта (in Gram–Schmidt) 203
  - Ланцоса (in Lanczos) 434, 435
  - точности (cancellation) 67
- Правило Симпсона (Simpson's rule) 494
- Предобусловливатели (preconditioners) 473
  - неполного разложения Холецкого (incomplete Cholesky) 473
  - неполные блочные (incomplete block) 474
- Преобразование Гаусса (Gauss transformations) 93
  - Гаусса – Жордана (Gauss – Jordan) 101–102
  - Кэли (Cayley transform) 194
  - матриц (transformation of matrices)
  - быстрое Гивенса (fast Givens) 191
  - вращения Гивенса (Givens rotations) 191
  - Гаусса (Gauss) 93
  - гиперболические (hyperbolic) 532
  - отражение Хаусхольдера (Householder reflections) 182
  - подобия (similarity transformation) 300
  - неунитарное (nonunitary) 306
  - определение (definition) 301
  - условия (conditions) 306
- Принцип «разделяй и властвуй» (“divide and conquer” principle) 42
- Проблема собственных значений (eigenproblem) 368

- - - несимметричная (unsymmetric) 299
- - - симметричная (symmetric) 368
- Проекции (projections) 77
- Простые матрицы (nonderogatory matrices) 332
- Процессор *id* (*id* processor) 240
- Псевдообратный (pseudoinverse) 223
- Пучок матриц (pencil) 353
  - - диагонализация (diagonalization) 418
  - - симметрично-определенный (symmetric-definite) 418
  - - собственные значения (eigenvalues) 353
  - - эквивалентность (equivalence) 355
- Равномерная загруженность (load balancing) 244, 254, 264
- Разбиение матрицы по столбцам (partitioning a matrix into columns) 20
  - - - строкам (rows) 20
- Разложение (factorization, decomposition)
  - жорданово (Jordan decomposition) 305
  - Кронекера (Kronecker form) 355
  - обобщенное вещественное Шура (generalized real Schur decomposition) 355
  - - симметричное  $2 \times 2$  400
  - трехдиагональное (tridiagonal) 376
  - Хессенберга (Hessenberg reduction) 326, 327
  - - треугольное (Hessenberg triangular) 358
  - Холецкого (Cholesky factorization) 134
  - - неполное (incomplete) 473
  - - Шура (Schur decomposition) 302
  - - вещественное 325, 326
  - - QR 195
- Размерность (dimension) 56
- Ранг матрицы (rank of a matrix) 57
  - - выбор подмножества (subset selection) 512
  - - численное нахождение (determination) 225
  - - численный (numerical) 226
  - - QR-разложение 225
  - - SVD, 75
- Распределенные структуры данных (distributed data structures) 242
- Расщепление (splitting) 454
- Релаксационный параметр (relaxation parameter) 456, 457
- Решение задачи наименьших квадратов (least squares solutions)
  - - - методом быстрых вращений Гивенса (fast Givens) 209, 210
  - - - - Ланцюша (Lanczos method) 447
  - - - модифицированным методом Грама-Шмидта (modified Gram-Schmidt method) 209
  - - - преобразования Хаусхолдера (Householder reduction) 208
- - - SVD 222
- - линейной системы (solving a linear system) 97
- Свойство чередования (interlacing property) 369, 370
- Сдвиг (shift)
  - в симметричном QR-алгоритме 380
  - - QR-итерации 335
  - - QZ-шаге 360
  - - SVD-алгоритме 387
  - Уилкинсона (Wilkinson) 378
- Секулярное уравнение (secular equation) 416, 503
- Сетевая топология (network topology) 239
- Сети процессоров (processor networks) 239
- Сеточная торoidalная процедура (mesh/torus algorithm)
- - - умножение матриц (matrix times matrix) 269
- - - Холецкого алгоритм 285, 286
- Сигнатурная матрица (signature matrix) 533
- Симметрично-определенный пучок (symmetric-definite pencil) 418
- Сингулярное разложение (singular value decomposition (SVD)) 74
  - - алгоритм 383, 387
  - - доказательство (proof) 74
  - - метод Ланцюша 446
  - - - наименьших квадратов с ограничением (constrained least squares) 503
  - - - общая задача (total least squares) 514
  - - обобщенное (generalized) 422
  - - пересечение подпространств (subspace intersection) 522
  - - поворот подпространств (subspace rotation) 518, 519
  - - проекции (projection) 77
  - - псевдообратная матрица (pseudo inverse) 223
  - - ранг матрицы (rank of a matrix) 75
  - - ядро (null-space) 75
- Сингулярные значения (singular values) 74
  - - возмущения (perturbations) 384
  - - матрицы (singular values of a matrix) 74
  - - минимаксная характеристика (minimax characterization) 369, 370
  - - собственные значения (eigenvalues) 383
- Сингулярный вектор левый 74
  - - нуль-пространство (null-space) 75
  - - образ (range) 75
  - - правый 74
- Синус матрицы (sine of a matrix) 490
- Системы Вандермонда (Vandermonde systems) 166, 167
  - общей памяти (shared memory systems) 253, 254

- Скалярное произведение (dot product) 18
- След (trace) 300
- Сложение (addition) 17
- Собственное значение
  - внутреннее (interior) 432
  - дефектное (defective eigenvalue) 305
- Собственные значения (eigenvalues) 305
  - детерминант (determinant) 317
  - доминирующие (dominant) 317
  - комплексно сопряженные (complex conjugate) 325, 326
  - обобщенные (generalized) 354
  - последовательность Штурма (Sturm sequence) 369
  - симметрические матрицы (symmetric matrices) 369
  - след (trace) 300
  - упорядочение в вещественной форме Шура (ordering in Schur form) 345, 346
- Характеристический многочлен (characteristic polynomial) 299, 300
- Собственный вектор (eigenvector) 300
  - возмущение (perturbation) 311
  - дефектная матрица (defective matrix) 306
  - левый (left) 300
  - плохо обусловленная матрица (ill-conditioned matrix) 306
  - правый (right) 300
  - формула Шура (Shur form) 303
  - чувствительность (sensitivity) 311
- Сопряженное транспонирование (conjugate transposition) 27
- Сопряженные направления (conjugate directions) 463
- Сосед (neighbor) 240
- Спектр (spectrum) 300
- Спектральный радиус (spectral radius) 454
- Стационарные значения (stationary values) 369
  - с ограничениями (constrained) 523
- Степени матрицы (matrix powers) 381
- Степенной метод (power method) 316
  - ряд матрицы (power series of a matrix) 490
- Стоимость обмена информацией (communication cost) 246
- Структура данных для блочных матриц (block data structure) 41, 42
  - ленточной матрицы (bandedness data structure) 143
  - при векторной обработке (vector computers) 51, 52
- Сходимость (convergence)
  - итераций Якоби (Jacobi iterations) 344
  - итерационных методов (iterative methods) 454
    - метода бисекций (bisection) 393
    - Гаусса – Зейделя (Gauss – Seidel) 455
    - итераций с отношениями Рэлея (Rayleigh quotient iterations) 396
    - Ланцоса (Lanczos) 430
    - наискорейшего спуска (steepest descent) 461
    - сопряженных градиентов (conjugate gradients) 468, 469
    - Якоби для симметрической задачи собственных значений (Jacobi's method for the symmetric eigenproblem) 401
    - чебышевских полупутерий (Chebyshev semi-iterative) 457
    - обратных итераций (inverse iterations) 344
    - ортогональной итерации (orthogonal iteration) 318
    - симметричного QR-шага со сдвигом (symmetric QR iteration) 380
    - степенных итераций (power method) 316
    - циклического метода Якоби (cyclic Jacobi) 403
    - QR-алгоритма (QR) 335
    - QZ-алгоритма (QZ) 353
    - SVD-алгоритма (SVD) 391
- Теорема (theorem)
  - Бауэра – Файка (Bauer – Fike) 308
  - Виляндта – Хоффмана о собственных значениях (Wielandt – Hoffman) 370
  - Куранта – Фишера о минимаксе (Courant – Fischer minimax theorem) 369
  - о кругах Гершгорина (Gershgorin circle theorem) 307, 308
    - минимакс для собственных значений 369
    - неявном Q (implicit Q theorem) 330, 331
  - Сильвестра об инерции (Sylvester law of inertia) 374
    - сингулярных значений (singular values) 385
- Теория возмущения (perturbation theory) 307
  - для инвариантных подпространств (invariant subspaces) 313
    - для недоопределенных систем (underdetermined systems) 236
    - псевдообратной матрицы (pseudo inverse) 223
      - собственных векторов (eigenvectors) 310, 311
        - значений (eigenvalues) 307
    - инвариантные подпространства симметрических матриц (invariant subspaces of symmetric matrices) 371
    - линейной системы (linear equation problem) 80, 81
    - обобщенное собственное значение (generalized eigenvalues) 355
    - пара сингулярных подпространств (singular subspace pair) 385, 386
    - сингулярные значения (singular values) 384, 385

- собственные значения симметричной матрицы (eigenvalues of symmetric matrix) 370
- Каниеля – Пейджа (Kaniel – Paige theory) 430
- Тёплицева матрица (Toeplitz matrix) 171
- система (Toeplitz system) 171
- Top (torus) 240
- Точность (precision) 66
- Треугольные системы (triangular systems)
  - ленточные (band) 142, 143
  - неквадратные (non-square) 91
  - случай нескольких правых частей (multiple) 89
- Трехдиагональная матрица (tridiagonal matrix) 475
  - обратная (inverse) 475
- Трехдиагональные системы (tridiagonal systems) 146
- Трехдиагонализация (tridiagonalization) 376, 427
  - Ланцоша (Lanczos) 427, 428
  - блочный вариант (block version) 438
  - внутренние собственные значения (interior eigenvalues) 432
  - с выборочной ортогонализацией (selective orthogonalization) 437
  - полной переортогонализацией (complete reorthogonalization) 436
  - s-шаговый (s-step) 440
  - сопряженные градиенты (conjugate gradients) 439
  - степенной метод (power method) 431
  - подпространство Крылова (Krylov subspace) 426
  - Хаусхолдера (Hausholder) 377
  
- Умножение блочных матриц (block matrix multiplication) 39
  - матриц по Штрассену (Strassen multiplication) 43
    - блочная версия (block version) 19
    - матрицы на вектор (matrix – vector) 20
    - матрицу (matrix – matrix) 17, 23
    - число (scalar matrix multiplication) 39
  - Уравнение Сильвестра (Sylvester equation) 347
  - Уравновешивание строчностолбцевое (row – column equilibration) 121
  - Уровень (level) 23, 27
  - операции (of operation) 23
  - Ускорение (speed-up) 246
  - Ускорение Ритца (Ritz acceleration) 396, 397
    - пара и метод Ланцоша (pair and Lanczos method) 437, 438
  - Условия Мура – Пенроуза (Moore – Penrose condition) 223
  - Устойчивость (stability) 498, 499
  
- Флоп (flop) 30, 31
- Формула двойных углов (doubling formulae) 492, 497
- Шермана – Моррисона (Sherman – Morrison formula) 57
- Функции от матриц (matrix functions) 482
  - интегральные (integrating) 494
  - многочлен (polynomial equation) 492, 493
  - Функция треугольной матрицы (function of triangular matrix) 484, 485, 488
  
- Характеристический многочлен (characteristic polynomial) 299
- обобщенная задача собственных значений (generalized eigenproblem) 354
- Хессенбергова форма и итерация Арнольди (Arnoldi process and Hessenberg form) 448
  - преобразования Хаусхолдера (Hausholder reduction) 328
  - QR-итерация 344
  - QR-разбиения 199
  - неприводимость матрицы (unreduced) 330
  - обратные итерации (inverse) 328
  - свойства 329
- Хессенбергово-треугольное преобразование (Hessenberg triangular form reduction) 357, 358
- Хессенберговы системы (Hessenberg systems) 145, 146
- Хранение ленточных матриц (band matrix store) 32, 33
  - по диагоналям 33, 34
  
- Циклическая редукция (cyclic reduction) 162
- Циклический метод Якоби (cyclic Jacobi method) 402, 403
  
- Число Кроуфорда (Crawford number) 420
  - обусловленности (condition number) 81, 82
  - по Шкеелю (Skeel condition number) 85
  - с плавающей точкой (floating point number) 65
- Чистка (sweep) 402
- Чувствительность. См. Теория возмущения
- Чувствительность линейного уравнения (linear equation sensitivity) 83–85
  
- Ширина ленты (bandwidth) 30
  - верхняя (upper bandwidth) 30
  - нижняя (lower bandwidth) 30
- Штрассена алгоритм (Strassen algorithm) 42

- Эквивалентность норм (equivalence of norms)** 63  
**Эффективность (effeciency)** 247
- Ядро (нуль-пространство) матрицы (null-space)** 57  
– пересечение (intersection) 519
- A-норма (A-norm)** 526  
**A-сопряженное направление спуска (A conjugate search direction)** 463
- cols** 20  
**continue** 259  
**CS-разложение (CS decomposition)** 79
- delay** 259
- gauss** 93  
**gaxpy** 22  
– алгоритм (gaxpy algorithms)  
**gaxpy в распределенной памяти (in distributed memory)** 243, 246, 248, 250  
– на общей памяти (in shared memory) 252, 255, 258, 261  
– и внешние произведения (gaxpy vs. outer product) 52  
**get** 254  
**givens** 188  
**glob. init** 253
- house** 183
- LDLT<sup>T</sup>-разложение (LDLT<sup>T</sup>)** 130, 131  
**LDM<sup>T</sup>-разложение (LDM<sup>T</sup>)** 127, 128  
**length** 18  
**loc. init** 241  
**LR-итерация (LR iterations)** 321  
**LU-разложение (LU factorization)**  
– ленточное (band) 141, 142  
– дифференцирование (differentiation) 101  
– детерминант (determinant) 95  
– существование (existence) 96, 97
- Mathlab** 18  
**mesh** 239  
**my. id** 241
- n × n матрица перестановок (n × n permutation matrix)** 172
- p-нормы (p-norms)** 60  
– минимизация (minimisation in) 205  
**put** 254
- quit** 241
- QR-алгоритмы для собственных значений (QR-algorithms for eigenvalues)** 316
- QR-итерации (QR-iterations)**  
– вывод (derivation) 319, 320  
– неявный двойной сдвиг (implicit double shift) 338  
– неявный сдвиг 337  
– одинарный сдвиг (single shift) 336  
– сдвиг и точное собственное значение 336
- QR-разложение (QR-decomposition)** 195  
– вычисление методом блочных отражений (block Householder computation) 197  
– – классическим методом Грама – Шмидта (classical Gram – Schmidt) 201  
– – методом быстрых вращений Гивенса (fast Givens rotations) 200  
– – модифицированным методом Грама – Шмидта (modified) 202  
– квадратные системы (square systems) 235  
– модификация (updating) 528  
– наименьших квадратов (LS) 208  
– неполного ранга с выбором ведущего столбца (rank deficient column pivoting) 215  
– преобразования Хаусхольдера (Householder computation) 196, 197  
– ранг матрицы (rank of matrix) 215, 216  
– свойства 201  
– сеточная систолическая процедура (systolic mesh) 287  
– симметричное (symmetric) 376  
– хессенберговой матрицы 199
- QR-шаг Фрэнсиса (Francis QR-step)** 340
- R-двуихдиагонализация (R-bidiagonalization)** 219
- recv** 214
- saxpy** 18  
**send** 241  
**span** 57
- SVD-шаг Голуба – Кахана (Golub – Kahan SVD step)** 389
- sym.schur2** 401
- WY-представление (WY-representation)** 187
- 2-норма**

# Оглавление

<b>Предисловие редактора перевода . . . . .</b>	5
<b>Предисловие к первому изданию . . . . .</b>	7
<b>Предисловие ко второму изданию . . . . .</b>	11
<b>Глава 1. Умножение матриц . . . . .</b>	16
1.1. Основные алгоритмы и обозначения . . . . .	16
1.2. Учет структуры матриц . . . . .	29
1.3. Блочные матрицы и алгоритмы . . . . .	36
1.4. Некоторые аспекты векторно-конвейерных вычислений . . . . .	45
<b>Глава 2. Матричный анализ . . . . .</b>	56
2.1. Основные сведения из линейной алгебры . . . . .	56
2.2. Векторные нормы . . . . .	59
2.3. Матричные нормы . . . . .	61
2.4. Матричные вычисления с конечной точностью . . . . .	65
2.5. Ортогональность и сингулярное разложение . . . . .	73
2.6. Проекции и CS-разложение . . . . .	77
2.7. Чувствительность квадратных систем к возмущениям . . . . .	80
<b>Глава 3. Линейные системы общего вида . . . . .</b>	87
3.1. Треугольные системы . . . . .	87
3.2. LU-разложение . . . . .	92
3.3. Анализ ошибок округления в методе исключения Гаусса . . . . .	102
3.4. Выбор ведущего элемента . . . . .	106
3.5. Уточнение и оценивание точности . . . . .	119
<b>Глава 4. Линейные системы специального вида . . . . .</b>	127
4.1. Разложения типа $LDM^T$ и $LDL^T$ . . . . .	127
4.2. Положительно определенные системы . . . . .	132
4.3. Ленточные системы . . . . .	141
4.4. Симметричные неопределенные системы . . . . .	150
4.5. Блочные трехдиагональные системы . . . . .	159
4.6. Системы Вандермонда . . . . .	166
4.7. Тёплицевы системы . . . . .	171
<b>Глава 5. Ортогонализация и метод наименьших квадратов . . . . .</b>	181
5.1. Матрицы Хаусхолдера и Гивенса . . . . .	181
5.2. QR-разложение . . . . .	195
5.3. Задача наименьших квадратов: случай полного ранга . . . . .	205
5.4. Другие ортогональные разложения . . . . .	215
5.5. Задача LS неполного ранга . . . . .	221
5.6. Взвешивание и итерационное уточнение . . . . .	230
5.7. Квадратные и недоопределенные системы . . . . .	234

<b>Глава 6. Параллельные матричные вычисления . . . . .</b>	<b>238</b>
6.1. Операция гахру на распределенной памяти . . . . .	238
6.2. Операция гахру на общей памяти . . . . .	252
6.3. Параллельное умножение матриц . . . . .	262
6.4. Кольцевые процедуры разложения . . . . .	273
6.5. Сеточные процедуры разложения . . . . .	281
6.6. Методы разложения на общей памяти . . . . .	290
<b>Глава 7. Несимметричная проблема собственных значений . . . . .</b>	<b>299</b>
7.1. Свойства и разложения . . . . .	300
7.2. Теория возмущения . . . . .	307
7.3. Степенные итерации . . . . .	316
7.4. Хессенбергова форма и вещественная форма Шура . . . . .	324
7.5. Практический QR-алгоритм . . . . .	334
7.6. Методы вычисления инвариантных подпространств . . . . .	343
7.7. QZ-метод для $Ax = \lambda Bx$ . . . . .	353
<b>Глава 8. Симметричная проблема собственных значений . . . . .</b>	<b>368</b>
8.1. Математические основы . . . . .	368
8.2. Симметричный QR-алгоритм . . . . .	376
8.3. Вычисление SVD . . . . .	383
8.4. Некоторые специальные методы . . . . .	392
8.5. Методы Якоби . . . . .	399
8.6. Метод «разделяй и властвуй» . . . . .	412
8.7. Более общие проблемы собственных значений . . . . .	418
<b>Глава 9. Методы Ланцоша . . . . .</b>	<b>426</b>
9.1. Выводы свойства сходимости . . . . .	426
9.2. Практические процедуры Ланцоша . . . . .	433
9.3. Приложения и обобщения . . . . .	442
<b>Глава 10. Итерационные методы для линейных систем . . . . .</b>	<b>452</b>
10.1. Стандартные итерации . . . . .	452
10.2. Метод сопряженных градиентов . . . . .	461
10.3. Сопряженные градиенты с предобусловливанием . . . . .	471
<b>Глава 11. Функции от матриц . . . . .</b>	<b>482</b>
11.1. Спектральные методы . . . . .	482
11.2. Аппроксимационные методы . . . . .	488
11.3. Матричная экспонента . . . . .	495
<b>Глава 12. Специальные разделы . . . . .</b>	<b>501</b>
12.1. Задача наименьших квадратов с ограничениями . . . . .	501
12.2. Выбор подмножества при помощи SVD . . . . .	509
12.3. Общая задача наименьших квадратов . . . . .	514
12.4. Сравнение подпространств при помощи SVD . . . . .	518
12.5. Модифицированные задачи на собственные значения . . . . .	523
12.6. Модификация QR-разложения . . . . .	528
Предметный указатель . . . . .	536

Научное издание

Джин Х. Голуб, Чарльз Ф. Ван Лоун

## МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

Заведующий редакцией академик В.И. Арнольд

Зам. зав. редакцией А.С. Попов

Ведущий редактор А.А. Брянданская

Редакторы Т.А. Денисова, Н.С. Полякова

Художник Ю.С. Урманчиев

Художественный редактор В.И. Шаповалов

Технический редактор М.А. Страшнова

Корректор Н.С. Курлова

Лицензия ЛР № 010174 от 20.05.97 г.

Подписано к печати 05.01.99 г. Формат 70×100<sup>1</sup>/<sub>16</sub>.

Бумага офсетная. Печать офсетная. Гарнитура таймс.

Объем 17,25 бум. л. Усл. печ. л. 44,85. Уч. -изд. л. 42,77.

Изд. № 1/8085. Тираж 5000 экз. Зак. 63. С 002

ИЗДАТЕЛЬСТВО «МИР» Государственного Комитета  
Российской Федерации по печати

129820, Москва, И-110, 1-й Рижский пер., 2

Отпечатано в полном соответствии

с качеством предоставленных диапозитивов

в ОАО «Можайский полиграфический комбинат»

143200, г. Можайск, ул. Мира, 93