de Gruyter Textbook
Deuflhard/Hohmann · Numerical Analysis

Peter Deuflhard
Andreas Hohmann

Numerical Analysis

A First Course in Scientific Computation

Translated from the German by F. A. Potra and F. Schulz

Walter de Gruyter
Berlin · New York 1995

Authors

Peter Deuflhard
Andreas Hohmann

Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10
D-10711 Berlin
Germany

and

Freie Universität Berlin
Institut für Mathematik
Arnimallee 2–6
D-14195 Berlin
Germany

1991 Mathematics Subject Classification: Primary: 65–01
Secondary: 65 Bxx, 65 Cxx, 65 Dxx, 65 Fxx, 65 Gxx

Title of the German original edition: Numerische Mathematik I, Eine algorithmisch orientierte Einführung, 2. Auflage, Walter de Gruyter · Berlin · New York, 1993

With 62 figures and 14 tables.

Printed on acid-free paper which falls within the guidelines of the ANSI to ensure permanence and durability.

Library of Congress Cataloging-in-Publication-Data

Deuflhard, P. (Peter)
[Numerische Mathematik I. English]
Numerical analysis : a first course in scientific computation / Peter Deuflhard, Andreas Hohmann ; translated by F. A. Potra and F. Schulz.
p. cm.
Includes bibliographical references (p. –) and index.
ISBN 3-11-014031-4 (cloth : acid-free). –
ISBN 3-11-013882-4 (pbk. : acid-free)
1. Numerical analysis – Data processing. I. Hohmann, Andreas, 1964– . II. Title.
QA297.D4713 1995
519.4–dc20 94-46993
CIP

Die Deutsche Bibliothek – Cataloging-in-Publication-Data

Numerical analysis / Peter Deuflhard ; Andreas Hohmann. Transl. by F. A. Potra and F. Schulz. – Berlin ; New York : de Gruyter.
(De Gruyter textbook)
Bd. 2 verf. von Peter Deuflhard und Folkmar Bornemann
NE: Deuflhard, Peter; Hohmann, Andreas; Bornemann, Folkmar
1. A first course in scientific computation. – 1995
ISBN 3-11-013882-4 kart.
ISBN 3-11-014031-4 Pp.

Printing in Germany. – Printing: Gerike GmbH, Berlin. – Binding: Lüderitz & Bauer GmbH, Berlin.

Preface

The topic of Numerical Analysis is the development and the understanding of computational methods for the numerical solution of mathematical problems. Such problems typically arise from areas outside of Mathematics — such as Science and Engineering. Therefore Numerical Analysis is directly situated at the confluence of Mathematics, Computer Science, Natural Sciences, and Engineering. A new interdisciplinary field has been evolving rapidly called *Scientific Computing*. Driving force of this evolution is the recent vigorous development of both computers and algorithms, which encouraged the refinement of mathematical models for physical, chemical, technical or biological phenomena to such an extent that their computer simulations are now describing reality to sufficient accuracy. In this process, the complexity of solvable problems has been expanding continuously. New areas of the natural sciences and engineering, which had been considered rather closed until recently, thus opened up. Today, Scientific Computing is contributing to numerous areas of industry (chemistry, electronics, robotics, automotive industry, air and space technology, etc.) as well as to important problems of society (balance of economy and ecology in the use of primary energy, global climate models, spread of epidemics).

The movement of the entire interdisciplinary net of Scientific Computing seizes each of its knots, including Numerical Analysis, of course. Consequently, fundamental changes in the selection of the material and the presentation in lectures and seminars have occurred — with an impact even to introductory courses. The present book takes this development into account. It is understood as an introductory course for students of mathematics and natural sciences, as well as mathematicians and natural scientists working in research and development in industry and universities. Possible readers are assumed to have basic knowledge of undergraduate Linear Algebra and Calculus. More advanced mathematical knowledge, say about differential equations, is not a required prerequisite, since this elementary textbook is intentionally excluding the numerics of differential equations. As a further deliberate restriction, the presentation of topics like interpolation or integration is limited to the one–dimensional case. Nevertheless, many essential concepts of modern Numerical Analysis, which play an important role in

numerical differential equation solving, are treated on the simplest possible model problems.

The aim of the book is to develop algorithmic feeling and thinking. After all, the algorithmic approach is historically one of the roots of todays Mathematics. That is why historical names like Gauss, Newton and Chebyshev are found in numerous places of the subsequent text together with contemporary names. The orientation towards algorithms should by no means be misunderstood. In fact, the most efficient algorithms do require a substantial amount of mathematical theory, which will be developed in the text. As a rule, elementary mathematical arguments are preferred. Wherever meaningful, the reasoning appeals to geometric intuition — which also explains the quite large number of graphical representations. Notions like scalar product and orthogonality are used throughout — in the finite dimensional case as well as in function spaces. In spite of the elementary presentation, the book does contain a significant number of rather recent results, some of which have not been published elsewhere. In addition, some of the more classical results are derived in a way, which significantly differs from more standard derivations.

Last, but not least, the selection of the material expresses the scientific taste of the authors. The first author has taught Numerical Analysis courses since 1978 at different German institutions such as the University of Technology in Munich, the University of Heidelberg, and now the Free University of Berlin. Over the years he has co–influenced the dynamic development of Scientific Computing by his research activities. Needless to say, he has presented his research results in numerous invited talks at international conferences and seminars at renowned university and industry places all over the world. The second author rather recently entered the field of Numerical Analysis, after having graduated in pure mathematics from the University of Bonn. Both authors hope that this combination of a senior and a junior has had a stimulating effect on the presentation in this book. Moreover, it is certainly a clear indication of the old dream of unity of pure and applied mathematics.

Of course, the authors stand on the shoulders of others. In this respect, the first author remembers with gratitude the time, when he was a graduate student of Roland Bulirsch. Numerous ideas of the colleagues Ernst Hairer and Gerhard Wanner (University of Geneva) and intensive discussions with Wolfgang Dahmen (Technical University of Aachen) have influenced our presentation. Cordial thanks go to Folkmar Bornemann for his many stimulating ideas and discussions especially on the formulation of the error analysis in Chapter 2. We also want to thank our colleagues at the Konrad Zuse Center Berlin, in particular Michael Wulkow, Ralf Kornhuber, Ulli Nowak and Karin Gatermann for many suggestions and a constructive atmosphere.

This book is a translation of our German textbook "Numerische Mathematik I (Eine algorithmisch orientierte Einführung)", second edition. Many thanks to our translators, Florian Potra and Friedmar Schulz, and to Erlinda Cadano–Körnig for her excellent work in the final polishing of the English version. May this version be accepted by the Numerical Analysis students equally well as the original German version.

Peter Deuflhard and Andreas Hohmann Berlin, May 1994

Teaching Hints

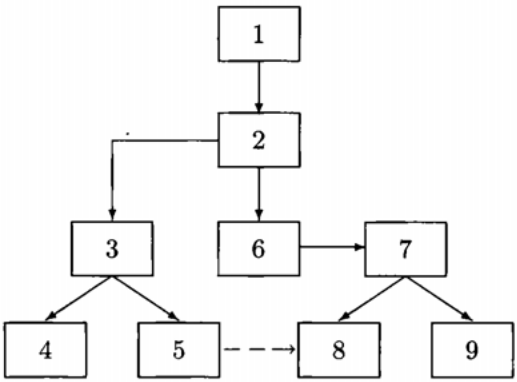
The present textbook addresses students of Mathematics, Computer Science and Science covering typical material for introductory courses in Numerical Analysis with clear emphasis towards Scientific Computing.

We start with Gaussian elimination for linear equations as a classical algorithm and discuss additional devices such as pivoting strategies and iterative refinement. Chapter 2 contains the indispensable error analysis based on the fundamental ideas of Wilkinson. The condition of a problem and the stability of algorithms are presented in a unified framework and exemplified by illustrative cases. Only the linearized theory of error analysis is presented — avoiding, however, the typical "ε–battle". Rather, only differentiation is needed as an analytical tool. As a specialty we derive a stability indicator which allows for a rather simple classification of numerical stability. The theory is then worked out for the case of linear equations, thus supplying a posteriori a deeper understanding of Chapter 1. In Chapter 3 we deal with methods of orthogonalization in connection with linear least squares problems. We introduce the extremely useful calculus of pseudoinverses, which is then immediately applied in Chapter 4. There, we consider iterative methods for systems of nonlinear equations (Newton's method), nonlinear least squares problems (Gauss–Newton method) and parameter–dependent problems (continuation methods) in close mutual connection. Special attention is given to the affine invariant form of the convergence theory and the iterative algorithms. A presentation of the power method (direct and inverse) and the QR–algorithm for symmetric eigenvalue problems follow in Chapter 5. The restriction to the real symmetric case is motivated from the beginning by a condition analysis of the general eigenvalue problem. In this context the singular value decomposition fits perfectly, which is so important in applications.

After the first five rather closely connected chapters the remaining four chapters also comprise a closely connected sequence. The sequence begins in Chapter 6 with an extensive treatment of the theory of three–term recurrence relations, which play a key role in the realization of orthogonal projections in function spaces. Moreover, the significant recent spread of symbolic computing has renewed interest in special functions also within Numerical Analysis.

The condition of three–term recurrences is presented via the discrete Green's function. Numerical algorithms for the computation of special functions are exemplified for spherical harmonics and Bessel functions. In Chapter 7 classical interpolation and approximation in the one–dimensional special case are presented first, followed by non–classical methods like Bézier techniques and splines, which nowadays play a central role in CAD (Computer Aided Design) or CAGD (Computer Aided Geometric Design), i.e. special disciplines of computer graphics. Our presentation in Chapter 8, which treats iterative methods for the solution of large symmetric linear equations, is conveniently based on Chapter 6 (three–term recurrences) and Chapter 7 (min–max property of Chebyshev polynomials). The same is true for the Lanczos algorithm for large symmetric eigenvalue problems. The final Chapter 9 turns out to be a bit longer: it carries the bulk of the task to explain principles of the numerical solution of differential equations by means of the simplest problem type, which here is numerical quadrature. After the historical Newton–Cotes formulas and the Gauss quadrature, we progress towards the classical Romberg quadrature as a first example of an adaptive algorithm, which, however, only adapts the approximation order. The formulation of the quadrature problem as an initial value problem opens the possibility of working out a fully adaptive Romberg quadrature (with order and stepsize control) and at the same time a didactic first step into extrapolation methods, which play a prominent role in the solution of ordinary differential equations. The alternative formulation of the quadrature problem as a boundary value problem is exploited for the derivation of an adaptive multigrid algorithm: in this way we once more present an important class of methods for ordinary and partial differential equation in the simplest possible case.

For a typical university term the contents of the book might be too rich. For a possible partitioning of the presented material into two parts we recommend the closely connected sequences Chapter 1 – 5 and Chapter 6 – 9. Of course, different "teaching paths" can be chosen. For this purpose, we give the following connection diagram:



As can be seen from this diagram, the chapters of the last row (Chapters 4, 5, 8, and 9) can be skipped without spoiling the flow of teaching — according to the personal scientific taste. Chapter 4 could be integrated into a course on "Nonlinear optimization", Chapters 5 and 8 into a course on "Numerical linear algebra" or Chapter 9 into "Numerical solution of differential equations".

At the end of each chapter we added exercises. Beyond these explicit exercises further programming exercises may be selected from the numerous algorithms, which are given informally (usually as pseudocodes) throughout the textbook. All algorithms mentioned in the text are internationally accessible via the electronic library *eLib* of the Konrad Zuse Center. In the interactive mode *eLib* can be reached via:

Datex-P:	+45050331033 (WIN) +2043623331033 (IXI)
INTERNET:	elib.ZIB-berlin.de (130.73.108.11)
login:	elib (no password necessary)

In addition, there is the following e-mail access:

X.400:	S=eLib;OU=sc;P=ZIB-Berlin;A=dbp;C=de
INTERNET:	elib@elib.ZIB-Berlin.de
BITNET:	eLib@sc.ZIB-Berlin.dbp.de
UUCP:	unido!sc.ZIB-Berlin.dbp.de!eLib

Especially for users of Internet there is an "anonymous ftp" access (elib.ZIB-Berlin.de - 130.73.108.11) .

Contents

1 Linear Systems 1
1.1 Solution of Triangular Systems 2
1.2 Gaussian Elimination 4
1.3 Pivoting Strategies and Iterative Refinement 7
1.4 Cholesky's Method for Symmetric Positive Definite Matrices . 15
1.5 Exercises 18

2 Error Analysis 23
2.1 Sources of Errors 23
2.2 Condition of Problems 25
2.2.1 Norm-wise condition analysis 28
2.2.2 Component-wise condition analysis 33
2.3 Stability of Algorithms 37
2.3.1 Stability concepts 38
2.3.2 Forward analysis 40
2.3.3 Backward analysis 45
2.4 Application to Linear Systems 48
2.4.1 A closer look at solvability 48
2.4.2 Backward analysis of Gaussian elimination 50
2.4.3 Assessment of approximate solutions 53
2.5 Exercises 56

3 Linear Least Squares Problems 62
3.1 Least Squares Method of Gauss 62
3.1.1 Formulation of the problem 62
3.1.2 Normal equations 65
3.1.3 Condition 67
3.1.4 Solution of normal equations 70
3.2 Orthogonalization Methods 72
3.2.1 Givens rotations 74
3.2.2 Householder reflections 76
3.3 Generalized Inverses 81
3.4 Exercises 85

4 Nonlinear Systems and Least Squares Problems 89
4.1 Fixed Point Iterations 89
4.2 Newton's Method for Nonlinear Systems 94
4.3 Gauss-Newton Method for Nonlinear Least Squares Problems 101
4.4 Nonlinear Systems Depending on Parameters 108
4.4.1 Structure of the solution 109
4.4.2 Continuation methods 111
4.5 Exercises 124

5 Symmetric Eigenvalue Problems 129
5.1 Condition of General Eigenvalue Problems 129
5.2 Power Method 133
5.3 QR-Algorithm for Symmetric Eigenvalue Problems 136
5.4 Singular Value Decomposition 143
5.5 Exercises 149

6 Three-Term Recurrence Relations 151
6.1 Theoretical Foundations 152
6.1.1 Orthogonality and three-term recurrence relations . . 153
6.1.2 Homogeneous and non-homogeneous recurrence relations 156
6.2 Numerical Aspects 159
6.2.1 Condition numbers 161
6.2.2 Idea of the Miller algorithm 167
6.3 Adjoint Summation 170
6.3.1 Summation of dominant solutions 171
6.3.2 Summation of minimal solutions 174
6.4 Exercises 178

7 Interpolation and Approximation 182
7.1 Classical Polynomial Interpolation 183
7.1.1 Uniqueness and condition number 183
7.1.2 Hermite interpolation and divided differences 187
7.1.3 Approximation error 196
7.1.4 Min-max property of Chebyshev polynomials 198
7.2 Trigonometric Interpolation 201
7.3 Bézier Techniques 209
7.3.1 Bernstein polynomials and Bézier representation . . . 210
7.3.2 De Casteljau's algorithm 217
7.4 Splines 225
7.4.1 Spline spaces and B-splines 226
7.4.2 Spline interpolation 234
7.4.3 Computation of cubic splines 238

7.5 Exercises 242

8 Large Symmetric Systems of Equations and Eigenvalue Problems 245
8.1 Classical Iteration Methods 247
8.2 Chebyshev Acceleration 253
8.3 Method of Conjugate Gradients 258
8.4 Preconditioning 266
8.5 Lanczos Methods 272
8.6 Exercises 277

9 Definite Integrals 281
9.1 Quadrature Formulas 282
9.2 Newton-Cotes Formulas 286
9.3 Gauss-Christoffel Quadrature 292
9.3.1 Construction of the quadrature formula 292
9.3.2 Computation of knots and weights 298
9.4 Classical Romberg Quadrature 301
9.4.1 Asymptotic expansion of the trapezoidal sum 301
9.4.2 Idea of extrapolation 303
9.4.3 Details of the algorithm 310
9.5 Adaptive Romberg Quadrature 313
9.5.1 Principle of adaptivity 314
9.5.2 Estimation of the approximation error 315
9.5.3 Derivation of the algorithm 319
9.6 Hard Integration Problems 325
9.7 Adaptive Multigrid Quadrature 329
9.7.1 Local error estimation and refinement rules 329
9.7.2 Global error estimation and details of the algorithm . 333
9.8 Exercises 337

References 341

Notation 347

Index 349

1 Linear Systems

We start with the classical *Gaussian elimination method* for solving systems of linear equations. Carl Friedrich Gauss (1777–1855) describes the method in his 1809 work on celestial mechanics "Theoria Motus Corporum Coelestium" [33] by saying "the values can be obtained with the usual elimination method". The method was used there in connection with the least squares method (cf. Section 3). In fact the method had been used previously by Lagrange in 1759 and had been known in China as early as the first century B.C. The problem is to solve a system of n linear equations

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & & & & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array}$$

or, in short form

$$Ax = b,$$

where $A \in \mathrm{Mat}_n(\mathbf{R})$ is a real (n,n)-matrix and $b, x \in \mathbf{R}^n$ are real n-vectors. Before starting to compute the solution x, we should ask ourselves whether or not the system is solvable or not? From linear algebra, we know the following result which characterizes solvability in terms of the determinant of the matrix A.

Theorem 1.1 *Let $A \in \mathrm{Mat}_n(\mathbf{R})$ be a real square matrix with $\det A \neq 0$ and $b \in \mathbf{R}^n$. Then there exists a unique $x \in \mathbf{R}^n$ such that $Ax = b$.*

Whenever $\det A \neq 0$, the solution $x = A^{-1}b$ can be computed by Cramer's rule. Here we already see a general property of a "good" algorithm, namely the connection of existence and uniqueness of the solution with a numerical method for computing it. The *cost* of computing $\det A$ amounts to $n \cdot n!$ arithmetic operations when the Leibniz representation

$$\det A = \sum_{\sigma \in S_n} \operatorname{sgn} \sigma \cdot a_{1,\sigma(1)} \cdots a_{n,\sigma(n)}$$

of the determinant as a sum of all permutations $\sigma \in S_n$ of the set $\{1, \ldots, n\}$ is used. Even with the recursive scheme involving development in subdeterminants according to Laplace's rule

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

there are 2^n arithmetic operations to be carried out, where $A_{1i} \in \mathrm{Mat}_{n-1}(\mathbf{R})$ is the matrix obtained from A by crossing out the first row and the i-th column. As we will see, all methods to be described in what follows are more efficient than Cramer's rule for $n \geq 3$ so that the latter is interesting only for $n = 2$.

Remark 1.2 Of course, we expect that a good numerical method solves a given problem at minimal cost (in terms of arithmetic operations). Intuitively there is a minimal cost for each problem which is called the *complexity* of the problem. The closer the cost of an algorithm is to the complexity of the problem, the more efficient that algorithm is. The cost of a concrete algorithm is therefore always an *upper bound* for the complexity of the problem it solves. Obtaining *lower bounds* is in general much more difficult — for details see the monograph of TRAUB and WOZNIAKOWSKI [75].

The notation $x = A^{-1}b$ could suggest the idea of computing the solution of $Ax = b$ by first computing the inverse matrix A^{-1} and then applying it to b. However the computation of A^{-1} inherently contains all difficulties related to solving $Ax = b$ for *arbitrary* right hand sides b. We will see in the second chapter that the computation of A^{-1} can be "badly behaved", even when for special b the solution of $Ax = b$ is "well behaved". $x = A^{-1}b$ is therefore meant only as a formal notation which has nothing to do with the actual computation of the solution x. One should therefore avoid talking about "inverting matrices", when in fact one is concerned with "solving linear systems".

Remark 1.3 There has been a long standing bet by an eminent colleague, who wagered a significant amount, that in practice the problem of "inverting a matrix" is always avoidable. As far as we know he has won the bet in all cases.

1.1 Solution of Triangular Systems

In the search for an efficient solution method for arbitrary linear systems we will first consider cases that are particularly easy to solve. Simplest is

obviously the case of a diagonal matrix A, where the corresponding system consists of n independent scalar equations. The method that transforms a general system into a diagonal one is called the *Gauss-Jordan method*. However we will omit it here, since it is less efficient than the method described in Section 1.2. Next, in terms of difficulty, is the case of a *triangular system*

$$\begin{array}{ccccccccc} r_{11}x_1 & + & r_{12}x_2 & + & \dots & + & r_{1n}x_n & = & z_1 \\ & & r_{22}x_2 & + & \dots & + & r_{2n}x_n & = & z_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & r_{nn}x_n & = & z_n \, , \end{array}$$

and in matrix notation

$$Rx = z \, , \tag{1.1}$$

where R is an upper triangular matrix , i.e. $r_{ij} = 0$ for all $i > j$. Obviously the components of x can be obtained recursively starting with the n'th row:

$$\begin{array}{lcll} x_n & := & z_n/r_{nn} & , \text{ if } r_{nn} \neq 0 \, , \\ x_{n-1} & := & (z_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1} & , \text{ if } r_{n-1,n-1} \neq 0 \, , \\ \vdots & & & \\ x_1 & := & (z_1 - r_{12}x_2 - \dots - r_{1n}x_n)/r_{11} & , \text{ if } r_{11} \neq 0 \, . \end{array}$$

Now, the determinant of the upper triangular matrix R is $\det R = r_{11} \cdots r_{nn}$, and therefore

$$\det R \neq 0 \iff r_{ii} \neq 0 \text{ for all } i = 1, \dots, n \, .$$

The above defined algorithm is therefore applicable (as in the case of Cramer's rule) if and only if $\det R \neq 0$, i.e. under the hypothesis of the existence and uniqueness theorem. The computational cost amounts to:

a) for the i-th row: $n - i$ additions and multiplications, and one division

b) for rows n through 1 together:

$$\sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2} \doteq \frac{n^2}{2}$$

multiplications and as many additions.

Here the notation "$\doteq$" stands for "equal up to lower order terms", i.e. we consider only the term containing the highest power of n, which dominates the cost for large values of n.

The solution of a triangular system of the form

$$Lx = z, \tag{1.2}$$

with a lower triangular matrix L, is completely analogous, if one starts now with the first row and works through to last one. This way of solving triangular systems is called *backward substitution* in case of (1.1) and *forward substitution* in case of (1.2). The name substitution or replacement is used because each component of the right hand side vector is successively replaced by the solution, as indicated in the following scheme describing the content of the vector stored in the memory of the machine (memory scheme) at each step:

$$\begin{aligned}
&(z_1, z_2, \ldots, z_{n-1}, z_n)\\
&(z_1, z_2, \ldots, z_{n-1}, x_n)\\
&\qquad\qquad \vdots\\
&(z_1, x_2, \ldots, x_{n-1}, x_n)\\
&(x_1, x_2, \ldots, x_{n-1}, x_n)\,.
\end{aligned}$$

1.2 Gaussian Elimination

We now return to the general linear system $Ax = b$,

$$\begin{array}{lcllcl}
a_{11}x_1 & + & a_{12}x_2 & +\ldots+ & a_{1n}x_n & = & b_1\\
a_{21}x_1 & + & a_{22}x_2 & +\ldots+ & a_{2n}x_n & = & b_2\\
\vdots & & \vdots & & \vdots & & \vdots\\
a_{n1}x_1 & + & a_{n2}x_2 & +\ldots+ & a_{nn}x_n & = & b_n
\end{array} \tag{1.3}$$

and try to transform it into a triangular one. The first row does not have to be changed. We want to manipulate the remaining rows such that the coefficients in front of x_1 vanish, i.e. the variable x_1 from rows 2 through n is *eliminated.* Thus we produce a system of the form

$$\begin{array}{rcccccl} a_{11}x_1 & + & a_{12}x_2 & + \ldots + & a_{1n}x_n & = & b_1 \\ & & a'_{22}x_2 & + \ldots + & a'_{2n}x_n & = & b'_2 \\ & & \vdots & & & & \vdots \\ & & a'_{n2}x_2 & + \ldots + & a'_{nn}x_n & = & b'_n \,. \end{array} \tag{1.4}$$

Having achieved this we can apply the same procedure to the last $n-1$ rows in order to recursively obtain a triangular system. Therefore it is sufficient to examine the first elimination step from (1.3) to (1.4). We assume that $a_{11} \neq 0$. In order to eliminate the term $a_{i1}x_1$ in row i $(i = 2, \ldots, n)$, we subtract from row i a multiple of row 1 (unaltered), i.e.

$$\text{new row } i \;:= \text{row } i - l_{i1} \cdot \text{row } 1$$

or explicitly

$$\underbrace{(a_{i1} - l_{i1}a_{11})}_{=0} x_1 + \underbrace{(a_{i2} - l_{i1}a_{12})}_{=a'_{i2}} x_2 + \cdots + \underbrace{(a_{in} - l_{i1}a_{1n})}_{=a'_{in}} x_n = \underbrace{b_i - l_{i1}b_1}_{=b'_i} \,.$$

From $a_{i1} - l_{i1}a_{11} = 0$ it follows immediately that $l_{i1} = a_{i1}/a_{11}$. Therefore the first elimination step can be performed under the assumption $a_{11} \neq 0$. The element a_{11} is called a *pivot element* and the first row a *pivot row.* After this first elimination step there remains an $(n-1, n-1)$-submatrix in rows 2 through n. By applying repeatedly the elimination procedure we obtain a sequence

$$A = A^{(1)} \to A^{(2)} \to \ldots \to A^{(n)} =: R$$

of matrices of the special form

$$A^{(k)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & \cdots & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & \cdots & \cdots & a_{2n}^{(2)} \\ & & \ddots & & & \vdots \\ & & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} \end{bmatrix} \tag{1.5}$$

with an $(n-k+1, n-k+1)$-submatrix, to which we can apply the elimination step

$$\begin{aligned} l_{ik} &:= a_{ik}^{(k)} / a_{kk}^{(k)} && \text{for } i = k+1, \ldots, n \\ a_{ij}^{(k+1)} &:= a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)} && \text{for } i, j = k+1, \ldots, n \\ b_i^{(k+1)} &:= b_i^{(k)} - l_{ik} b_k^{(k)} && \text{for } i = k+1, \ldots, n \end{aligned}$$

whenever the *pivot* $a_{kk}^{(k)}$ does not vanish. Since every elimination step is a linear operation applied to the rows of A, the transformation from $A^{(k)}$ and $b^{(k)}$ into $A^{(k+1)}$ and $b^{(k+1)}$ can be represented as a *premultiplication* by a matrix $L_k \in \mathrm{Mat}_n(\mathbf{R})$, i.e.

$$A^{(k+1)} = L_k A^{(k)} \quad \text{and} \quad b^{(k+1)} = L_k b^{(k)} .$$

(In case of operations on columns one obtains an analogous *postmultiplication*). The matrix

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & 1 \end{bmatrix}$$

is called a *Frobenius matrix*; It has the nice property that its inverse L_k^{-1} is obtained from L_k by changing the signs of the l_{ik}'s. Furthermore the product of the L_k^{-1}'s satisfies

$$L := L_1^{-1} \cdot \ldots \cdot L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & & \ddots & \ddots & \\ l_{n1} & \ldots & \ldots & l_{n,n-1} & 1 \end{bmatrix}$$

In this way we have reduced the system $Ax = b$ to the equivalent triangular system $Rx = z$ with

$$R = L^{-1} A \quad \text{and} \quad z = L^{-1} b .$$

A lower (resp. upper) triangular matrix, whose main diagonal elements are all equal to one is a called a *unit* lower (resp. upper) triangular matrix. The above representation $A = LR$ of the matrix A as a product of a unit lower triangular matrix L and an upper triangular matrix R is called the *Gaussian triangular factorization*, or briefly *LR factorization* of A. In the English literature the matrix R is often denoted by U (from *u*pper triangular) and the corresponding Gaussian triangular factorization is called the LU factorization. If such a factorization exists, then L and R are uniquely determined (cf. Exercise 1.2).

Algorithm 1.4 *Gaussian Elimination.*

a) $A = LR$ Triangular Factorization, R upper and L lower triangular matrix

b) $Lz = b$ Forward Substitution

c) $Rx = z$ Backward Substitution.

The *memory scheme* for the Gaussian elimination is based upon the representation (1.5) of the matrices $A^{(k)}$. In the remaining memory locations one can store the l_{ik}'s, because the other elements, with values 0 or 1, do not have to be stored. The entire memory cost for Gaussian elimination amounts to $n(n+1)$ memory locations, i.e. as many as needed to define the problem. The cost in terms of number of multiplications is

$\sim \sum_{k=1}^{n-1} k^2 \doteq n^3/3$ for a)

$\sim \sum_{k=1}^{n-1} k \doteq n^2/2$ both for b) and c).

Therefore the main cost comes from the LR-factorization. However, if different right hand sides $b_1, \ldots, b_j$ are considered, then this factorization has to be carried out only once.

1.3 Pivoting Strategies and Iterative Refinement

As seen from the simple example

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \det A = -1, \quad a_{11} = 0$$

there are cases where the triangular factorization fails even when $\det A \neq 0$. However an interchange of rows leads to the simplest LR-factorization we

can imagine, namely

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \longrightarrow \quad \bar{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I = LR \text{ with } L = R = I \ .$$

In the numerical implementation of Gaussian Elimination difficulties can arise not only when pivot elements vanish, but also when they are "too small".

Example 1.5 (cf. [30]) We compute the solution of the system

$$\begin{aligned} (a) \quad & 1.00 \cdot 10^{-4}\, x_1 + 1.00\, x_2 = 1.00 \\ (b) \quad & 1.00\, x_1 + 1.00\, x_2 = 2.00 \end{aligned}$$

on a machine, which, for the sake of simplicity, works only with three exact decimal figures. By completing the numbers with zeros, we obtain the "exact" solution with four correct figures

$$x_1 = 1.000 \quad x_2 = 0.9999 \ ,$$

and with three correct figures

$$x_1 = 1.00 \quad x_2 = 1.00 \ .$$

Let us now carry out the Gaussian elimination on our computer, i.e. in three exact decimal figures

$$l_{21} = \frac{a_{21}}{a_{11}} = \frac{1.00}{1.00 \cdot 10^{-4}} = 1.00 \cdot 10^4 \ ,$$

$$\begin{aligned} &(1.00 - 1.00 \cdot 10^4 \cdot 1.00 \cdot 10^{-4}) x_1 + (1.00 - 1.00 \cdot 10^4 \cdot 1.00) x_2 \\ &\qquad = 2.00 - 1.00 \cdot 10^4 \cdot 1.00 \ . \end{aligned}$$

Thus we obtain the upper triangular system

$$\begin{aligned} 1.00 \cdot 10^{-4}\, x_1 + 1.00\, x_2 &= 1.00 \\ -1.00 \cdot 10^4\, x_2 &= -1.00 \cdot 10^4 \end{aligned}$$

and the "solution"

$$x_2 = 1.00 \text{ (true)} \quad x_1 = 0.00 \text{ (false!)} \ .$$

However, if before starting the elimination, we interchange the rows

$$\begin{aligned} (\tilde{a}) \quad & 1.00\, x_1 + 1.00\, x_2 = 2.00 \\ (\tilde{b}) \quad & 1.00 \cdot 10^{-4}\, x_1 + 1.00\, x_2 = 1.00 \ , \end{aligned}$$

then $\tilde{l}_{21} = 1.00 \cdot 10^{-4}$, which yields the upper triangular system

$$\begin{aligned} 1.00\, x_1 \;+\; 1.00\, x_2 &= 2.00 \\ 1.00\, x_2 &= 1.00 \end{aligned}$$

as well as the "true solution"

$$x_2 = 1.00 \quad x_1 = 1.00 \; .$$

By interchanging the rows in the above example we obtain

$$|\tilde{l}_{21}| < 1 \;\text{ and }\; |\tilde{a}_{11}| \geq |\tilde{a}_{21}| \; .$$

Thus, the new pivot $\tilde{a}_{11}$ is the largest element, in absolute value, of the first column.

We can deduce the *partial pivoting* or *column pivoting* strategy from the above considerations . This strategy is to choose at each Gaussian elimination step as pivot row the one having the largest element in absolute value within the pivot column. More precisely, we can formulate the following algorithm:

Algorithm 1.6 *Gaussian elimination with column pivoting*

a) In elimination step $A^{(k)} \rightarrow A^{(k+1)}$ choose a $p \in \{k, \ldots, n\}$, such that

$$|a_{pk}^{(k)}| \geq |a_{jk}^{(k)}| \quad \text{for } j = k, \ldots, n$$

Row p becomes pivot row.

b) Interchange rows p and k

$$A^{(k)} \rightarrow \tilde{A}^{(k)} \;\text{ with }\; \tilde{a}_{ij}^{(k)} = \begin{cases} a_{kj}^{(k)} & , \text{ if } i = p \\ a_{pj}^{(k)} & , \text{ if } i = k \\ a_{ij}^{(k)} & , \text{ otherwise} \end{cases}$$

Now we have

$$|\tilde{l}_{ik}| = \left| \frac{\tilde{a}_{ik}^{(k)}}{\tilde{a}_{kk}^{(k)}} \right| = \left| \frac{\tilde{a}_{ik}^{(k)}}{a_{pk}^{(k)}} \right| \leq 1 \; .$$

c) Perform the next elimination step for $\tilde{A}^{(k)}$, i.e.

$$\tilde{A}^{(k)} \rightarrow A^{(k+1)} \; .$$

Remark 1.7 Instead of column pivoting with row interchange one can also perform *row pivoting* with column interchange. Both strategies require at most $O(n^2)$ additional operations. If we combine both methods and look at each step for the largest element in absolute value of the entire remaining matrix, then we need $O(n^3)$ additional operations. This *total pivoting* strategy is therefore almost never employed.

In the following formal description of the triangular factorization with partial pivoting we use *permutation matrices* $P \in \mathrm{Mat}_n(\mathbf{R})$. For each permutation $\pi \in S_n$ we define the corresponding matrix

$$P_\pi = \left[e_{\pi(1)} \cdots e_{\pi(n)}\right],$$

where $e_j = (\delta_{1j}, \ldots, \delta_{nj})^T$ is the j-th unit vector. A permutation π of the rows of the matrix A can be expressed as a premultiplication by P_π

$$\text{Permutation of rows } \pi\colon A \longrightarrow P_\pi A.$$

and analogously a permutation π of the columns as a postmultiplication

$$\text{Permutation of columns } \pi\colon A \to AP_\pi\ .$$

It is known from linear algebra that the mapping

$$\pi \longmapsto P_\pi$$

is a group homeomorphism $S_n \to \mathbf{O}(n)$ of the symmetric group S_n into the orthogonal group $\mathbf{O}(n)$. In particular we have

$$P^{-1} = P^T\ .$$

The determinant of the permutation matrix is just the sign of the corresponding permutation

$$\det P_\pi = \operatorname{sgn}\pi \in \{\pm 1\}\ ,$$

i.e. it is equal to $+1$, if π consists of an even number of transpositions, and -1 otherwise. The following proposition shows that, *theoretically*, the triangular factorization with partial pivoting fails only when the matrix A is singular.

Theorem 1.8 *For every invertible matrix A there exists a permutation matrix P such that a triangular factorization of the form*

$$PA = LR$$

is possible. Here P can be chosen so that all elements of L are less than or equal to one in absolute value, i.e.

$$|L| \leq 1\ .$$

Proof. We employ the LR-factorization algorithm with column pivoting. Since $\det A \neq 0$, there is a transposition $\tau_1 \in S_n$ such that the first diagonal element $a_{11}^{(1)}$ of the matrix

$$A^{(1)} = P_{\tau_1} A$$

is different from zero and is also the largest element in absolute value in the first column, i.e.

$$0 \neq |a_{11}^{(1)}| \geq |a_{i1}^{(1)}| \quad \text{for} \quad i = 1, \ldots, n \, .$$

After eliminating the remaining elements of the first column we obtain the matrix

$$A^{(2)} = L_1 A^{(1)} = L_1 P_{\tau_1} A = \left[\begin{array}{c|ccc} a_{11}^{(1)} & * & \cdots & * \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right] ,$$

where all elements of L_1 are less than or equal to one in absolute value, i.e. $|L_1| \leq 1$, and $\det L_1 = 1$. The remaining matrix $B^{(2)}$ is again invertible since $|a_{11}^{(1)}| \neq 0$ and

$$0 \neq \operatorname{sgn}(\tau_1) \det A = \det A^{(2)} = a_{11}^{(1)} \det B^{(2)} \, .$$

Now by we can proceed by induction and obtain

$$R = A^{(n)} = L_{n-1} P_{\tau_{n-1}} \cdots L_1 P_{\tau_1} A \, , \tag{1.6}$$

where $|L_k| \leq 1$, and τ_k is either the identity or the transposition of two numbers $\geq k$. If $\pi \in S_n$ only permutes numbers $\geq k+1$, then the Frobenius matrix

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & 1 \end{bmatrix}$$

satisfies

$$\hat{L}_k = P_\pi L_k P_\pi^{-1} = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{\pi(k+1),k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{\pi(n),k} & & 1 \end{bmatrix} . \tag{1.7}$$

Therefore we can separate Frobenius matrices L_k and permutations P_{τ_k} by inserting in (1.6) the identities $P_{\tau_k}^{-1} P_{\tau_k}$ i.e.

$$R = L_{n-1} P_{\tau_{n-1}} L_{n-2} P_{\tau_{n-1}}^{-1} P_{\tau_{n-1}} P_{\tau_{n-2}} L_{n-3} \cdots L_1 P_{\tau_1} A .$$

Hence we obtain

$$R = \hat{L}_{n-1} \cdots \hat{L}_1 P_{\pi_0} A \quad \text{with} \quad \hat{L}_k = P_{\pi_k} L_k P_{\pi_k}^{-1} ,$$

where $\pi_{n-1} := \mathrm{id}$ and $\pi_k = \tau_{n-1} \cdots \tau_{k+1}$ for $k = 0, \ldots, n-2$. Since the permutation π_k interchanges in fact only numbers $\geq k+1$, the matrices $\hat{L}_k$ are of the form (1.7). Consequently

$$P_{\pi_0} A = LR$$

with $L := \hat{L}_1^{-1} \cdots \hat{L}_{n-1}^{-1}$ or explicitly

$$L = \begin{bmatrix} 1 & & & & \\ l_{\pi_1(2),1} & 1 & & & \\ l_{\pi_1(3),1} & l_{\pi_2(3),2} & 1 & & \\ \vdots & & \ddots & \ddots & \\ l_{\pi_1(n),1} & & \cdots & l_{\pi_{n-1}(n),n-1} & 1 \end{bmatrix} ,$$

and therefore $|L| \leq 1$. □

Note that we have used the Gaussian elimination algorithm with column pivoting to constructively prove an existence theorem.

Remark 1.9 Let us also note that the *determinant* of A can be easily computed by using the $PA = LR$ factorization of Proposition 1.8 via the formula

$$\det A = \det(P) \cdot \det(LR) = \operatorname{sgn}(\pi_0) \cdot r_{11} \cdots r_{nn}$$

A warning should be made against the naive computation of determinants! As is well known, multiplication of a linear system by an arbitrary scalar α results in

$$\det(\alpha A) = \alpha^n \det A \, .$$

This trivial transformation may be used to convert a "small" determinant into an arbitrarily "large" one and the other way around. The only invariants under this class of trivial transformations are the Boolean quantities $\det A = 0$ or $\det A \neq 0$; for an odd n we have additionally $\operatorname{sgn}(\det A)$. The above noted theoretical difficulty will lead later on to a completely different characterization of the solvability of linear systems.

Furthermore, it is apparent that the pivoting strategy can be arbitrarily changed by multiplying different rows by different scalars. This observation leads to the question of *scaling*. By row scaling we mean premultiplication of A by a diagonal matrix

$$A \to D_r A \, , \quad D_r \text{ diagonal matrix}$$

and analogously, by column scaling we mean postmultiplication by a diagonal matrix

$$A \to A D_c \, , \quad D_c \text{ diagonal matrix} \, .$$

(As we have already seen in the context of Gaussian elimination, linear operations on the rows of a matrix can be expressed by premultiplication with suitable matrices and correspondingly operations on columns are represented by postmultiplication.) Mathematically speaking scaling changes the length of the basis vectors of the range (row scaling) and of the domain (column scaling) of the linear mapping defined by the matrix A, respectively. If this mapping models a physical phenomenon then we can interpret scaling as a change of unit, or gauge transformation (e.g. from Å to km). In order to make the solution of the linear system $Ax = b$ independent of the choice of unit we have to appropriately scale the system by pre– or postmultiplying the matrix A by suitable diagonal matrices:

$$A \to \tilde{A} := D_r A D_c \, ,$$

where

$$D_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \quad \text{and} \quad D_c = \operatorname{diag}(\tau_1, \ldots, \tau_n) \, .$$

At first glance the following three strategies seem to be reasonable:

a) *Row equilibration* of A with respect to a vector norm $\|\cdot\|$. Let A^i be the i-th row of A and assume that there are no zero rows. By setting $D_s := I$ and

$$\sigma_i := \|A^i\|^{-1} \text{ for } i = 1, \ldots, n,$$

we make all rows of $\tilde{A}$ have norm one.

b) *Column equilibration*. Suppose that there are no columns A_j of A equal to zero. By setting $D_z := I$ and

$$\tau_j := \|A_j\|^{-1} \text{ for } j = 1, \ldots, n,$$

we make all columns of $\tilde{A}$ have norm one.

c) Following a) and b) it is natural to require that all rows of A have the same norm and at the same time that all columns of A have the same norm. In order to determine σ_i and τ_j up to a mutual common factor one has to solve a *nonlinear* system with $2n-2$ unknowns. This obviously requires a great deal more effort than solving the original problem. As will be seen in the fourth chapter the solution of this nonlinear system requires the solution of a sequence of linear systems, now in $2n-2$ unknowns, for which the problem of scaling has to be addressed again.

Because of this dilemma, most programs (e.g. LINPACK [26]) leave the scaling issue to the user.

The pivoting strategies discussed above cannot prevent the possibility of computing a rather inaccurate solution $\tilde{x}$. How can one improve the accuracy of $\tilde{x}$ without too much effort? Of course we can simply discard the solution $\tilde{x}$ altogether and try to compute a "better" solution by using a higher machine precision. However in this way all information obtained in computing $\tilde{x}$ is lost. This is avoided in the so called *iterative refinement* method by explicitly evaluating the *residual*

$$r(y) := b - Ay = A(x - y)$$

The absolute error $\Delta x_0 := x - x_0$ of $x_0 := \tilde{x}$ satisfies the equation

$$A\Delta x_0 = r(x_0)\,. \tag{1.8}$$

In solving this *corrector equation* (1.8), we obtain an approximate correction $\tilde{\Delta} x_0 \neq \Delta x_0$ which is again afflicted by rounding errors. In spite of this fact we expect that the approximate solution

$$x_1 := x_0 + \tilde{\Delta} x_0$$

is "better" than x_0. The idea of iterative refinement consists in repeating this process until the approximate solution x_i is "accurate enough". We should remark that the linear system (1.8) differs from the original linear system only by the right hand side, so that the computation of the corrections Δx_i requires little effort. In Section 2.4.3 we will make precise the meaning of the terms "better approximate solution" and "accurate enough". In fact iterative refinement works excellently in conjunction with Gaussian elimination. In Section 2.4.3 we will state the substantial result of SKEEL [70] that for triangular factorization with column pivoting, a single refinement step is sufficient for obtaining a suitably accurate solution of the given problem.

1.4 Cholesky's Method for Symmetric Positive Definite Matrices

We want now to apply Gaussian elimination to the special class of systems of equations with symmetric positive definite matrices. It will become clear that in this case, the triangular factorization can be substantially simplified. We recall that a symmetric matrix $A = A^T \in \mathrm{Mat}_n(\mathbf{R})$ is *positive definite* if and only if

$$\langle x, Ax \rangle > 0 \quad \text{for all } x \neq 0 \,. \tag{1.9}$$

We call such matrices for short *spd-matrices.*

Theorem 1.10 *For any spd-matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ we have:*

i) *A is invertible.*

ii) *$a_{ii} > 0$ for $i = 1, \ldots, n$.*

iii) $\max_{i,j=1,\ldots,n} |a_{ij}| = \max_{i=1,\ldots,n} a_{ii}$.

iv) *Each rest matrix obtained during Gaussian elimination without pivoting is also symmetric positive definite.*

Obviously iii) and iv) say that row or column pivoting is not necessary for LR factorization, in fact even absurd because it might destroy the structure of A. In particular iii) means that total pivoting can be reduced to diagonal pivoting.

Proof. The invertibility of A follows immediately from (1.9). If we put in (1.9) a basis vector e_i instead of x, it follows immediately that $a_{ii} = \langle e_i, Ae_i \rangle > 0$ and therefore the second claim is proven. The third statement is proved similarly, cf. Exercise 1.7. In order to prove statement iv) we write

$A = A^{(1)}$ as

$$A^{(1)} = \left[\begin{array}{c|c} a_{11} & z^T \\ \hline z & B^{(1)} \end{array}\right] \tag{1.10}$$

where $z = (a_{12}, \ldots, a_{1n})^T$ and after one elimination step we obtain

$$A^{(2)} = L_1 A^{(1)} = \left[\begin{array}{c|c} a_{11} & z^T \\ \hline 0 & \\ \vdots & B^{(2)} \\ 0 & \end{array}\right] \quad \text{with} \quad L_1 = \begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{bmatrix}.$$

Now if we premultiply $A^{(2)}$ with L_1^T, then z^T in the first row is also eliminated and and the submatrix $B^{(2)}$ remains unchanged, i.e.

$$L_1 A^{(1)} L_1^T = \left[\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right].$$

The operation $A \to L_1 A L_1^T$ describes a change of basis for the bilinear form defined by the symmetric matrix A. According to the inertia theorem of Sylvester, $L_1 A^{(1)} L_1^T$ and with it $B^{(2)}$ remain positive definite. □

Together with the LR factorization we can deduce now the *rational Cholesky factorization* for symmetric positive definite matrices.

Theorem 1.11 *For every symmetric positive definite matrix A there exists a uniquely determined factorization of the form*

$$A = LDL^T,$$

where L is a unit lower triangular matrix and D a positive diagonal matrix.

Proof. We continue the construction from the proof of Theorem 1.10 for $k = 2, \ldots, n-1$ and obtain immediately L as the product of $L_1^{-1}, \ldots, L_{n-1}^{-1}$ and D as the diagonal matrix of the pivots. □

Corollary 1.12 *Since* $D = \mathrm{diag}(d_i)$ *is positive, the square root* $D^{\frac{1}{2}} = \mathrm{diag}(\sqrt{d_i})$ *exists and with it the* Cholesky factorization

$$A = \bar{L}\bar{L}^T , \tag{1.11}$$

where $\bar{L}$ *is the lower triangular matrix* $\bar{L} := LD^{\frac{1}{2}}$.

The matrix $\bar{L} = (l_{ij})$ can be computed by using *Cholesky's method*, :

Algorithm 1.13 *Cholesky's method.*

for $k := 1$ **to** n **do**
  $l_{kk} := (a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2)^{1/2}$;
  for $i := k+1$ **to** n **do**
    $l_{ik} = (a_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj})/l_{kk}$;
  end for
end for

The derivation of this algorithm is nothing more than the element-wise evaluation of equation (1.11)

$$\begin{bmatrix} l_{11} & & \\ \vdots & \ddots & \\ l_{n1} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & \dots & l_{n1} \\ & \ddots & \vdots \\ & & l_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$$

$$\begin{aligned} i = k : a_{kk} &= l_{k1}^2 + \cdots + l_{k,k-1}^2 + l_{kk}^2 \\ i > k : a_{ik} &= l_{i1}l_{k1} + \cdots + l_{i,k-1}l_{k,k-1} + l_{ik}l_{kk} . \end{aligned}$$

The sophistication of the method is contained in the sequence of computations for the elements of $\bar{L}$. As for the computational cost we have

$$\sim \frac{1}{6}n^3 \text{ multiplications and } n \text{ square roots} .$$

In contrast, the rational Cholesky factorization requires no square roots, but only rational operations (whence the name). By smart programming the cost can be kept here also to $\sim \frac{1}{6}n^3$. An advantage of the rational Cholesky factorization is that almost singular matrices D can be recognized. Also the method can be extended to symmetric indefinite matrices ($x^T Ax \neq 0$ for all x).

Remark 1.14 The supplemental spd property has obviously led to a sensible reduction of the computational cost. At the same time, this property forms the basis of completely different types of solution methods that will be described in Section 8.

1.5 Exercises

Exercise 1.1 Give an example of a full nonsingular (3,3)-matrix for which Gaussian elimination without pivoting fails.

Exercise 1.2 a) Show that the unit (nonsingular) lower (upper) triangular matrices form a subgroup of $\mathrm{GL}(n)$.

b) Apply a) to show that the representation

$$A = LR$$

of a nonsingular matrix $A \in \mathrm{GL}(n)$ as the product of a unit lower triangular matrix L and a nonsingular upper triangular matrix R is unique, provided it exists.

c) If $A = LR$ as in b), then L and R can be computed by Gaussian triangular factorization. Why is this another proof of b) ? Hint: use induction.

Exercise 1.3 A matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ is called strictly diagonally dominant if

$$|a_{ii}| > \sum_{\substack{i=1 \\ j \neq i}}^{n} |a_{ij}| \quad \text{for } i = 1, \ldots, n.$$

Show that Gaussian triangular factorization can be performed for any matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ with a strictly diagonally dominant transpose A^T. In particular any such A is invertible. Hint: use induction.

Exercise 1.4 The *numerical range* $W(A)$ of a matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ is defined as the set

$$W(A) := \{\langle Ax, x\rangle \mid \langle x, x\rangle = 1,\ x \in \mathbf{R}^n\}$$

Here $\langle \cdot, \cdot \rangle$ is the Euclidean scalar product on $\mathbf{R}^n$.

a) Show that the matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ has an LR factorization (L unit lower triangular, R upper triangular) if and only if the origin is not contained in the numerical range of A, i.e.

$$0 \notin W(A) .$$

Hint: use induction.

b) Use a) to show that the matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 5 & 3 \end{bmatrix}$$

has no LR factorization.

Exercise 1.5 Program the Gaussian triangular factorization. The program should read data A and b from a data file and should be tested on the following examples:

a) with the matrix from Example 1.1,

b) with $n = 1$, $A = 25$ and $b = 4$,

c) with $a_{ij} = i^{j-1}$ and $b_i = i$ for $n = 7$, 15 and 50.

Compare in each case the computed and the exact solutions.

Exercise 1.6 Gaussian factorization with column pivoting applied to the matrix A delivers the factorization $PA = LR$, where P is the permutation matrix produced during elimination. Show that:

a) Gaussian elimination with column pivoting is invariant with respect to

i) Permutation of rows of A (with the trivial exception that there are several elements of equal absolute value per column)

ii) Multiplication of the matrix by a number $\sigma \neq 0$, $A \longrightarrow \sigma A$.

b) If D is a diagonal matrix, then Gaussian elimination with column pivoting applied to $\bar{A} := AD$ delivers the factorization $P\bar{A} = L\bar{R}$ with $\bar{R} = RD$.

Consider the corresponding behavior for a row pivoting strategy with column interchange as well as for total pivoting with row and column interchange.

Exercise 1.7 Let the matrix $A \in \mathrm{Mat}_n(\mathbf{R})$ be symmetric positive definite.

a) Show that

$$|a_{ij}| \leq \sqrt{a_{ii}a_{jj}} \leq \frac{1}{2}(a_{ii} + a_{jj}) \quad \text{for all } i, j = 1, \ldots, n.$$

Hint: show first that the matrix $\begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$ is symmetric positive definite for all i, j.

b) Deduce from a) that

$$\max_{i,j} |a_{ij}| = \max_{i} a_{ii} \, .$$

Interpret the result in the context of pivoting strategies.

Exercise 1.8 Show that for any $u, v \in \mathbf{R}^n$ we have:

a) $(I + uv^T)^{-1} = I - \dfrac{uv^T}{1 + v^T u}$, whenever $u^T v \neq -1$

b) $I + uv^T$ is singular whenever $u^T v = -1$.

Exercise 1.9 The linear system $Ax = b$ with matrix

$$A = \left[\begin{array}{c|c} R & v \\ \hline u^T & 0 \end{array}\right] ,$$

is to be solved, where $R \in \mathrm{Mat}_n(\mathbf{R})$ is an invertible upper triangular matrix, $u, v \in \mathbf{R}^n$ and $x, b \in \mathbf{R}^{n+1}$.

a) Specify the triangular factorization of A.

b) Show that A is nonsingular if and only if

$$u^T R^{-1} v \neq 0 \, .$$

c) Formulate an economical algorithm for solving the above linear system and determine its computational cost.

Exercise 1.10 In the context of probability distributions one encounters matrices $A \in \mathrm{Mat}_n(\mathbf{R})$ with the following properties:

i) $\sum_{i=1}^{n} a_{ij} = 0$ for $j = 1, \ldots, n$;

ii) $a_{ii} < 0$ and $a_{ij} \geq 0$ for $i = 1, \ldots, n$ and $j \neq i$.

Let $A = A^{(1)}, A^{(2)}, \ldots, A^{(n)}$ be produced during Gaussian elimination. Show that:

a) $|a_{11}| \geq |a_{i1}|$ for $i = 2, \ldots, n$;

b) $\sum_{i=2}^{n} a_{ij}^{(2)} = 0$ for $j = 2, \ldots, n$;

c) $a_{ii}^{(1)} \leq a_{ii}^{(2)} \leq 0$ for $i = 2, \ldots, n$;

d) $a_{ij}^{(2)} \geq a_{ij}^{(1)} \geq 0$ for $i, j = 2, \ldots, n$ and $j \neq i$;

e) If the diagonal elements produced successively during the first $n - 2$ Gaussian elimination steps are all nonzero (i.e. $a_{ii}^{(i)} < 0$ for $i = 1, \ldots, n-1$) then $a_{nn}^{(n)} = 0$.

Exercise 1.11 A problem from astrophysics ("cosmic maser") can be formulated as a system of $(n+1)$ linear equations in n unknowns of the form

$$\begin{pmatrix} & & \\ & A & \\ & & \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

where A is the matrix from Exercise 1.10. In order to solve this system we apply Gaussian elimination on the matrix A with the following two additional rules, where the matrices produced during elimination are denoted again by $A = A^{(1)}, \ldots, A^{(n-1)}$ and the relative machine precision is denoted by eps.

a) If during the algorithm $|a_{kk}^{(k)}| \leq |a_{kk}|$eps for some $k < n$, then shift simultaneously column k and row k to the end and the other columns and rows towards the front (rotation of rows and columns).

b) If $|a_{kk}^{(k)}| \leq |a_{kk}|$ eps for all remaining $k < n - 1$, then terminate the algorithm.

Show that:

i) If the algorithm does not terminate in b) then after $n - 1$ elimination steps it delivers a factorization of A as $PAP = LR$, where P is a permutation and $R = A^{(n-1)}$ is an upper triangular matrix with $r_{nn} = 0$, $r_{ii} < 0$ for $i = 1, \ldots, n-1$ and $r_{ij} \geq 0$ for $j > i$.

ii) The system has in this case a unique solution x, and all components of x are nonnegative (interpretation: probabilities).

Give a simple scheme for computing x.

Exercise 1.12 Program the algorithm developed in Exercise 1.11 for solving the special system of equations and test the program on two examples

of your choice of dimensions $n = 5$ and $n = 7$, as well as on the matrix

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

Exercise 1.13 Let a linear system $Cx = b$ be given, where C is an invertible $(2n, 2n)$-matrix of the following special form:

$$C = \begin{bmatrix} A & B \\ B & A \end{bmatrix} , \quad A, B \text{ invertible} .$$

a) Let C^{-1} be partitioned as C:

$$C^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix} .$$

Prove SCHUR's identity:

$$E = H = (A - BA^{-1}B)^{-1} \quad \text{and} \quad F = G = (B - AB^{-1}A)^{-1} .$$

b) Let $x = (x_1, x_2)^T$ and $b = (b_1, b_2)^T$ be likewise partitioned and

$$(A + B)y_1 = b_1 + b_2, \quad (A - B)y_2 = b_1 - b_2 .$$

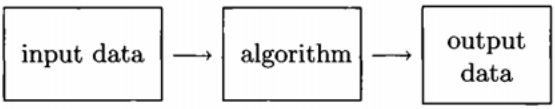
Show that

$$x_1 = \frac{1}{2}(y_1 + y_2) , \quad x_2 = \frac{1}{2}(y_1 - y_2) .$$

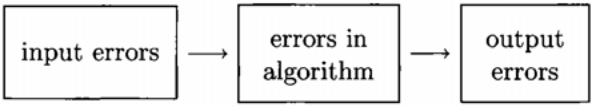
Numerical advantage?

2 Error Analysis

In the last chapter, we introduced a class of methods for the numerical solution of linear systems. There, from a given input (A, b) we computed the solution $f(A, b) = A^{-1}b$. In a more abstract formulation the problem consists in evaluating a mapping $f : U \subset X \to Y$ at a point $x \in U$. The numerical solution of such a problem (f, x) computes the result $f(x)$ from the input x by means of an algorithm that eventually produces some intermediate values as well.



In this chapter we want to see how errors arise in this process and in particular to see if Gaussian elimination is indeed a dependable method. The errors in the numerical result arise from *errors in the data* or *input errors* as well as from *errors in the algorithm*.



In principle we are powerless against the former, as they belong to the given problem and at best they can be avoided by changing the setting of the problem. The situation appears to be different with the errors caused by the algorithm. Here we have the chance to avoid, or to diminish, errors by changing the method. The distinction between the two kind of errors will lead us in what follows to the notions of *condition of a problem* and *stability of an algorithm*. First we want to discuss the possible sources of errors.

2.1 Sources of Errors

Even when input data are considered to be given exactly, errors in the data may still occur because of the machine representation of non-integer numbers. With today's usual *floating point representation*, a number z of "real

type" is represented as $z = ad^e$, where the *basis* d is a power of two (as a rule 2, 8 or 16) and the *exponent* e is an integer of a given maximum number of binary positions,

$$e \in \{e_{\min}, \ldots, e_{\max}\} \subset \mathbf{Z} .$$

The so called *mantissa* a is either 0 or a number satisfying $d^{-1} \leq |a| < 1$ and has the form

$$a = v \sum_{i=1}^{l} a_i d^{-i},$$

where $v \in \{\pm 1\}$ is the *sign*, $a_i \in \{0, \ldots, d-1\}$ are the *digits* (it is assumed that $a = 0$ or $a_1 \neq 0$), and l is the *length of the mantissa*. The numbers that are representable in this way form a subset

$$\mathbf{F} := \{x \in \mathbf{R} \mid \text{ there is } a, e \text{ as above, so that } \; x = ad^e\}$$

of real numbers. The range of the exponent e defines the largest and smallest number that can be represented on the machine (by which we mean the processor together with the compiler). The *length of the mantissa* is responsible for the *relative precision* of the representation of real numbers on the given machine. Every number $x \neq 0$ with

$$d^{e_{\min}-1} \leq |x| \leq d^{e_{\max}}(1 - d^{-l})$$

is represented as a *floating point number* by rounding to the closest *machine number* whose relative error is estimated by

$$\frac{|x - \mathrm{fl}(x)|}{|x|} \leq \mathrm{eps} := d^{1-l}/2$$

Here we use for division the convention $0/0 = 0$ and $x/0 = \infty$ for $x > 0$. We say that we have an *underflow* when $|x|$ is smaller than the smallest machine number $d^{e_{\min}-1}$ and, an *overflow* when $|x| > d^{e_{\max}}(1 - d^{-l})$. We call eps the *relative machine precision* or the *machine epsilon*. In the literature this quantity is also denoted by $\mathbf{u}$ for "unit roundoff" or "unit round". For *single precision* in FORTRAN, or *float* in C, we have usually $\mathrm{eps} \approx 10^{-7}$.

Let us imagine that we wanted to enter in the machine a mathematically exact real number x, for example

$$x = \pi = 3.141592653589\ldots ,$$

It is known theoretically that π as an irrational number cannot be represented with a finite mantissa and therefore it is a quantity affected by errors on any computer, e.g. for $\mathrm{eps} = 10^{-7}$

$$\pi \mapsto \mathrm{fl}(\pi) = 3.141593 , \quad |\mathrm{fl}(\pi) - \pi| \leq \mathrm{eps}\, \pi .$$