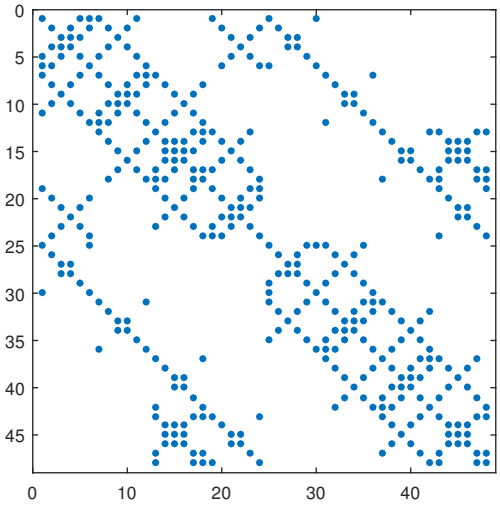
Direct and Iterative Methods for Linear Systems



Gérard Meurant[1]

March 21, 2024

[1]©Gérard Meurant, 30 rue du sergent Bauchat, 75012 Paris, France. (gerard.meurant@gmail.com),
started in Paris, December 2022, version March 21, 2024.

This book is dedicated to James H. Wilkinson (1919-1986) and Gene H. Golub (1932-2007)

L'inconnu de la vie des êtres est comme celui de la nature, que chaque découverte scientifique ne fait que reculer mais n'annule pas.

The unknown element in the lives of other people is like that of nature, which each new scientific discovery merely reduces but does not abolish.

– Marcel Proust, A la Recherche du Temps Perdu

Contents

Preface

Solving linear systems of equations is ubiquitous in scientific computing. Therefore, numerical algorithms for solving them are of paramount importance. There are two main kinds of methods for solving nonsingular linear systems. Direct methods are designed to obtain the solution after a finite number of basic operations, additions, subtractions, multiplications and divisions. Iterative methods define iterates that are supposed to converge in a certain sense to the solution.

This book covers both direct and iterative methods for solving nonsingular linear systems of equations. Very large sparse systems are now solved due to the progresses of numerical simulation and computers. Many methods have been invented over the years, particularly since the advent of the first digital computers during and after World War II. Some of the older methods proposed in the 1950s and 1960s are not so efficient anymore when applied to very large systems and there is still a very active research in this area. This book covers what we think are the most important and efficient algorithms known today.

The first chapter recalls some mathematical results and tools that are needed in the next chapters. We gather some definitions and put a particular emphasis on floating-point arithmetic and rounding errors in numerical computations.

The direct methods we consider are different versions of Gaussian elimination. Chapter 2 is devoted to Gaussian elimination for general linear systems. We describe several pivoting strategies and some variants designed for modern computers. We also consider methods which are not often described in books on direct methods: the Gauss-Jordan, Gauss-Huard and Purcell algorithms.

The topic of Chapter 3 is Gaussian elimination for sparse systems, that is, with matrices having many zero entries. We describe several ordering techniques designed to reduce the number of floating-point operations and the computer storage, as well as techniques designed to improve the efficiency on parallel computers like supernodal and multifrontal methods.

Classical iterative methods are recalled in Chapter 4. This means methods that have been developed in the 19th century and used on the computers of the 1950s and 1960s. We study Jacobi, Gauss-Seidel, SOR, and SSOR methods, as well as Richardson and alternating direction methods. Even though there are not really efficient for solving the problems people solve today, they are still of interest for constructing preconditioners.

In Chapter 5 the emphasis is put on the conjugate gradient (CG) method which is probably the most efficient method for solving sparse symmetric positive definite linear systems. We show how to cheaply compute bounds or estimates of error norms during the iterations. We also consider a few methods for symmetric indefinite problems.

Chapter 6 consider Krylov methods for nonsymmetric matrices with a particular interest for GMRES and methods using short recurrences, derived from the nonsymmetric Lanczos algorithm, that is, BiCG, BiCGStab, QMR and some variants. We also consider IDR methods.

An important issue for using Krylov methods efficiently is preconditioning. Therefore, in Chapter 7 we describe the most efficient preconditioners, mainly for symmetric problems.

This book should be useful for engineers and scientists solving linear systems as well as graduate students interested in numerical computations.

1
The tools of the trade

In this chapter, for the convenience of the reader, we recall some definitions and prove some well-known fundamental theorems on matrix properties that will be used in the following chapters. More details can be found in the classical books (in alphabetical order) by O. Axelsson [64], A. Berman and R.J. Plemmons [123], G.E. Forsythe and C.B. Moler [461], G.H. Golub and C. Van Loan [548], A.S. Householder [647], G. Strang [1054], G.W. Stewart [1040, 1045, 1046], R.S. Varga [1098], J.H. Wilkinson [1120], and D.M. Young [1144], as well as [811]. We also recall some facts about computer arithmetic that will be useful in studying rounding errors. More on that topic can be found in the books by N.J. Higham [631, 633], F. Chatelin and V. Frayssé [215], and the handbook [846]. We give a few examples of linear systems arising from discretization of partial differential equations which can be solved with the numerical methods that are considered in this book.

There are two main classes of algorithms to compute the solution of a linear system, direct methods and iterative methods. Mathematically, direct methods obtain the solution after a finite number of floating-point operations by doing combinations and modifications of the given equations. Of course, since on a digital computer the result of floating-point operations can only be obtained to a certain given precision, the computed solution is generally different from the exact solution, even with a direct method.

Iterative methods define a sequence of approximations that are expected to be closer and closer to the exact solution in some given norm, stopping the iterations using some predefined criterion, and obtaining a vector which is only an approximation of the solution.

Without being explicitly stated, we will consider square matrices A of order n with real coefficients that we denote by $a_{i,j}$. Most of the time we will denote matrices by capital letters, vectors by roman letters and scalars by Greek letters. Components of a vector x are denoted by x_i. Generally, elements of a vector sequence (iterates) will be denoted by x_k. To avoid confusion, when needed, their components are denoted as $[x_k]_i$.

1.1 ▪ Discretization methods for partial differential equations

Linear systems of equations are ubiquitous in scientific computing. Although not the only source of linear systems, problems that arise from discretization of elliptic and parabolic (systems of) partial differential equations (PDEs) are some of the most important ones. Generally, nonlinear problems are handled with iterative algorithms like Newton's method which gives a linear system to solve at each nonlinear iteration. Linear system solves are also involved in optimization

algorithms.

There are several ways to discretize PDEs: finite difference methods, finite volume methods, and finite element methods. Finite volumes can usually be interpreted as finite elements. These kinds of methods have many things in common, and finite differences can often be seen as particular cases of finite element methods. Nowadays, finite elements methods are the most used since scientists and engineers solve problems on complicated geometries and finite difference methods are not well suited for these problems.

Nevertheless, let us start with finite differences because they give model problems which are easy to construct. Most often, finite differences are used on a regular (cartesian) mesh and yield matrices with a regular nonzero structure that are easy to store in the computer memory. The model problem that was and still is widely used for testing algorithms is the Poisson equation on the open unit square $\Omega = (0,1) \times (0,1)$,

$$\begin{aligned} -\Delta u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= f \quad \text{in} \quad \Omega, \\ u\big|_{\partial\Omega} &= 0, \end{aligned}$$

where $\partial\Omega$ is the boundary of Ω and f is given in an appropriate functional space (for instance $L_2(\Omega) = \left\{u \middle| u \text{ measurable and } \int_\Omega u^2 dx < +\infty\right\}$). Non-homogeneous boundary conditions can be easily handled as well. To compute an approximate solution of this infinite dimensional problem we cover Ω with a regular mesh Ω_h having m points in each direction. This gives a mesh size $h = \frac{1}{m+1}$ as in Figure 1.1.

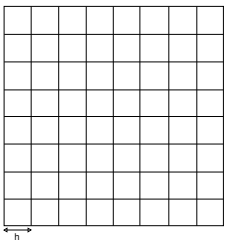


Figure 1.1. *A regular cartesian mesh*

We want to compute an approximate solution at each point (x_i, y_j) inside Ω since we already know u on the boundary (for i and j equal 0 or $m+1$). We denote $u_{i,j}$ the approximation of u at (x_i, y_j). Then we approximate the partial derivative $\frac{\partial u}{\partial x}$ at $[(ih + \frac{h}{2}), jh]$ by the finite difference

$$\left(\frac{\partial u}{\partial x}\right)_{i+\frac{1}{2},j} \approx \frac{u_{i+1,j} - u_{i,j}}{h}.$$

For the second order derivative,

$$\begin{aligned} \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} &\approx \frac{1}{h}\left(\left(\frac{\partial u}{\partial x}\right)_{i+\frac{1}{2},j} - \left(\frac{\partial u}{\partial x}\right)_{i-\frac{1}{2},j}\right), \\ &\approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}. \end{aligned}$$

Doing this for both directions, we obtain an approximation of minus the Laplacian Δ that we denote (after multiplication by h^2) by $-\Delta_5$.

$$\begin{aligned} (-\Delta_5 u)_{i,j} &= -u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1}, \\ &= h^2 f(x_i, y_j), \qquad i = 1, \ldots, m, \quad j = 1, \ldots, m. \end{aligned}$$

The solution is known for points on $\partial\Omega$, that is, for $i = 0$ or $m+1$, $j = 0$ or $m+1$. We have m^2 equations with m^2 unknowns and a linear system to solve. If we number the points from left to right and from bottom to top and rename the unknowns from u to x to agree with the notation that is traditionally used for linear algebra, the system can be written as

$$Ax = b$$

where

$$x = \{u_{1,1}, u_{1,2}, \ldots, u_{m,m}\}^T, \quad b = h^2\{f_{1,1}, f_{1,2}, \ldots, f_{m,m}\}^T,$$

$f_{i,j}$ being the value of f at point (i, j). The matrix A can be written in block form as

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix},$$

where I is the identity matrix and T is a tridiagonal matrix, both of order m,

$$T = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix}.$$

Of course, if we would have used other orderings for the mesh points, the structure of the matrix would have been different. We must warn the reader that this problem is relatively easy to solve, and therefore not a very good test problem for comparing algorithms, particularly for iterative methods.

It can be shown that $u_{i,j} \to u(x_i, y_j)$ in some sense when $h \to 0$. In this example, the matrix coefficients are integers. However, a linear system with the same nonzero structure is obtained when solving a diffusion problem with variable (and possibly discontinuous) coefficients like

$$-\frac{\partial}{\partial x}\left[\lambda_1(x,y)\frac{\partial u}{\partial x}\right] - \frac{\partial}{\partial y}\left[\lambda_2(x,y)\frac{\partial u}{\partial y}\right] = f(x,y),$$
$$u\big|_{\partial\Omega} = 0.$$

For this problem we obtain a linear system $Ax = b$ where the matrix can be written blockwise as

$$A = \begin{pmatrix} D_1 & A_2^T & & & \\ A_2 & D_2 & A_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & A_{n-1} & D_{n-1} & A_n^T \\ & & & A_n & D_n \end{pmatrix}.$$

With a 5-point discretization stencil as above the equation for the point (i, j) involves only the unknowns at points (i, j), $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ and $(i, j+1)$. The matrices D_i are tridiagonal and the matrices A_i are diagonal. Other types of boundary conditions give rise to matrices that are only slightly different. Let us look at the modifications given by Neumann or periodic boundary conditions on $\partial\Omega$, $\Omega = (0, 1) \times (0, 1)$ for the problem $-\Delta u + \sigma u = f$ where $\sigma > 0$ is a given coefficient.

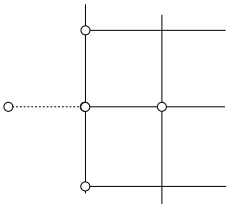


Figure 1.2. *Shadow unknowns for Neumann boundary conditions*

With Neumann boundary conditions $\frac{\partial u}{\partial n} = 0$, n being the outward normal to the boundary, we have $m+2$ unknowns for each mesh line, that is, $(m+2)^2$ unknowns in total. Difference equations on the boundary are obtained using "shadow" unknowns as in Figure 1.2.

We set $u_{i,-1} = u_{i,1}$ for handling $\frac{\partial u}{\partial n} = 0$. Hence, the equation for the point $(i,0)$ becomes

$$-u_{i-1,0} - u_{i+1,0} + (4+\sigma h^2)u_{i,0} - 2u_{i,1} = h^2 f_{i,0}.$$

Putting together the results for the whole mesh, we obtain

$$A = \begin{pmatrix} T & -2I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -2I & T \end{pmatrix},$$

where

$$T = \begin{pmatrix} 4+\sigma h^2 & -2 & & & \\ -1 & 4+\sigma h^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4+\sigma h^2 & -1 \\ & & & -2 & 4+\sigma h^2 \end{pmatrix}.$$

The matrix A is not symmetric but it can be symmetrized with a multiplication by a suitable diagonal matrix.

For periodic boundary conditions there are $(m+1)^2$ unknowns and the matrix (for $\sigma = 0$) is

$$A = \begin{pmatrix} T & -I & & & -I \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ -I & & & -I & T \end{pmatrix},$$

where

$$T = \begin{pmatrix} 4 & -1 & & & -1 \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ -1 & & & -1 & 4 \end{pmatrix}.$$

We observe that the matrix A is singular. This is linked to the fact that the solution of the continuous problem is not unique. It is often important to make sure that $b \in \text{Range}(A)$.

Parabolic problems solved with time implicit discretization schemes are usually a little easier to solve than elliptic problems. At each time step, we have to solve a linear system that looks like an elliptic problem. However, due to the discretization of the time derivative, there is an additional term (typically $h^2/\Delta t$ where Δt is the time step) added to the diagonal of the matrix giving better numerical properties. Three dimensional problems also give matrices with the same pattern as for two dimensional problems but then, the matrices D_i are themselves block tridiagonal.

Convection-diffusion problems are an interesting source of nonsymmetric linear systems. A typical problem is

$$\begin{aligned} -\epsilon\Delta u + 2P_x\frac{\partial u}{\partial x} + 2P_y\frac{\partial u}{\partial y} &= f \quad \text{in} \quad \Omega, \\ u\big|_{\partial\Omega} &= 0. \end{aligned}$$

The first order derivatives are approximated with centered or upwind finite differences.

The matrices arising from these finite difference methods have many zero entries. They are called *sparse matrices*. Dealing with sparse matrices, our aim is to be able to avoid storing the zero entries of the matrix A and to avoid doing operations on these zeros. Finite differences matrices can be conveniently stored by diagonals, that is, a diagonal is stored in a one dimensional array. At least, this is true if one wants to use an iterative method to solve the linear system , in which case only matrix-vector products are needed.

Another way to solve elliptic and parabolic PDEs is to use the finite element method. There is a variety of such methods, see [264]. We now give the flavor of the simplest one. Let Ω be a two dimensional domain and suppose we want to solve again the Poisson model problem,

$$-\Delta u = f \quad \text{in} \quad \Omega, \quad u\big|_{\partial\Omega} = 0,$$

with f in $L^2(\Omega)$.

Using Green's formula, one can see that u is the solution of the so-called variational problem,

$$a(u,v) = (f,v) \quad \text{for all } v \in H_0^1(\Omega),$$

where

$$H_0^1(\Omega) = \left\{ u \,|\, u \in L^2(\Omega),\, \frac{\partial u}{\partial x_i} \in L^2(\Omega),\, u\big|_{\partial\Omega} = 0 \right\},$$

and $a(u,v)$ is the bilinear form

$$a(u,v) = \int_\Omega \nabla u \nabla v \, dx, \quad (f,v) = \int_\Omega f v \, dx,$$

where ∇ is the gradient operator. To approximate the solution u, we define u_h as the solution of

$$a(u_h, v_h) = (f, v_h) \quad \forall v_h \in V_h,$$

where $V_h \subset H_0^1(\Omega)$ is a finite dimensional subspace. For example, one is given a triangularization of Ω (in our simple case Ω is supposed to be a polygon) as in Figure 1.3.

In the simplest method V_h is the set of continuous and piecewise polynomials of degree one on each triangle. The unknowns are taken to be the values of u_h at each vertex of the mesh. A basis for V_h is given by the functions $w_i \in V_h$ whose value is 1 at node i and 0 outside the triangles to which i is a vertex of. Then, the discrete problem is

$$a(u_h, w_i) = (f, w_i) \quad \text{for all basis functions } w_i.$$

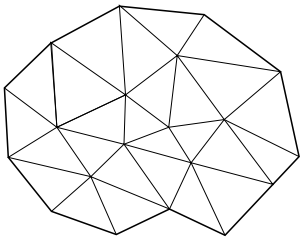


Figure 1.3. *A finite element triangular mesh in* $\mathbb{R}^2$

The approximate solution u_h can be decomposed on the basis functions as $u_h = \sum_j u_h(i)w_j$. Hence, we get a linear system whose unknowns are $u_h(i)$ with a matrix A for which $a_{i,j} = a(w_i, w_j)$. The nonzero structure of the matrix depends on the numbering scheme which is used for the mesh nodes. Generally these matrices cannot be stored by diagonals. The storage mode depends on the method (direct or iterative) that is used to solve the system and also on the type of computer that is used.

The most natural storage scheme is to store only the nonzero entries $a_{i,j}$ of A, together with the row and column indices i and j. If nz is the number of nonzeros of A, the storage needed is nz floating-point numbers and $2\,nz$ integers. However, this mode of storage (which is sometimes called the *coordinate scheme*) is not very convenient for direct methods as they usually require easy access to rows and/or columns of the matrix.

One common way (called *CSR*) to store a sparse matrix, more suited to direct methods, is to hold the nonzero entries of each row (resp. column) as a packed sparse vector `AA`, together with the column (resp. row) index of each element in a vector `JA`. These two vectors have a length of nz words. A third vector `IA` of integers of length $n+1$ is needed for pointing to the beginning of each row (or column) in `AA` and `JA`. Let us look at this storage scheme on a small example. Let

$$A = \begin{pmatrix} a_1 & 0 & 0 & a_2 \\ a_3 & a_4 & a_5 & 0 \\ 0 & a_6 & a_7 & 0 \\ a_8 & 0 & 0 & a_9 \end{pmatrix},$$

the stored quantities are

	1	2	3	4	5	6	7	8	9
`AA`	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
`JA`	1	4	1	2	3	2	3	1	4
`IA`	1	3	6	8	10				

Note that `IA(n+1)=nz+1`. This allows to compute the length of row i as `IA(i+1)-IA(i)`. Equivalently, the lengths of rows can be stored in place of `IA`. In some codes, the diagonal elements are treated in a special way. Since they are often all nonzero, they are stored in an additional vector of length n or the diagonal entry of each row can be stored in the first (or last) position of the row. If the matrix is symmetric, only the lower or upper part is stored.

Another storage scheme is using *linked lists*. Each nonzero entry (together with the column index) has a pointer `IPA` to the location of the next element in the row. To be able to add or delete entries easily in the list, it is sometimes useful to have a second pointer to the location of the previous entry. Finally, we must know the beginning of the list for each row in a vector `IA`. Going back to our small example, we have the following storage when using only forward links.

Note that the entries could be stored in any order,

	1	2	3	4	5	6	7	8	9
AA	a_3	a_2	a_9	a_1	a_4	a_8	a_6	a_5	a_7
JA	1	4	4	1	2	1	2	3	3
IPA	5	0	0	2	8	3	9	0	0
IA	4	1	7	6					

A zero value in `IPA(.)` indicates the end of the list for the given row.

Later on, we will consider other schemes for storing sparse matrices that are more useful for some particular algorithms. There also exist storage schemes that are more suited to iterative methods, especially those which need only matrix-vector products.

1.2 ▪ Where are the errors coming from?

When we want to simulate a phenomenon with a computer, there are many sources of error which make that the result of the simulation can be different from the real phenomenon (if it can be observed or measured without too many errors). The following list is obviously not exhaustive.

• Errors from the model.

The chosen mathematical model for the physics (or chemistry, biology, or any other science) may not represent the reality in a sufficiently precise way since one has almost always to make approximations. For example, continuum mechanics may not be sufficient, the diffusion equations may not represent correctly the transfer of heat or radiation in some media, and so on. The chosen model depends on the level that the scientist is considering: microscopic, macroscopic or an intermediate between the two and it is not simple to use models at different scales together. This is generally called multi-layer physics.

• Approximation or truncation errors.

Generally, the model is posed in a space of infinite dimension. It can be, for example, a linear or nonlinear partial differential equation having a solution denoted u in a bounded domain of $\mathbb{R}^3$ or $\mathbb{R}^4$ if the problem depends on time, or even higher dimensions if the problem is posed in a phase space, like for the equations of the transport of certain types of particles. In many cases an analytical solution is not known or is out of reach. By discretization techniques we reduce the so-called "continuous problem" to a finite dimensional problem whose solution is denoted u_h. This can be done, as we have seen above, by calculating the solution only in a finite number of points of the domain and replacing the derivatives in the model by differential quotients. In general u_h is not equal to u and there is an error depending on how refined is the discretization.

• Errors in the solution.

In many nonlinear cases the finite dimensional problem must be solved with an iterative method (for example Newton's method) which can only compute an approximation $\hat{u}$ of the solution u_h of the discrete problem. This solution $\hat{u}$ can, itself, be computed only approximately by another iterative method to solve the linear system at each iteration of the nonlinear method.

• Errors in data.

Many sources of error or inaccuracy may exist in the data needed to solve the problem. For example, the definition of the geometry of the domain in which the problem is posed may not be accurate enough. Since all the underlying physics may not be necessarily contained in the model, many simulations call upon material laws which in fact simulate the physics at a smaller scale (for example equations of state, nuclear cross sections, and so on). They are themselves the result of computations or measurements that are subject to errors.

• Rounding errors

The numerical simulation is carried out on digital computers which cannot, by definition, represent all real numbers. This results in rounding errors during the conversion of the data and at each elementary operation during the calculation. These errors can sometimes lead, as we will see in the following, to totally erroneous results.

• Human and computer errors.

The person trying to solve the problem intervenes in many stages of this process. Mistakes are therefore possible, if not probable. By example, programming errors when the algorithm that should give the solution is translated in a high-level language (Fortran, C, C++,...) When, despite these errors, the program provides a result, it is not always easy to see that there was an error. Hence, the importance of verification and validation of the codes with respect to known solutions (in some simple cases), comparisons to other codes and to experimental results. More rarely, computers can produce wrong results temporarily (hardware or software failure) or permanently (see the design error of the Pentium IV processor).

Generally, solving problems in scientific computing is a complicated process that requires iterations with the different stages: modelization, discretization, numerical solution, verification and validation. If the validation does not give good results, one has to verify the code or sometimes improve the numerical methods, the discretization scheme or eventually the model itself. It may take several of such loops to obtain a satisfactory solution.

1.3 ▪ IEEE floating-point arithmetic

Most problems that we have to solve in scientific computing use real or complex numbers (defined as couples of real numbers). For instance, the coefficients and solution of linear systems are generally real or complex numbers. Unfortunately, for solving these systems we use digital computers in which all real (or complex) numbers cannot be exactly represented. In particular, registers and memory words designed to store the data and the intermediate and final results have a finite length or capacity and cannot store all real numbers. Moreover, when computations are performed on a computer, each arithmetic operation $(+, -, *, /)$ is generally affected by rounding errors as only a finite number of digits (or bits) can be retained in the result.

The subject of rounding error analysis is to try to understand the effects of these limitations on the result of solving a problem. Therefore, let us describe a few basic facts about floating-point arithmetic.

In day to day's life we use a positional base-10 number system. For instance,

$$(71.25)_{10} = 7 \times 10 + 1 \times 1 + 2 \times \frac{1}{10} + 5 \times \frac{1}{100}.$$

Today's scientific computers use a binary representation of real numbers, that is, the digits are equal to 0 or 1. In the past certain manufacturers, in particular IBM for its 360 series and for his successors used the hexadecimal system, that is, a representation in base 16. In binary, the reals are therefore decomposed into powers of 2,

$$71.25 = 2^6 + 2^2 + 2^1 + 2^0 + 2^{-2}.$$

Of course, not all real numbers have a finite binary representation, for example $1/10$ has a finite representation in decimal but infinite in binary. The rational numbers (quotients of integers) have finite or infinite representations with a repetition of a sequence of digits. Irrational numbers (for example π) have an infinite representation without repetition.

The numbers that can be represented in the computer are a (finite) subset of the real numbers. This set, denoted by F, is characterized by four integers: the base β, the number t of base-β

digits in the fractional part m (also called the mantissa or the significand) and the exponent range $[e_L, e_U]$,

$$f = \pm m \times \beta^E, \quad 1 \leq m < \beta.$$

Such numbers are called floating-point numbers. In a binary computer $\beta = 2$.

Computers developed in the 1960s and 1970s used a wide variety of formats to define and store floating-point numbers. The number of bits (that can store 0 or 1) contained in each memory word as well as the number of these bits allocated to the mantissa m or the exponent E were not standardized. For example, in the 1960s the IBM 704 had words of 36 bits, the IBM 7030 Stretch had 64 bits words (including 48 bits of mantissa), in the 1970s the Control Data 6400, 6600 and 7600 had 60-bit words. The CRAY machines in the 1980s had 64 bits words with 48 bits of mantissa. The arithmetic aberrations were legion on these machines.

The way in which the elementary operations $(+, -, *, /)$ were done also differed from one architecture to another. Up to recent times, all computers using base 2 implemented the IEEE 754 standard defined in 1985 by the Institute of Electrical and Electronics Engineers (IEEE) in a more or less strict way.

The genesis of this development began in 1976 when Intel decided to develop a floating co-processor (named 8087) for its 8086 processor. John Palmer from Intel wanted the arithmetic of the 8087 to be "the best possible". To achieve this goal, Intel asked William M. Kahan, professor at the University of California at Berkeley to act as a consultant. Kahan had already been very interested in the properties and anomalies of floating-point arithmetic. He was, moreover, a well-known researcher in the field of numerical analysis and scientific computing.

These attempts interested other manufacturers and led to the creation in 1977 of a committee (named IEEE p754) under the aegis of IEEE. The second meeting in November 1977 attracted a dozen people among whom representatives of National Semiconductor, Zilog, Motorola and IBM. The most prominent manufacturers of scientific computers at that time, that is, CDC and Cray, did not participated in this committee. After agreement from Intel, Kahan, his student J. Coonen, and H. Stone wrote a proposed standard known as KCS. Although Kahan was not authorized to disclose all the details of the 8087, the KCS document was heavily inspired by the work done for Intel.

There were many discussions and even arguments (or religious wars) during the lively meetings of this committee. Particularly with the representatives of DEC who, with the arithmetic of their VAX computers also had good arguments. One of the controversial parts of KCS was the introduction of denormalized numbers (that are now called subnormal numbers) to deal with underflow in a gradual way. Finally, a consensus eventually emerged around the main principles of KCS and the proposal was submitted to the IEEE, and was published in 1985. It was a 23-page document which defines the formats of the floating-point numbers and the constraints that a binary floating-point arithmetic must implement. Kahan received the ACM Turing award (Association for Computing Machinery) in 1989 for his contribution to the definition of the standard.

This American standard was adopted internationally in 1989 under the name IEC 60559. Another IEEE Std 854 standard was created in 1987. It defined a floating point arithmetic whatever the base. It is consistent with IEEE 754 when the base is 2. The IEEE 754 was updated in 2008 with the introduction of decimal floating-point arithmetic. It is a 58-page document. Another revision was done in 2019 with only minor changes.

The adoption of the IEEE standard by most computer manufacturers was a blessing for people developing algorithms and for programmers. In particular, as we will see, a standard model could be used in the rounding error analysis of linear algebra algorithms.

Let us now consider what is defined in IEEE 754. The standard asks for

- a consistent representation of floating-point numbers,

- floating-point operations with a result correctly rounded with several rounding modes,
- a consistent handling of exceptions (for instance, divisions by zero).

There is no requirement on the way integers are represented and stored. The standard describes formats for storing floating-point numbers and constraints on operations but not how this must be implemented. Moreover, this can be done by hardware or software. For example, on a machine with only the single format, one can implement the support of the double format in software.

As we said above floating-point numbers (or floats for short) are written as $f = \pm m \times 2^E$. The binary significand is

$$m = d_0.d_1d_2\cdots, \quad d_i = 0 \text{ or } 1.$$

If $d_0 = 1$ the float is said to be normalized and the corresponding bit is not stored. This allows to have one more significant bit in the mantissa. d_0 is called the hidden bit. Floats with $d_0 = 0$ are called subnormals and are identified by their exponent. A memory word is divided in three fields: the sign, the exponent, and the mantissa. The precision is the number p of bits in the mantissa, including the hidden bit.

The smallest normalized float larger than 1 is

$$(1.00\cdots01)_2 = 1 + 2^{1-p}.$$

The number 2^{1-p} is called the machine epsilon, and denoted as ϵ or ϵ_M.

The 1985 version of the standard defined three formats: single, double, and extended. The single format (also known as binary32 or fp32) uses 32 bits with one sign bit, 8 bits for the exponent, and 23 bits for the mantissa (not including the hidden bit) which gives $p = 24$. The number 0 has a special representation as well as for $-\infty$, $+\infty$ and NaN (Not a Number). The exponent is biased by 127. This means that what is stored in the 8 bits of the exponent is the binary representation of $E + 127$. If the sign bit is 0 (resp. 1) the float is positive (resp. negative). If the float is denoted as

$\pm$	$a_1a_2\cdots a_8$	$d_1d_2\cdots d_{23}$

Table 1.1 shows representations of normalized single floats (except zero). The first row shows the representation of 0, and the last one shows $\pm\infty$ and NaN depending on the bits in the mantissa. There exist $+0$ and -0. For the non-biased exponent the minimum and maximum values are $E_{\min} = -126$ and $E_{\max} = 127$. The smallest normalized number is $N_{\min} = 2^{-126} \approx 1.17 \times 10^{-38}$, and the largest one is $N_{\max} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}$. The machine epsilon is $\epsilon_M \approx 1.1921\ 10^{-7}$.

Subnormal numbers use the particular exponent -126 (corresponding to $(0\cdots0)_2$ in binary after bias and shift. For instance, $2^{-149} \approx 1.4 \times 10^{-45}$ is

0	00000000	00000000000000000000001

It is the smallest positive number that can be used. Subnormal numbers are less accurate than normalized ones since they have less significant bits.

The double format (also known as binary64 or fp64) follows the same principles. It uses 64 bits with one sign bit, 11 bits for the exponent, and 52 bits for the mantissa ($p = 53$). The exponent bias is 1023. Therefore, $N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308}$ and $N_{\max} = (2 - 2^{-52}) \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}$. The smallest subnormal positive number is $2^{-1074} \approx 4.9 \times 10^{-324}$. The machine epsilon is $\epsilon_M \approx 2.2204\ 10^{-16}$. Characteristics of double floats are shown in Table 1.2.

Table 1.1. *IEEE 754 single format*

$a_1a_2\cdots a_8$	value
$(00000000)_2 = (0)_{10}$	$\pm(0.d_1d_2\cdots d_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.d_1d_2\cdots d_{23})_2 \times 2^{-126}$
$(00000011)_2 = (2)_{10}$	$\pm(1.d_1d_2\cdots d_{23})_2 \times 2^{-125}$
$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$\pm(1.d_1d_2\cdots d_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.d_1d_2\cdots d_{23})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111110)_2 = (254)_{10}$	$\pm(1.d_1d_2\cdots d_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ if $d_1 = \cdots = d_{23} = 0$, NaN otherwise

Table 1.2. *IEEE 754 double format*

$a_1a_2\cdots a_{11}$	value
$(00000000000)_2 = (0)_{10}$	$\pm(0.d_1d_2\cdots d_{52})_2 \times 2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$\pm(1.d_1d_2\cdots d_{52})_2 \times 2^{-1022}$
$(00000000011)_2 = (2)_{10}$	$\pm(1.d_1d_2\cdots d_{52})_2 \times 2^{-1021}$
$\vdots$	$\vdots$
$(01111111111)_2 = (1023)_{10}$	$\pm(1.d_1d_2\cdots d_{52})_2 \times 2^{0}$
$(10000000000)_2 = (1024)_{10}$	$\pm(1.d_1d_2\cdots d_{52})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111111110)_2 = (2046)_{10}$	$\pm(1.d_1d_2\cdots d_{52})_2 \times 2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$\pm\infty$ if $d_1 = \cdots = d_{52} = 0$, NaN otherwise

The 1985 standard suggested an extended format with at least 15 bits for the exponent and at least 53 bits for the mantissa. This was used in some Intel microprocessors for the floating-point registers. Therefore, the basic operations were done more accurately than in double precision before rounding to double.

The 2008 revision of the standard introduced a quadruple format (also known as binary128 or fp128). As the name implies it uses 128 bits with one sign bit, 15 bits for the exponent, and 112 bits for the mantissa ($p = 113$). The exponent bias is 16383. Therefore, $N_{\min} = 2^{-16382} \approx 3.4 \times 10^{-4932}$ and $N_{\max} = (2 - 2^{-128}) \times 2^{16383} \approx 2^{16384} \approx 1.2 \times 10^{4932}$. Unfortunately most common microprocessors do not have an hardware implementation of fp128.

An interchange half precision format (also known as binary16 or fp16) was also defined in that revision. It uses 16 bits with one sign bit, 5 bits for the exponent, and 10 bits for the mantissa ($p = 11$). The exponent bias is 15. Therefore, $N_{\min} = 2^{-14} \approx 6.1 \times 10^{-5}$ and $N_{\max} = (2 - 2^{-10}) \times 2^{15} = 65504$. The smallest subnormal number is $5.9605\ 10^{-8}$ and the machine epsilon is $\epsilon_M = 9.7656250\ 10^{-4}$. The fp16 format is now used in specialized hardware and software for neural networks and machine learning.

The IEEE standard also prescribes how rounding must be done. Floats are the numbers which can be represented in the chosen format. A real number x is in the normalization interval if $N_{\min} \leq |x| \leq N_{\max}$. If a real x is not a float, either x is not in this interval or the precision p is

too small to exactly represent x. In both cases x has to be approximated by something else. Let x_- and x_+ be the two closest floats below and above x. If x is larger than $N_{\max}$, $x_- = N_{\max}$ and $x_+ = \infty$. If x is positive and smaller than $N_{\min}$, then x_- is subnormal or 0 and x_+ is subnormal or $N_{\min}$. If x is negative we have the symmetric of this situation. The rounding modes are defined as follows with a result $f = fl(x)$,

- if x is a float, $f = x$,
- *round down* (towards $-\infty$), closest float to and no greater than the infinitely precise result, $f = x_-$,
- *round up* (towards $+\infty$), closest float to and no less than the infinitely precise result, $f = x_+$,
- *round towards zero*, closest float to and no greater in magnitude than the infinitely precise result, $f = x_-$ if $x \geq 0$ and x_+ if $x \leq 0$,
- *round to nearest*, f is from x_- and x_+ the closest to x. If there is a tie,
  - (roundTiesToEven), the one with an even least significant digit,
  - (roundTiesToAway), the one with larger magnitude.

Generally, roundTiesToEven is the default rounding mode. Every operation $(+, -, *, /, \sqrt{})$ must be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to the rounding mode. The standard also considered a fused multiply-add (FMA) operation computing $x + (y * z)$ with only one rounding at the end. If we denote $\oplus, \ominus, \otimes, \oslash$ the hardware realizations of the arithmetic operations $+$, $-$, $*$, $/$, if x and y are two floats we must have

$$x \oplus y = fl(x + y), \;\; x \ominus y = fl(x - y),$$
$$x \otimes y = fl(x * y), \;\; x \oslash y = fl(x/y).$$

These requirements can be satisfied if the processor uses guard digits. Let us, for instance, add 3 and 3×2^{-23} in single precision. The mantissa of the second operand must be shifted to obtain the same exponent,

$$\begin{array}{cll} & (1.10000000000000000000000 &)_2 \times 2^1 \\ + & (0.00000000000000000000001|1 &)_2 \times 2^1 \\ = & (1.10000000000000000000001|1 &)_2 \times 2^1 \end{array}$$

If the bit after the bar in the second operand is not kept, the result cannot be rounded correctly. Using only one guard digit is not always enough. If we subtract $(1.0001)_2 \times 2^{-25}$ from 1,

$$\begin{array}{cll} & (1.00000000000000000000000 &)_2 \times 2^0 \\ - & (0.00000000000000000000000|010001 &)_2 \times 2^0 \\ = & (0.11111111111111111111111|101111 &)_2 \times 2^0 \\ \mathrm{n} & (1.11111111111111111111111|011110 &)_2 \times 2^{-1} \end{array}$$

The last step is the normalization. Fortunately, it is not necessary to have a large number of guard digits. It is enough to have two guard digits if we add to them another bit (called the sticky bit) which is 1 to indicate that we have removed some bits equal to 1 to the right of the guard digits. The previous example becomes

$$\begin{array}{cll} & (1.00000000000000000000000 &)_2 \times 2^0 \\ - & (0.00000000000000000000000|011 &)_2 \times 2^0 \\ = & (0.11111111111111111111111|101 &)_2 \times 2^0 \\ \mathrm{n} & (1.11111111111111111111111|01 &)_2 \times 2^{-1} \end{array}$$

Operations which are mathematically well defined must give the correct result, $x-\infty = -\infty$, if $x > 0$ $x/0 = \infty$, $x * \infty = \infty$, $\infty + \infty = \infty$. Results without mathematical meaning like $0/0$, ∞/∞, $\infty - \infty$ must give a NaN as a result. An operation with a NaN operand must give a NaN.

1.4 ▪ Properties of the IEEE standard and examples

If x is a real number different from zero, let the relative rounding error be

$$\delta = \frac{fl(x) - x}{x}.$$

If $fl(x)$ has an exponent E and x is in the normalization interval, $|x| \geq 2^E$ and

$$|\delta| < \frac{2^{1-p} \times 2^E}{2^E} = \epsilon_M,$$

whatever is the rounding mode. If the rounding mode is "round to nearest", we have a better bound,

$$|\delta| < \frac{2^{-p} \times 2^E}{2^E} = \frac{\epsilon_M}{2}.$$

The quantity $u = \epsilon_M/2$ is called the unit roundoff. It is $1.110223024625157\ 10^{-16}$ for double precision. In all cases we have

$$x \circ y = (x \bullet y)(1 + \delta), \quad \circ = \oplus, \ominus, \otimes, \oslash, \quad \bullet = +, -, *, /,$$

with $|\delta| < \epsilon_M$ or $|\delta| < u$ if rounding to nearest is used. The upper bound on $|\delta|$ can be attained as one can see in the following example,

```
>> u=eps/2
u = 1.110223024625157e-016
>> up6 = 6+u
up6 = 6
>>
```

The difference is $u = \epsilon_M/2$. However, in some cases, the bound is pessimistic. For example, if

```
>> x=1/2-1/16
x = 4.375000000000000e-01
```

and the result is exact. Of course, in this example $1/2$ and $1/16$ are exactly represented. An interesting result about subtraction due to P.H. Sterbenz [1038] is the following.

Theorem 1.1. *Let x and y be two normalized floating-point numbers whose exponents differ at most by 1. Then, $fl(x - y) = x - y$.*

Proof. See [633]. □

A useful corollary is obtained from Theorem 1.1.

Corollary 1.2. *Let x and y be two normalized floating-point numbers such that $y/2 \leq x \leq 2y$. Then, $fl(x - y) = x - y$.*

Let us consider an example with $x = fl(1/3)$ and $y = fl(1/5)$. The mathematical result is

$$\frac{1}{3} - \frac{1}{5} = \frac{1}{10}\left(1 + \frac{1}{3}\right) = 0.13333333333333\ldots$$

The binary representation of $x = 1/3$ is

0	01111111101	0101010101010101010101010101010101010101010101010101010101010101

The unbiased exponent is -2. The representation of $y = 1/5$ is

0	01111111100	1001100110011001100110011001100110011001100110011001100110011010

The unbiased exponent is -3. The result of the difference of these two floats is

0	01111111100	0001000100010001000100010001000100010001000100010001000100010000

The hypotheses of Theorem 1.1 are satisfied and the result is obtained without rounding. Converted to decimal, the binary fractional part is $6.666666666666643\ 10^{-2}$. We have to add 1 and to multiply by 2^{-3}. When converted to decimal the binary result is $1.333333333333333\ 10^{-1}$ with 16 correct decimal digits.

The elements of F are not regularly distributed on the real line; see Figure 1.4 that shows the elements for a toy floating-point format with only three bits in the mantissa.

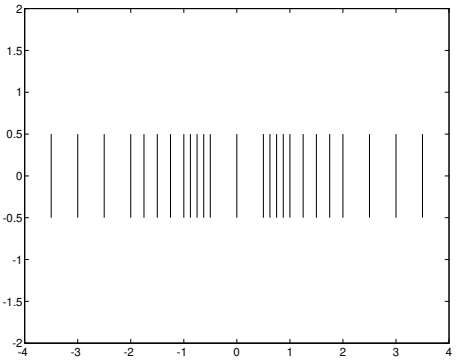


Figure 1.4. *Floating-point numbers with three digits in the mantissa*

Let us now consider some interesting questions:

- How many floats do we have?

	Normalized	Subnormal
fp32	$4.3\ 10^{9}$	$1.7\ 10^{7}$
fp64	$1.8\ 10^{19}$	$9\ 10^{15}$

- How many fp64 floats do we have in between two fp32 floats?

  The answer is the same for every pair of normalized fp32 floats. Hence, we can look in between 1 and $1 + 2^{-23}$. In that interval the distance between two fp64 floats is 2^{-52}. Hence, the answer is $2^{29} - 1 \approx 5.4\ 10^{8}$

- What is the largest integer m such that all the integers in $[-m, m]$ are represented exactly in fp64?

  The answer is $m = 2^{53}$. Therefore, the largest integer that can be exactly represented is $2^{53} \approx 9.01\ 10^{15}$.

The IEEE 754 arithmetic has the following nice properties,

- $fl(x \circ y) = fl(y \circ x),\ \circ = +, *$
- $fl(x - y) = -fl(y - x)$ if the rounding mode is not towards $\pm\infty$
- $fl(x + x) = fl(2 * x)$
- $fl(0.5 * x) = fl(x/2)$
- $x \leq fl((x + y)/2) \leq y$ if $x \leq y$

Unfortunately it does not have $fl((x + y) + z) = fl(x + (y + z))$, which means that addition is commutative but not associative. This is a problem when summing several floats since the answer may depend on the order of the operations. For instance,

```
>> pi+1e16-1e16
4
>> pi+(1e16-1e16)
3.1416e+00
```

In the first case π is almost lost in the first addition whose binary representation of the result is

0	10000110100	0001110000110111100100110111111000001000000000000010

whence 10^{16} is

0	10000110100	0001110000110111100100110111111000001000000000000000

1.5 ▪ Other arithmetic formats

Half precision refers to arithmetics using words with 16 bits. There are only 5 bits for the exponent in `fp16`. Hence, the range of numbers that can be represented is rather limited, even though `fp16` allows subnormal numbers. This is why another (non standard) format named `bfloat16` was promoted by Google and Intel [664]. Their characteristics are shown in Table 1.3 in which *Nmin* (resp. *Nmax*) stands for the smallest (resp. largest) normalized number. The format `bfloat16` has three more bits for the exponent which allows larger or smaller numbers, but, of course, there are less bits for the significand and this limits the accuracy of the computations.

Table 1.3. *Parameters for* `bfloat16` *and* `fp16`

format	sign	signif.	expo.	min. subnormal	Nmin	Nmax
`bfloat16`	1	7	8	-	$1.18\ 10^{-38}$	$3.39\ 10^{38}$
`fp16`	1	10	5	$5.96\ 10^{-8}$	$6.10\ 10^{-5}$	$6.55\ 10^{5}$

In the past and still now, some people would like to get rid of the IEEE 754 floating point arithmetic IEEE 754 standard, arguing that they can obtain the same or even better accuracy while storing a smaller amounts of bits. For instance, see the logarithmic arithmetic [705] in the

1970s and tapered floating point arithmetic [844] also in the 1970s, a format similar to floating point, but with variable-sized entries for the mantissa and exponent. The last avatar is called unums and posits (which are unums of type III); see [583, 584]. Wether this is good or bad is a debate. For an analysis of the positive and negative aspects of posits, see [323].

1.6 ▪ Misconceptions about rounding errors

In this section we discuss examples about some common misconceptions about rounding errors,

1- Subtraction operations are always bad,

2- Rounding errors are only important if they accumulate during a large number of successive operations,

3- A computation with only few operations without underflow or overflow is always accurate,

4- Increasing the precision, that is, the number of bits of the mantissa, always improves the accuracy,

5- The final accuracy of a computation cannot be more accurate than that of intermediate results,

6- Rounding errors are always a bad thing.

1.6.1 ▪ Subtraction cancellation

Let us first consider an example showing that assertion 1 is sometimes true. We would like to compute

$$f(x) = \frac{1 - \cos(x)}{x^2},$$

for $x = 1.25\ 10^{-5}$. Using the Taylor expansion,

$$\cos(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + O(x^8),$$

and

$$\frac{1 - \cos(x)}{x^2} = \frac{1}{2} - \frac{x^2}{24} + \frac{x^4}{720} + O(x^6).$$

and the correct answer is clearly close to $1/2$. In fact, using variable precision in Matlab[1] with 64 decimal digits we obtain,

```
>> digits(64)
>> xx=vpa(1.25e-5)
0.0000125

>> cc=cos(xx)
0.99999999992187500000101725260416136847601998006067578752161327 23

>> yy=vpa(1)-cc
0.00000000007812499999898274739583863152398001993932421247838672769550198716

>> zz=yy/(xx*xx)
0.49999999999348958333336724175347212761167495986167505725121271 78
```

[1] Matlab is a trademark of The MathWorks.

With the fp64 (double precision) format we obtain

```
>> x=1.25e-5
1.250000000000000e-05
>> (1-cos(x))/x^2
4.999996860988176e-01
```

The relative difference with the variable precision result is $6.277893\ 10^{-7}$. This is far from being proportional to the unit roundoff. The cosinus computed with fp64 has 15 correct decimal digits but $1 - \cos(x)$ has only 6 correct decimal digits. Things can be worse with a lower precision. Using fp32 (single precision) format the value of $\cos(1.25\ 10^{-5})$ is 1 and the numerator of f is 0. This can be understood by looking at the Taylor expansion of the cosine. The term $x^2/2$ is $7.8124993\ 10^{-11}$ with fp64 as well as in fp32. But, in single precision (fp32), this number has a binary representation

0	01011101	01010111100110001110110

The unbiased exponent is -34 and when a shift is done to subtract from 1, all the significant digits are lost and the result is 1. More mantissa bits are needed to obtain a nonzero result for the numerator of $f(x)$. In fact, with 8 bits of exponent we need at least 32 bits in the mantissa to obtain a cosine different from 1.

However, $f(x)$ can be computed in a different way. Using the relation $\cos(x) = 1 - 2\sin^2(x/2)$, we can write

$$f(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

In double precision we obtain

```
>> x=1.25e-5
1.250000000000000e-05
>> sx=2*sin(x/2)/x
9.999999999934897e-01
>> f=sx^2/2
4.999999999934897e-01
```

Only the rightmost decimal digit is different from the exact result. In single precision the value of the function is 1.

However, we have seen in Theorem 1.1 and Corollary 1.2 that subtraction of floating-point numbers can be exact if they are close enough. Hence, assertion 1 is not always true.

1.6.2 ▪ Accumulation of rounding errors

We are concerned with assertions 2 and 3. We have already shown at the end of the previous section an example with only two operations and a wrong result. Let us consider some other examples.

Even though this is not a very good idea, we would like to compute $e = \exp(1)$ with

$$e = \lim_{n\to\infty}\left(1 + \frac{1}{n}\right)^n.$$

The limit in the right-hand side was first considered by the Swiss mathematician Jacob Bernoulli in 1683. He proved that the limit is between 2 and 3. The values of n we consider are $n = 10^m$, $m = 1, \ldots, 16$.

The value of the exponential computed in variable precision with 64 decimal digits is

```
e = 2.71828182845904523536028747135266249775724709369995957496696762 8
```

In fp64 we obtain

```
 m              f                       |f - e|
 1      2.593742460100002e+00     1.245393683590429e-01
 2      2.704813829421529e+00     1.346799903751675e-02
 3      2.716923932235594e+00     1.357896223451649e-03
 4      2.718145926824926e+00     1.359016341197286e-04
 5      2.718268237192298e+00     1.359126674774804e-05
 6      2.718280469095753e+00     1.359363291853169e-06
 7      2.718281694132082e+00     1.343269634748315e-07
 8      2.718281798347358e+00     3.011168751033943e-08
 9      2.718282052011560e+00     2.235525150215830e-07
10      2.718282053234788e+00     2.247757423188143e-07
11      2.718282053357110e+00     2.248980649153106e-07
12      2.718523496037238e+00     2.416675781925168e-04
13      2.716110034086901e+00     2.171794372144353e-03
14      2.716110034087023e+00     2.171794372022673e-03
15      3.035035206549262e+00     3.167533780902166e-01
16      1.000000000000000e+00     1.718281828459045e+00
```

When increasing m, the result improves until $m = 8$ (but we have only 8 correct decimal digits) and then it deteriorates until having no correct decimal digits for the last two values of m. In each of these computations, there are only one addition, one division and an exponentiation. We loose some significant digits when computing $1 + 1/n$. For example, for $m = 16$ this is equal to 1. This explains the results for the last values of m. However, the exponentiation involves many hidden operations through the logarithm and the exponential.

We can construct examples for which the results are not accurate because we do a large number of iterations. Let us consider the following code which sums small positive values. In this code `ps` is the accumulated value and `psc` is the correct result if n is even.

```
pi3 = pi / 3;
epsi = 10 * eps;
ps = 0.;
for i=1:2:n-1
 x(i) = pi3;
 x(i+1) = epsi;
 y(i) = epsi;
 y(i+1) = pi3;
 ps = ps + x(i) * y(i) + x(i+1) * y(i+1);
end
psc = n * pi3 * epsi;
```

The results using fp64 are

```
    n              ps                    |(ps - psc) / psc|
    1     2.325245665339088e-15              0
   10     2.325245665339088e-14     1.357036664091101e-16
  100     2.325245665339087e-13     2.171258662545761e-16
 1000     2.325245665339066e-12     9.553538115201348e-15
10000     2.325245665339668e-11     2.495731557076599e-13
```

```
   100000     2.325245665342595e-10     1.508555778598194e-12
  1000000     2.325245665361659e-09     9.706872749351716e-12
 10000000     2.325245664914360e-08     1.826594643824356e-10
```

The relative error is increasing with n. The results for fp32 (single precision) are given below. Of course, we have to replace eps by the single precision machine epsilon, that is 2^{-23}.

```
      n              ps       |(ps - psc) / psc|
      1     1.2483567e-06                0
     10     1.2483569e-05     7.2855350e-08
    100     1.2483573e-04     4.6627426e-07
   1000     1.2483563e-03     4.6627420e-07
  10000     1.2482489e-02     8.6465894e-05
 100000     1.2498596e-01     1.2038678e-03
1000000     1.2395369e+00     7.0652380e-03
10000000    1.1708032e+01     6.2124588e-02
```

For $n = 10000$ we have only 4 correct decimal digits. The last result has just one correct decimal digit.

We will come back to summation algorithms later and show that sums like the last one can be computed more accurately.

1.6.3 ▪ Increase of precision

In [633] N.J. Higham gave examples in which increasing the precision does not always improve the accuracy of the result. Let us consider the inverse A of the Hilbert matrix of order 5,

$$A = \begin{pmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{pmatrix}$$

and a right-hand side $b = [-1; 2; -3; 4; -5]$. The linear system $Ax = b$ is solved with Gaussian elimination without pivoting. For the data and the result of every operation, the mantissa is truncated to t digits. The relative error in the maximum norm as a function of t is shown in Figure 1.5. One can see that when t is increased, the accuracy is not always better.

1.6.4 ▪ Final accuracy

Let us consider two algorithms for computing $(e^x - 1)/x$ with x close to zero. When x is very small, the result must be close to 1. Algorithm 1 is the naive and straightforward way. In algorithm 2 we make a change of variable.

```
 Algo 1

for i=nmin:nmax
 x = 10^(-i);
 if x == 0
  f1(i) = 1;
 else
```

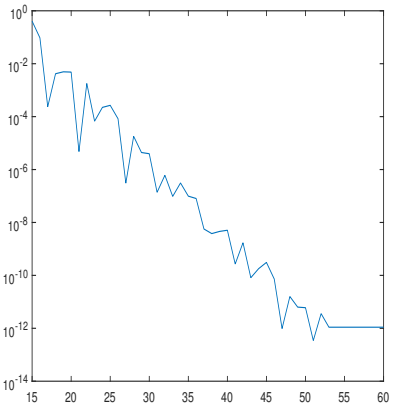


Figure 1.5. *Relative error as a function of the number of bits in the mantissa*

```
  f1(i) = (exp(x) - 1) / x;
 end
end

 Algo 2

for i=nmin:nmax
 x = 10^(-i);
 y = exp(x);
 if y == 1
  f2(i) = 1;
 else
  f2(i) =(y - 1) / log(y);
 end
end
```

We use $nmin = 5, nmax = 16$, and using fp64 we obtain

```
    i             Algo 1                    Algo 2

    5      1.000005000006965e+00     1.000005000016667e+00
    6      1.000000499962184e+00     1.000000500000167e+00
    7      1.000000049433680e+00     1.000000050000002e+00
    8      9.999999939225290e-01     1.000000005000000e+00
    9      1.000000082740371e+00     1.000000000500000e+00
   10      1.000000082740371e+00     1.000000000050000e+00
   11      1.000000082740371e+00     1.000000000005000e+00
   12      1.000088900582341e+00     1.000000000000500e+00
   13      9.992007221626409e-01     1.000000000000050e+00
   14      9.992007221626409e-01     1.000000000000005e+00
   15      1.110223024625157e+00     1.000000000000000e+00
   16                          0     1.000000000000000e+00
```

One can see that Algorithm 1 gives results which are not accurate. Let us consider the case

$i = 13$. In variable precision, we have

```
>> digits(32)
>> xx = vpa(1e-13)
0.0000000000001
>> ee = exp(xx)
1.0000000000001000000000000005
>> eex1 = ee - 1
0.0000000000001000000000000050000000000018957
>> eex1 / xx
1.0000000000000500000000000018957
```

We see that Algorithm 2 gives the correctly rounded result. In fp64 we obtain

```
>> x = 1e-13
1.000000000000000e-13
>> e = exp(x)
1.000000000000100e+00
>> ex1=e-1
9.992007221626409e-14
>> ex1/x
9.992007221626409e-01
```

The binary representation of the exponential e is

0	01111111111	0000000000000000000000000000000000000000000000111000010

When we subtract 1 from this, we just obtain the rightmost bits as leading bits and the result of the subtraction is far from being accurate. In variable precision, Algorithm 2 gives

```
>> yy = exp(xx)
1.0000000000001000000000000005
>> eey1 = yy - 1
0.0000000000001000000000000050000000000018957
>> lyy = log(yy)
0.0000000000001000000000000000000000000000229
>> eey1 / lyy
1.0000000000000500000000000016667
```

With fp64, we obtain

```
>> y = exp(x)
1.000000000000100e+00
>> ey1 = y - 1
9.992007221626409e-14
>> ly = log(y)
9.992007221625909e-14
>> ey1 / ly
1.000000000000050e+00
```

$y - 1$ and $\log(y)$ (whose exact value must be 10^{-13}) are not very accurate but their errors compensate and the result of the division is accurate even though the intermediate results are not.

1.6.5 ▪ Rounding errors are not random

Let us consider an example devised by W. Kahan and reported in [633]. We would like to compute the rational function

$$r(x) = \frac{(622 - x(751 - x(324 - x(59 - 4x))))}{(112 - x(151 - x(72 - x(14 - x))))},$$

for $x = 1.606 + (i-1)2^{-52}$, $i = 1, \ldots, 361$. The function r is almost constant on that interval with an average value 8.752376580778492. Figure 1.6 shows $100(r - 8.752376580778492)$. We clearly see that rounding errors are not random since many points are aligned.

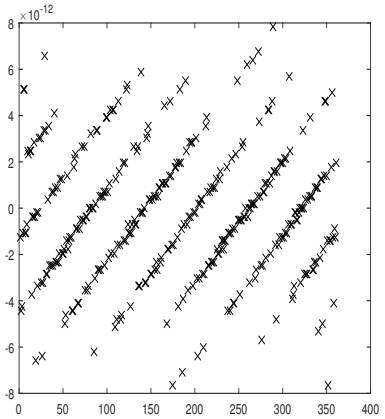


Figure 1.6. *Values of* $100(r - 8.752376580778492)$

1.6.6 ▪ Rounding errors can sometimes help (but not always)

Let us consider the singular matrix

$$A = \begin{pmatrix} 0.7 & -0.4 & -0.3 \\ -0.4 & 0.9 & -0.5 \\ -0.3 & -0.5 & 0.8 \end{pmatrix}.$$

The eigenvector corresponding to the zero eigenvalue is $e = (1 \quad 1 \quad 1)^T$. Let us suppose that we want to use the power method to compute the largest eigenvalue which is 1.373205080756888. Starting from a given x_0, we iterate

$$y = Ax_k, \quad x_{k+1} = \frac{y}{\|y\|}.$$

The vectors x_k converges to the eigenvector and the Rayleigh quotient $(x_k)^T Ax_k$ converges to the largest eigenvalue. However, if we are so stupid to choose $x_0 = e$, mathematically we find $y = Ax_0 = Ae = 0$ and the method cannot converge. But, in finite precision arithmetic, rounding errors come to rescue us. With fp64 and doing 50 iterations we obtain

```
>> x =
   -2.113267819223670e-001
    7.886756481218821e-001
   -5.773488661995152e-001
```

and $(x_k)^T A x_k = 1.373205080754842$ which has 12 correct decimal digits. This is due to rounding errors which are, in this case, useful. For the first iteration, we obtain

```
>> A * e
 = -5.551115123125783e-017
                         0
                         0
```

The first component is equal to $-\epsilon_M/4$ due to rounding errors. It allows the algorithm to continue without a breakdown.

Of course, we know that there are many cases where rounding errors are a nuisance. Let us consider the recurrence

$$x_{k+1} = 111 - \frac{1130 - \frac{3000}{x_{k-1}}}{x_k}, \quad x_0 = \frac{11}{2}, x_1 = \frac{61}{11},$$

proposed by J.-M. Muller. Mathematically, this is an increasing sequence converging to 6. Using fp64, the first 20 iterations give

```
5.500000000000000e+00
5.545454545454546e+00
5.590163934426244e+00
5.633431085044251e+00
5.674648620514802e+00
5.713329052462441e+00
5.749120921136040e+00
5.781810945409518e+00
5.811314669233340e+00
5.837663962407220e+00
5.861078484508624e+00
5.883542934069212e+00
5.935956716634138e+00
6.534421641135182e+00
1.541304318084583e+01
6.747239836474626e+01
9.713715118465481e+01
9.982469414672073e+01
9.998953968869486e+01
9.999937614164210e+01
```

The sequence converges to 100 and not to 6. The general solution of such a recurrence is

$$x_k = \frac{100^{k+1}a + 6^{k+1}b + 5^{k+1}c}{100^k a + 6^k b + 5^k c}.$$

Taking $c = 1$, we find $a = 0$, $b = 1$, and

$$x_k = 6 - \frac{1}{1 + \frac{6}{5}^k}.$$

In finite precision everything happens as if $a \neq 0$. In that case the sequence converges to 100. In fact, the rational number $61/11$ is not exactly representable. There is a rounding error corresponding to the truncation of the number. If, instead of using $61/11$ to compute the coefficients

of the solution, we use its rounded value, we find $a = 3.761294558884072\ 10^{-19}$. Thus, a small initial rounding error leads to a wrong solution.

Let us consider a more practical example. We would like to solve the ordinary differential equation (ODE) $y' = -y$, for $t \in [0, 1]$ with the initial value $y(0) = 1$. The solution is $y = exp(-t)$. Let us use one of the simplest methods, the order 1 forward Euler scheme,

$$y_{n+1} = y_n - hy_n, \quad y_0 = 1,\ n = 0, 1, \ldots, m-1,\ h = \Delta t = 1/m,$$

h being the time step. Let us look at the solution y_m at the end of the time interval and at the error $|y_m - \exp(-1)|$.

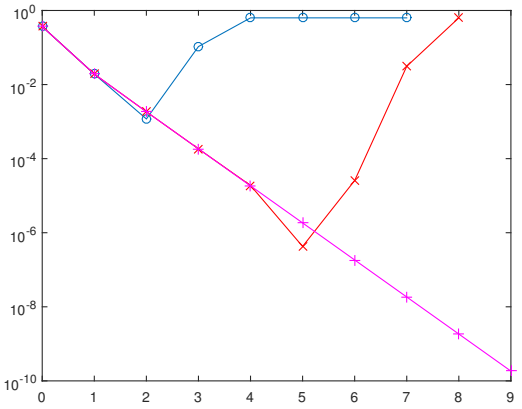


Figure 1.7. *Forward Euler scheme, error at $t = 1$ as a function of p, $m = 10^p$, fp16 (o), fp32 (+), and fp64 (*)*

Figure 1.7 show the error at $t = 1$ for as a function of the number of time steps in half, single and double precision. In half and single precisions, instead of continuing decreasing when we decrease h, at some point the error goes up, and then we do not obtain the correct solution at $t = 1$.

However, there is another way to compute y since we have $y_{n+1} = (1 - h)y_n$. We can compute the factor $1 - h$ once for all and then just do multiplications. Note that to be able to do this the time step h has to be constant in time.

There are other ways to solve this ODE. We can choose an implicit scheme (backward Euler),

$$y_{n+1} = y_n - hy_{n+1}, \quad y_0 = 1,\ n = 0, 1, \ldots, m-1,\ h = \Delta t = 1/m.$$

The multiplicative version is

$$y_{n+1} = \frac{1}{1+h} y_n.$$

Another possibility is to use a Crank-Nicolson centered scheme,

$$y_{n+1} = y_n - h\frac{y_{n+1} + y_n}{2}, \quad y_0 = 1,\ n = 0, 1, \ldots, m-1,\ h = \Delta t = 1/m,$$

and

$$y_{n+1} = \frac{1 - h/2}{1 + h/2} y_n.$$

For these three schemes, we have

$$y_{n+1} = f_h y_n \Rightarrow y_m = f_h^m y_0 = f_h^m, \quad h = 1/m,$$

with $f_h = 1 - h$ for the implicit scheme, $f_h = 1/(1+h)$ for the implicit scheme and $f_h = (1-h/2)/(1+h/2)$ for Crank-Nicolson.

Using a Taylor expansion it is straightforward to see that

$$f_h^m = e^{-1}\left(1 + \frac{h}{2} + \frac{11}{24}h^2 + O(h^3)\right)$$

for the explicit scheme and the error

$$\frac{|e^{-1} - f_h^m|}{e^{-1}} = \frac{h}{2} + \frac{11}{24}h^2 + O(h^3).$$

It is an order 1 scheme. For the implicit scheme we obtain

$$f_h^m = e^{-1}\left(1 + \frac{h}{2} - \frac{5}{24}h^2 + O(h^3)\right).$$

This is also an order 1 scheme but the factor of the second order term is smaller than for the explicit scheme. For Crank-Nicolson,

$$f_h^m = e^{-1}\left(1 - \frac{h^2}{2} + O(h^3)\right).$$

It is a second order scheme.

In finite precision arithmetic, we have to analyse the effect of rounding errors. Let us consider the multiplicative versions of those schemes. Assuming that h was computed exactly, the computed factor $\hat{f}_h$ is

$$\hat{f}_h = (1-h)(1+\delta) = f_h(1+\delta), \quad |\delta| \leq u,$$

u being the unit roundoff. Then,

$$fl(\hat{f}_h \times \hat{f}_h) = f_h^2(1+\delta)^2(1+\delta_2), \quad |\delta_2| \leq u,$$

and so on. In the end, we obtain,

$$f_h^m(1+\delta)^m \prod_{i=2}^{m}(1+\delta_i), \quad |\delta_i| \leq u.$$

From [633], we have

$$\prod_{i=2}^{m}(1+\delta_i) = 1 + \theta_m, \quad |\theta_m| \leq \frac{(m-1)u}{1-(m-1)u} = \gamma_{m-1}.$$

As long as $(m-1)u$ is not to close to 1, this factor must be close to 1. The other multiplying factor is

$$(1-h)^m(1+\delta)^m = e^{-1}\left[1 + \frac{h}{2} + \frac{11}{24}h^2 + O(h^3)\right]\left[1 + m\delta + \frac{m(m-1)}{2}\delta^2 + O(\delta^3)\right].$$

The most important terms are

$$e^{-1}\left[1+\frac{h}{2}+\frac{\delta}{h}+\frac{1-h}{2}\left(\frac{\delta}{h}\right)^2+\cdots\right].$$

When h is decreasing, terms involving δ/h can become large compared to the term $h/2$. After that δ/h and $h/2$ are of the same order, δ/h is increasing whence $h/2$ decreases.

For Crank-Nicolson, assuming that $h/2$ is exact, we obtain,

$$\frac{(1-h/2)(1+\delta_-)}{(1+h/2)(1+\delta_+)}(1+\delta)=\frac{1-h/2}{1+h/2}(1+\theta),\quad |\theta|\leq\gamma_3=\frac{3u}{1-3u},$$

and

$$e^{-1}\left[1-\frac{h^2}{2}+\frac{\theta}{h}+\frac{1-h}{2}\left(\frac{\theta}{h}\right)^2+\cdots\right](1+\theta_m).$$

Everything depends on the size of h compared to θ. In these examples, it is the error on the computation of the factor which is amplified by the successive multiplications.

If we code the explicit scheme as $y_{n+1}=y_n-hy_n$, denoting $\hat{y}_i$ the computing values, we obtain

$$\hat{y}_{n+1}=(\hat{y}_n-h\hat{y}_n(1+\delta^*_{n+1}))(1+\delta^+_{n+1}),\quad |\delta^*_{n+1}|,\,|\delta^+_{n+1}|\leq u,$$

that we can write

$$\hat{y}_{n+1}=(1-h+\mu_n)\hat{y}_n,\quad \mu_n=\delta^+_{n+1}-h(\delta^+_{n+1}+\delta^*_{n+1})+\cdots$$

Therefore,

$$\hat{y}^m=\prod_{i=0}^{m-1}(1-h+\mu_i).$$

The product can be written as

$$(1-h)^m+(1-h)^{m-1}\sum_{i=0}^{m-1}\mu_i+\cdots$$

The first term is what we obtain without rounding errors. If we denote $\hat{\epsilon}_m=\hat{y}_m-e^{-1}$ the error at $t=1$ and ϵ_m the error in exact arithmetic, we have

$$\hat{\epsilon}_m=\epsilon_m+(1-h)^{m-1}\sum_{i=0}^{m-1}\mu_i+\cdots$$

Using the Taylor expansion of $(1-h)^{m-1}$, we can write this as

$$\hat{\epsilon}_m=\epsilon_m+e^{-1}\sum_{i=0}^{m-1}\mu_i+\cdots$$

The term ϵ_m decreases with h, but the second term may increase. However, the additive coding may give better results than the multiplicative versions.

1.7 ▪ Summation algorithms

A problem that occurs quite frequently in numerical linear algebra is to compute accurately the sum of n floating-point numbers. It happens, for instance, when computing dot products or norms of vectors. The straightforward naive algorithm for a vector x is

```
s = 0
for i=1:length(x)
 s = s + x(i)
end
```

Clearly, the rounding errors depend on the order in which the components x_i are summed. We can, for instance, sort the components (if they are all available) in increasing or decreasing order. Note that this may be a costly operation. We also can sum the components pairwise $y_i = x_{2i-1} + x_{2i}$ and apply recursively the same process for the y_i's. This is called *pairwise summation*.

Whatever is the chosen ordering, denoting S the exact result, from Chapter 4 of [633], we have,

$$|S - s| \leq (n-1)u \sum_{i=1}^{n} |x_i| + O(u^2).$$

The multiplying factor can be improved to $(n-1)/(1+u)$ when summing floating-point numbers, see [676]. Another possibility is to use *compensated summation*. This technique was proposed by W. Kahan [689] in 1965 (see also [690]), and almost at the same time by I. Babuška; see also T.J. Dekker [328], A. Klein [706] and D.M. Priest [929, 930]. It allows to compute the sum of two floating-point numbers and its rounding error as follows,

```
function [s,e] = comp_sum(x,y)
if |x| < |y|, swap x and y
s = x + y;
w = s - x;
e = y - w;
% s is the sum and e the rounding error
end
```

The following scheme shows why we can obtain the rounding error in this algorithm,

x	x_1	x_2		
$+y$		y_1	y_2	
$= s$	x_1	$x_2 + y_1$		
w		y_1	0	
e			y_2	0

In fact, W. Kahan proposed the algorithm without the initial swap and this is the way it is generally used. If we use this algorithm on an example we have seen above we obtain

```
>> x = 6
6
>> y = eps / 2
1.110223024625157e-016
>> s = x + y
6
```

```
>> w = s - x
0
>> e = y - w
1.110223024625157e-016
```

We obtain the rounding error in e.

Compensated summation can be used in a summation algorithm,

```
function [s,e] = compsum(x)
s = 0
e = 0
for i=1:length(x)
 temp = s
 y = x(i) + e
 s = temp + y
 e = (temp - s) + y
end
```

The rounding error at one stage is immediately re-injected in the next stage. Note that there is no swap, and e may not contain the exact rounding error. Moreover, compensated summation is not always accurate. C.-P. Jeannerod showed in [674] that the smallest dimension n for which Kahan's summation with decreasing ordering can yield a large relative error is 4.

Deterministic and probabilistic error bounds were given by E. Hallman and I.C.F. Ipsen in [612] and [613, 614]. For the standard summation algorithm, they showed that

$$|S - s| \leq nu(1+u)^n \sum_{i=1}^{n} |x_i|.$$

For the probabilistic bound the multiplying factor of the sum depends only on $u\sqrt{n}$. For compensated summation they showed that

$$s = \sum_{i=1}^{n} (1+\rho_i)x_i, \quad |\rho_i| \leq 3u + [4(n-i)+6]u^2 + O(u^3).$$

The probabilistic bounds are too complicated to be reported here.

In [709], D.E. Knuth proposed a variant of Kahan's summation which uses more operations but no swapping.

```
function [s,e] = TwoSum(x,y)
s = x + y;
w = s - x;
e = (x - (s - w)) + (y - w);
% s is the sum and e the rounding error
end
```

Many algorithms have been proposed to sum accurately n floating-point numbers. Since the end of the 1990s, there has been at least one proposal every year. The following algorithm [869] uses TwoSum,

```
function [s,c] = sum2(x)
s = 0;
c = 0;
```

```
for i = 1:length(x)
 sm  = s;
 s = s + x(i);
 t = s - sm;
 t2 = s - t;
 t3  = x(i) - t;
 t4 = sm - t2;
 t5 = t4 + t3;
 c = c + t5; % rounding error
end
s = s + c;
```

One can go one step further. The following algorithm simulates quadruple precision (DD refers to double-double),

```
function [s,c] = DDsum(x)
s = 0;
c = 0;
for i = 1:length(x)
 sm  = s;
 s = s + x(i);
 t = s - sm;
 t2 = s - t;
 t3  = x(i) - t;
 t4 = sm - t2;
 t5 = t4 + t3;
 c = c + t5;
 sm = s;
 s = s + c;
 e = sm - s;
 c = c + e;
end
```

Another type of algorithm using TwoSum is called distillation. One first transforms the input vector x. Starting from

$$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad \cdots x_{n-1} \quad x_n]$$

we apply TwoSum to x_1 and x_2, $[s_1, e_1] = TwoSum(x_1, x_2)$ and put the results in a new vector x,

$$[e_1 \quad s_1 \quad x_3 \quad x_4 \quad \cdots x_{n-1} \quad x_n]$$

Therefore, $\sum_{i=1}^{n} x_i = e_1 + s_1 + \sum_{i=3}^{n} x_i$. Then, we apply TwoSum to s_1 and x_3 and obtain

$$[e_1 \quad e_2 \quad s_2 \quad x_4 \quad \cdots x_{n-1} \quad x_n]$$

and $\sum_{i=1}^{n} x_i = e_1 + e_2 + s_2 + \sum_{i=4}^{n} x_i$. We do this up to the end of the vector. In the end

$$[e_1 \quad e_2 \quad e_3 \quad e_4 \quad \cdots e_{n-1} \quad s_n]$$

and $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n-1} e_i + s_n$. This process can be iterated on the vector just obtained, see [968, 969, 1158, 1159]. The transformation of the vector x is done by

```
function q = VecSum(x)
q(1) = x(1);
```

```
for i = 2:length(x)
 [qi,qi1] = TwoSum(x(i),q(i-1));
 q(i) = qi;
 q(i-1) = qi1;
end
```

Then, this can be used recursively with $k > 1$,

```
function s = sumK(x,k)
for i = 1:k-1
 x = VecSum(x);
end
s = sum(x);
```

Note that one can use another summation algorithm in the last line instead of the standard recursive algorithm.

Recently, there has been an interest in mixed precision algorithms. A block algorithm was proposed in [134] where the vector x to be summed is split into blocks of m numbers. Each block is summed in low precision and the partial sums are summed using an higher precision. Here, we use this type of algorithm only in double precision with $m = 128$, but the partial sums are summed either using the standard recursive algorithm or compensated summation.

To compare the summation algorithms, let us consider an example of summation proposed by J.H. Wilkinson. Let $t = 53$ and r a small integer $r \ll t$, the sequence x_i is computed in double precision as

```
n = 2^r
x(1) = 1
x(2) = 1 - 2^(-t)
for j = 2:r
 x(2^(j-1)+1:2^j) = 1 - 2^(j-1-t)
end
```

We first choose $r = 10$ which gives 1024 components for the vector x. All the values are close to 1 but not exactly 1, except x_1. The sum is obviously close to 1024.

Computed in variable precision with the `floatp` toolbox [821] with 128 binary digits in the mantissa, the rounded sum is $1.023999999999961\ 10^3$. Using Matlab variable precision with 64 decimal digits, the sum is

$$1023.9999999999636202119290828704833984375$$

In single precision all the components of x are rounded to 1 and the sum is 1024. Table 1.4 gives the results for the summation algorithms we have described above. Except the standard recursive summation, all the algorithms give a correct result except for the last decimal digit.

Let us now use $r = 20$. The vector x has $1,048,576$ components. The variable precision result is

$$1048575.99995930989825865253806114196777343 75$$

The results are displayed in Table 1.5. All the sophisticated summation algorithms give a correctly rounded result.

Another example is the following. We compute a vector x with components

$$x_i = \sin\left(2\pi\left(\frac{i}{n} - \frac{1}{2}\right)\right),$$

Table 1.4. *Wilkinson's example,* $r = 10$

Algo.	s	cum. rounding err.	c/u
sum	1024		
comp. sum	$1.023999999999961\ 10^{3}$		
sum2	$1.023999999999961\ 10^{3}$	$-3.880507026821078\ 10^{-11}$	-349525
DDsum	$1.023999999999961\ 10^{3}$	$-3.785860513971784\ 10^{-14}$	-341
sumK 2	$1.023999999999961\ 10^{3}$	$-5.684341886080802\ 10^{-14}$	-512
sumK 3	$1.023999999999961\ 10^{3}$	$-3.785860513971784\ 10^{-14}$	-341
blk + double	$1.023999999999962\ 10^{3}$		
blk + comp	$1.023999999999961\ 10^{3}$		

Table 1.5. *Wilkinson's example,* $r = 20$

Algo.	s	cum. rounding err.	c/u
sum	1048576		
comp. sum	$1.048575999959310\ 10^{6}$		
sum2	$1.048575999959310\ 10^{6}$	$-4.069010416662966\ 10^{-5}$	$-3.66503875925\ 10^{11}$
DDsum	$1.048575999959310\ 10^{6}$	$-3.880507026821078\ 10^{-11}$	-349525
sumK 2	$1.048575999959310\ 10^{6}$	$-5.820766091346741\ 10^{-11}$	-524288
sumK 3	$1.048575999959310\ 10^{6}$	$-3.880507026821078\ 10^{-11}$	-349525
blk + double	$1.048575999959310\ 10^{6}$		
blk + comp	$1.048575999959310\ 10^{6}$		

and we sum its components. The relative errors of the summation algorithms are shown in Figure 1.8. The x-axis is the $\log_{10}$ of n.

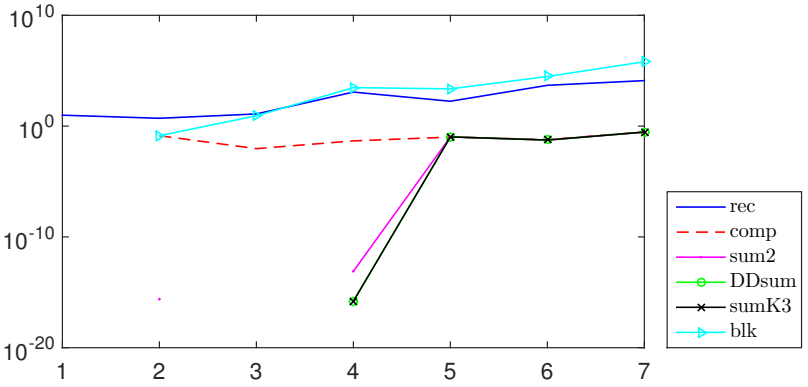


Figure 1.8. *Sine example, relative errors*

With $n = 100$ and computed in variable precision with 64 decimal digits, the sum is

$$0.00000000000000081635407030545815848214619550260235577506921246775800637962561 27$$

which, when rounded, is $8.163540703054582\ 10^{-16}$. Table 1.6 displays the results. One can see that the result of the standard recursive summation is completely wrong. Even the sign is

not correct. The result of compensated summation is better but not completely satisfactory. The three other algorithms give correct results. The best result is given by sumK with $K = 3$.

Table 1.6. *Sine example,* $n = 100$

Algo.	s	cum. rounding err.	c/u
sum	$-3.346982272038879\ 10^{-15}$		
comp. sum	$7.053317678429425\ 10^{-16}$		
sum2	$8.163540703054580\ 10^{-16}$	$4.163336342344337\ 10^{-15}$	37.5
DDsum	$8.163540703054582\ 10^{-16}$	$-2.465190328815662\ 10^{-32}$	$-2.220446049250313\ 10^{-16}$
sumK 3	$8.163540703054582\ 10^{-16}$	0	0

For $n = 1000$, the sum is

$$0.00000000000000561806865180926019581435315999540020733756921246775800637962561 3$$

which, when rounded, is $5.618068651809260\ 10^{-15}$. The results are displayed in Table 1.7. The conclusions are the same as for $n = 100$. However, Figure 1.8 shows that the results of sum2, DDsum and sumK deteriorate when $n \geq 10^5$ but their results are slightly better than for the standard algorithm.

Table 1.7. *Sine example,* $n = 1000$

Algo.	s	cum. rounding err.	c/u
sum	$-6.496436013873507\ 10^{-14}$		
comp. sum	$5.670110356088564\ 10^{-15}$		
sum2	$5.618068651809260\ 10^{-15}$	$7.058242879054433\ 10^{-14}$	635.75
DDsum	$5.618068651809260\ 10^{-15}$	$1.725633230170963\ 10^{-31}$	$1.554312234475219\ 10^{-15}$
sumK 3	$5.618068651809260e\ 10^{-15}$	0	0
blk + comp	$-4.263256414560601\ 10^{-14}$		

Let us go back to an example discussed in Subsection 1.6.2 in which the components of the vectors were alternatively $\pi/3$ and 10ε. The results for the compensated summation are:

```
      n                ps                 |(ps - psc) / psc|
      1    2.325245665339088e-15
     10    2.325245665339088e-14                           0
    100    2.325245665339087e-13     2.171258662545761e-16
   1000    2.325245665339088e-12                           0
  10000    2.325245665339088e-11                           0
 100000    2.325245665339088e-10                           0
1000000    2.325245665339087e-09     1.778695096357487e-16
10000000   2.325245665339087e-08     1.422956077085990e-16
```

Remember that "psc" was the exact sum. We see that the relative errors are of the order of the unit roundoff. The same conclusion holds for DDsum:

```
      n                ps                 |(ps - psc) / psc|
      1    2.325245665339088e-15
```

```
      10    2.325245665339088e-14                    0
     100    2.325245665339087e-13    2.171258662545761e-16
    1000    2.325245665339088e-12                    0
   10000    2.325245665339088e-11                    0
  100000    2.325245665339088e-10                    0
 1000000    2.325245665339087e-09    1.778695096357487e-16
10000000    2.325245665339087e-08    1.422956077085990e-16
```

Let us now apply the summation algorithms to the solution of the ODE $y' = -y$ with the Crank-Nicolson scheme. In Figure 1.9 the x-axis is the $\log_{10}$ of the number of time steps in $[0, 1]$ and the curves are the absolute errors at $t = 1$. The curve for compensated summation is hidden behind the curve for DDsum. We see that the multiplicative version of the scheme does not give accurate results when $n > 10^5$. Compensated summation and DDsum give better results than the standard summation and allow to obtain an accuracy close to the unit roundoff.

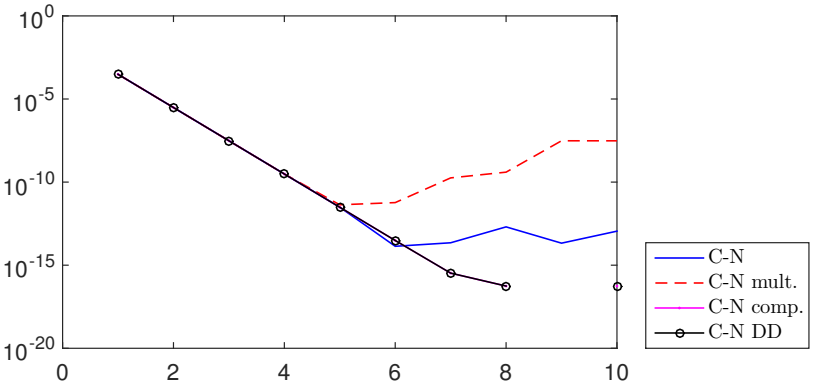


Figure 1.9. *Crank-Nicolson scheme, errors at $t = 1$*

As we said at the beginning of this section, summation algorithms can be used to compute dot products of vectors.

Definition 1.3. *The Euclidean dot product of two vectors x and y in $\mathbb{R}^n$, denoted by (x, y) or $x^T y$, is defined by*

$$(x, y) = x^T y = \sum_{i=1}^{n} x_i y_i$$

Computing a dot product needs n multiplications and $n - 1$ additions, that is $2n - 1$ floating-point operations in total.

If the vectors are complex, the dot product is defined as

$$(x, y) = \sum_{i=1}^{n} \bar{x}_i y_i,$$

where $\bar{x}_i$ is the conjugate of x_i.

The first operation to compute a dot product is the product of the two vectors component by component. A compensated product algorithm of two floating-point numbers was published in [328]. It is the following,

```
function [x,y] = TwoProd(a,b);
x = a .* b;
[ah,al] = Split(a);
[bh,bl] = Split(b);
y = al .* bl - (((x - ah .* bh) - al .* bh) - ah .* bl);
end

function [x,y] = Split(a);
% this is for double precision f = 2^27 + 1
f = 134217728 + 1;
z = a * f;
x = z - (z - a);
y = a - x;
end
```

This algorithm can be combined with any summation algorithm to compute a dot product of two vectors x and y. Since TwoProd was coded to allow the inputs to be vectors, let us write

```
function [s,c] = dotp(x,y);
[xy,e] = TwoProd(x,y); % product of components
[s,c] = DDsum(xy + e);
end
```

According to [869] the condition number (see Section 1.9) of a dot product is for $x^T y \neq 0$,

$$\operatorname{cond}(x^T y) = 2\frac{|x|^T|y|}{|x^T y|}.$$

In [1135] an algorithm was given to generate examples of dot products with a prescribed condition number. We use it to generate vectors x and y of length 1000 giving dot products with a condition number increasing from 10 to 10^{20}. The computed dot products are

cond	x^T y	dotp
10^1	1.000000000000000e-01	1.000000000000000e-01
10^2	1.000000000000000e-02	1.000000000000000e-02
10^3	1.000000000000000e-03	1.000000000000000e-03
10^4	1.000000000000000e-04	1.000000000000000e-04
10^5	1.000000000000000e-05	1.000000000000000e-05
10^6	1.000000000000000e-06	1.000000000000000e-06
10^7	1.000000000000000e-07	1.000000000000000e-07
10^8	9.999997527121991e-09	1.000000000000000e-08
10^9	9.999971547903214e-10	1.000000000000000e-09
10^10	9.999711755715453e-11	1.000000000000000e-10
10^11	9.997557923047679e-12	9.999999999999999e-12
10^12	9.971578704271450e-13	1.000000000000000e-12
10^13	9.773959005888173e-14	1.000000000000000e-13
10^14	7.442490654175345e-15	1.000000000000000e-14
10^15	-6.327471696014490e-15	1.000000000000000e-15
10^16	-7.227471660589171e-15	1.000000000000004e-16
10^17	-7.317471662340594e-15	9.999999999999885e-18
10^18	-7.326471683691561e-15	9.999999999999096e-19

```
10^19    -7.327371669944790e-15     1.000000000004643e-19
10^20    -7.327461668570112e-15     9.999999999888655e-21
```

When the condition number is 10^i, the exact value of the dot product is 10^{-i}. The relative errors in the dot products are

```
cond            x^T y                       dotp

10^1                        0                          0
10^2                        0                          0
10^3                        0                          0
10^4                        0                          0
10^5                        0                          0
10^6                        0                          0
10^7                        0                          0
10^8     2.472878009134568e-07                         0
10^9     2.845209678660400e-06                         0
10^10    2.882442845470812e-05                         0
10^11    2.442076952320854e-04                         0
10^12    2.842129572854992e-03                         0
10^13    2.260409941118270e-02                         0
10^14    2.557509345824655e-01                         0
10^15    7.327471696014489e+00     1.972152263052529e-16
10^16    7.327471660589170e+01     4.190823558986625e-15
10^17    7.327471662340594e+02     1.155557966632341e-14
10^18    7.327471683691560e+03     9.051870738620007e-14
10^19    7.327471669944789e+04     4.642694872188480e-12
10^20    7.327471668570113e+05     1.113443295426561e-11
```

When the condition number is larger that 10^7, the error for $x^T y$ starts increasing whence this happens only much later for dotp when the condition number is larger than 10^{15}. However, the complexity of dotp, which needs $26n$ floating-point operations, is much larger than for the standard dot product with $2n - 1$ operations.

1.8 ▪ Vector and matrix norms

From the dot product, we can define the Euclidean norm (also called the ℓ_2 norm) of a vector x.

Definition 1.4. *The Euclidean norm of x in $\mathbb{R}^n$ is defined as*

$$\|x\| = (x, x)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}.$$

Proposition 1.5. *Every vector norm has the three following properties,*

$$\|\lambda x\| = |\lambda| \, \|x\|, \ \forall \lambda \in \mathbb{R},$$
$$\|x\| \geq 0 \text{ and } \|x\| = 0 \textit{ if and only if } x = 0,$$
$$\|x + y\| \leq \|x\| + \|y\|.$$

□

For the ℓ_2 norm, the last relation can be shown using the Cauchy-Schwarz inequality.

Proposition 1.6. *For the dot product of two vectors in $\mathbb{R}^n$, we have*

$$|(x, y)| \leq \|x\| \, \|y\|.$$

Equality holds only if x and y are collinear.

Proof. If $y = 0$ the proof is trivial. Hence, we can assume $y \neq 0$, then we have

$$0 \leq \|x - \frac{(x, y)}{\|y\|^2} y\|^2 = \|x\|^2 - 2\frac{(x, y)^2}{\|y\|^2} + \frac{(x, y)^2}{\|y\|^4}\|y\|^2 = \frac{\|x\|^2\|y\|^2 - (x, y)^2}{\|y\|^2}.$$

Hence,

$$\|x\|^2\|y\|^2 - (x, y)^2 \geq 0,$$

which proves the result. □

A matrix norm can be defined in such a way that it is related to any given vector norm. Such a matrix norm is said to be induced by (or subordinate to) the vector norm.

Definition 1.7. *Let $\| \cdot \|_p$ be any vector norm. The induced matrix norm is defined as*

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

Another equivalent definition of an induced matrix norm is $\|A\|_p = \max_{\|y\|_p = 1} \|Ay\|_p$. Definition 1.7 implies $\|Ax\|_p \leq \|A\|_p \|x\|_p$. The following result states the properties of a matrix norm, see [548].

Proposition 1.8. *A matrix norm satisfies the following properties:*

$$\begin{aligned}
&\|\lambda A\|_p = |\lambda| \, \|A\|_p, \ \forall \lambda \in \mathbb{R}, \\
&\|A\|_p \geq 0 \text{ and} \|A\|_p = 0 \iff A = 0, \\
&\|A + B\|_p \leq \|A\|_p + \|B\|_p.
\end{aligned}$$

Proposition 1.9. *For every induced norm,*

$$\|AB\|_p \leq \|A\|_p \|B\|_p.$$

More generally, one defines every function from

$$M(\mathbb{R}^n) = \{\text{matrices of order } n \text{ with real coefficients}\}$$

to $\mathbb{R}^+$ which satisfies Proposition 1.8 as a matrix norm. We shall, however, add the property of Proposition 1.9 to the definition of a norm since it is often the most useful one. Let us consider some examples of norms. We will see the norm that is induced by the ℓ_2 norm in Proposition 1.24 but this requires some additional definitions.

Definition 1.10. *The maximum norm of a vector is* $\|x\|_\infty = \max_i |x_i|$ *and the induced matrix norm, denoted by* $\|A\|_\infty$ *is characterized in the following proposition.*

Proposition 1.11. *The* $\|.\|_\infty$ *norm of* A *is*

$$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{i,j}|.$$

Proof. Let $y = Ax$. Then,

$$\|y\|_\infty = \max_i \left| \sum_j a_{i,j} x_j \right| \leq \max_i \sum_j |a_{i,j}| \cdot |x_j| \leq \|x\|_\infty \max_i \sum_j |a_{i,j}|.$$

Hence, $\|A\|_\infty \leq \max_i \sum_j |a_{i,j}|$. Now, let us suppose that $\max_i \sum_j |a_{i,j}| = \sum_j |a_{I,j}|$. We define x such that $x_j = \text{sign } a_{I,j}$. Clearly, we have $\|x\|_\infty = 1$ and $y_I = \sum_j a_{I,j} x_j = \sum_j |a_{I,j}|$. Since,

$$\|y\|_\infty = \max_j |y_i| \geq y_I = \sum_j |a_{I,j}|,$$

$$\|A\|_\infty \geq \|Ax\|_\infty = \|y\|_\infty \geq \sum_j |a_{I,j}| = \max_i \sum_j |a_{i,j}|.$$

□

The energy norm or A-norm of a vector x is defined as $\|x\|_A = (Ax, x)^{\frac{1}{2}}$, where A is a given matrix. However, for this to be a norm, we need $(Ax, x) > 0, \forall x \neq 0$. This is true only for positive definite matrices that we will consider later in this chapter. Another example of matrix norm is the Frobenius norm.

Definition 1.12. *The Frobenius norm of* A *denoted by* $\|A\|_F$ *is defined as*

$$\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2 \right)^{\frac{1}{2}} = (\textit{trace}\ [A^T A])^{\frac{1}{2}},$$

where trace $[A^T A]$ *is the sum of the diagonal elements of* $A^T A$ *and* A^T *is the transpose of* A*.*

The Frobenius norm satisfies Proposition 1.9 but is not an induced norm, since for I_n, the identity matrix of order n, we have $\|I_n\|_F = n^{\frac{1}{2}}$. For finite dimensional spaces all norms are equivalent. In particular, the ℓ_2 and Frobenius norms are related to each other by the following inequalities.

Proposition 1.13. *The Euclidean and Frobenius norms satisfy*

$$\max_{i,j} |a_{i,j}| \leq \|A\| \leq \|A\|_F \leq n\|A\|.$$

Proof. Let us prove the first inequality. For all $x, y \neq 0$ the Cauchy-Schwarz inequality (Proposition 1.6) gives

$$\frac{|(x, Ay)|}{\|x\| \, \|y\|} \leq \frac{\|Ay\|}{\|y\|} \leq \|A\|.$$

Let us take $x = e_i$ and $y = e_j$ where e_i and e_j are vectors with zeros in all components except in the i^{th} and j^{th} position where the value is 1. Then, $|(x, Ay)| = |a_{i,j}|$, and hence $|a_{i,j}| \leq \|A\|$. This inequality holds for every i and j. Therefore,

$$\max_{i,j} |a_{i,j}| \leq \|A\|.$$

We use, once again, the Cauchy-Schwarz inequality to prove the second inequality.

$$\begin{aligned}
\|Ax\|^2 &= \sum_{i=1}^{n} (\sum_{j=1}^{n} a_{i,j} x_j)^2, \\
&\leq \sum_{i=1}^{n} \left[(\sum_{j=1}^{n} a_{i,j}^2)(\sum_{j=1}^{n} x_j^2) \right], \\
&= (\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2)(\sum_{j=1}^{n} x_j^2), \\
&= \|A\|_F^2 \|x\|^2.
\end{aligned}$$

To obtain the last inequality, we bound $a_{i,j}$ by $\max |a_{i,j}|$,

$$\|A\|_F = (\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2)^{\frac{1}{2}} \leq n \max_{i,j} |a_{i,j}| \leq n\|A\|.$$

□

Definition 1.14. *Let L be a nonsingular matrix, the L-norm of A is defined by*

$$\|A\|_L = \|LAL^{-1}\|.$$

It is easy to prove that this definition satisfies the properties of Proposition 1.8.

1.9 ▪ Condition numbers

Let us assume that we have computed an approximation $\hat{y}$ of a quantity $y = f(x)$, x and y being real numbers. The *absolute error* is defined as

$$\Delta y = y - \hat{y},$$

or $\hat{y} - y$ since the sign does not really matter. If $y \neq 0$, the *relative error* is

$$\frac{\Delta y}{y} = \frac{y - \hat{y}}{y}.$$

This type of error is called the *forward error*. Ideally, what we would like to obtain are tight bounds of the absolute value of the forward error. This is not always possible but we can be satisfied if we could answer to the question: For which data is the computed $\hat{y}$ an exact solution of the problem? That is, what are the perturbations Δx of the data x such that

$$\hat{y} = f(x + \Delta x).$$

The quantity Δx is called the *backward error* and $\Delta x/x$ the relative backward error. They measure the sensitivity of the solution to perturbations of the data. We are interested in the norm or at the minimum of the norm for all possible perturbations. In many cases, we can be satisfied if we have a small (relative) backward error. For instance, if the data is only known approximately and if the backward error is smaller than the uncertainties on the data, the computed solution is probably the best we can hope for.

A numerical method to compute $y = f(x)$ is called *backward stable* if $\hat{y}$ is obtained with a "small" Δx (or $\Delta x/x$), the meaning of "small" depending on the problem. Backward error analysis was first pioneered for linear algebra by J.W. Givens, but its use was mainly developed by J.H. Wilkinson [1119, 1120].

The relative forward and backward errors are approximately linked through the condition number of the problem. In the simple scalar case, $y = f(x)$, let us assume that f is twice continuously differentiable. For a small perturbation Δx and using a Taylor expansion we can write the forward error as

$$\hat{y} - y = f(x + \Delta x) - f(x) = f'(x)\Delta x + \frac{f''(x + \theta x)}{2}(\Delta x)^2,$$

for $\theta \in [0, 1]$. Then,

$$\frac{\hat{y} - y}{y} = \left(\frac{x f'(x)}{f(x)}\right)\frac{\Delta x}{x} + O(\Delta x)^2.$$

The quantity

$$\kappa(x) = \left|\frac{x f'(x)}{f(x)}\right|$$

is called the *condition number* of the problem. A more general definition is

$$\lim_{\epsilon \to 0} \sup_{\|\Delta x\| \leq \epsilon \|x\|} \frac{\|f(x + \Delta x) - f(x)\|}{\epsilon \|f(x)\|}.$$

If the function f is differentiable with a Jacobian J, this is equal to

$$\frac{\|x\| \, \|J(x)\|}{\|f(x)\|}.$$

The relative forward and backward errors are related by

$$\text{forward error} \approx \text{condition number} \times \text{backward error}.$$

Therefore, a problem with a small backward error can have a large forward error if the condition number is large. The condition number is a property of the problem and not of the algorithm to compute a solution.

Different definitions of the condition number were given in the literature. For instance, for the dot product of two vectors x and y we can use

$$\limsup_{\epsilon \to} \left\{ \left| \frac{(x + \Delta x)^T (y + \Delta y) - x^T y}{\epsilon \, x^T y} \right|, |\Delta x| \leq \epsilon |x|, |\Delta y| \leq \epsilon |y| \right\}.$$

Here, the perturbations are bounded elementwise. As we have seen above this is equal to $2(|x|^T |y|)/|x^T y|$. With our previous definition we would have $2(\|x\| \, \|y\|)/|x^T y|$ which is larger.

1.10 ▪ Rounding error analysis

Let us denote the set of floating-point numbers by F and assume that there are no overflow or underflow. As we have seen in Section 1.4, people generally use the standard model corresponding to IEEE arithmetic,

$$fl(x\ op\ y) = (x\ op\ y)(1+\delta), \quad |\delta| \leq u$$

for two numbers x and y in F and the four basic operations $(+, -, *, /)$, u being the unit roundoff. This means that the relative error is bounded by u. However, this was recently slightly improved by C.-P. Jeannerod and S.M. Rump [676]. For $t \neq 0$ or $fl(t) \neq 0$, let

$$E_1(t) = \frac{|t - fl(t)|}{|t|}, \quad E_2(t) = \frac{|t - fl(t)|}{|fl(t)|}.$$

When t is the exact result of an operation on one or two numbers in F and the rounding mode is "round to nearest" with any tie-breaking strategy, we have the bounds on $E_1(t)$ and $E_2(t)$ given in Table 1.8. This may seems an innocuous modification since, when computed in finite precision, u and $u/(1+u)$ are the same, but it allows an improvement on the rounding error bounds for higher level operations.

Table 1.8. *Bounds on E_1 and E_2 from [676]*

Op..	E_1	E_2
$x \pm y$	$\frac{u}{1+u}$	u
$x\,y$	$\frac{u}{1+u}$	u
x/y	$u - 2u^2$	$\frac{u-2u^2}{1+u-2u^2}$
$\sqrt{x}$	$1 - \frac{1}{\sqrt{1+2u}}$	$\sqrt{1+2u} - 1$

The basic operations we are interested in are dot products, norms, ratios of dot products or norms, matrix-vector multiplications and additions of scalar multiples of vectors.

For dot products, the standard rounding error analysis (see, for instance, [633]) using the recursive summation algorithm is

$$fl(x^T y) = (x + \Delta x)^T y = x^T (y + \Delta y),$$

where x and y are vectors of length n and

$$|\Delta x| \leq \gamma_n |x|, \quad |\Delta y| \leq \gamma_n |y|, \quad \gamma_n = \frac{nu}{1-nu}, \quad nu < 1.$$

This relates the result of $fl(x^T y)$ to perturbations of the data x or y, $fl(x^T y)$ is the exact result of a dot product with perturbed data. Therefore we can write

$$fl(x^T y) = x^T y + \Delta x^T y = x^T y + \omega nu,$$

where ω is a real number such that

$$|\omega| \leq \frac{1}{1-nu} |x^T|\,|y|.$$

The forward error bound is

$$|x^T y - fl(x^T y)| \leq \gamma_n |x^T|\,|y|.$$

Note that $|x|^T|y|$ can be much different from $x^T y$ if the signs of the elements of x and y are not all positive or negative.

The problems with these bounds is that we must have $nu < 1$. It implies that n has to be smaller than $9.0072\ 10^{15}$ for fp64, $1,6777,216$ for fp32 and $2,048$ for fp16. For low-precision formats it is a severe constraint. Moreover, if nu is close to 1, $|\omega|$ can be large. But, when x and y are in F, from [675, 676], we have

$$|x^T y - fl(x^T y)| \leq \zeta_n |x^T|\,|y|, \quad \zeta_n = \frac{(1+2u)nu - u^2}{(1+u)^2} < nu.$$

There is no restriction on n in this expression. Note that when $nu \approx 1$, the relative error bound nu is $O(1)$. The upper bound on $|\omega|$ can be replaced by $|\omega| \leq |x^T|\,|y|$. The elementwise backward error bound is

$$fl(x^T y) = (x + \Delta x)^T y = x^T (y + \Delta y), \quad |\Delta x| \leq \zeta_n |x|,\ |\Delta y| \leq \zeta_n |x|,$$

and ζ_n can be replaced by nu. For the computation of the ℓ_2 norm of a vector, from [676] we have

$$fl(\|x\|) = \|x\|(1+\delta), \quad |\delta| \leq \left(\frac{n}{2} + 1\right) u.$$

Our next step is to compute ratios of (nonzero) dot products as this is useful for some iterative methods. Assuming $w^T z \neq 0$, we have

$$fl\left(\frac{x^T y}{w^T z}\right) = \frac{fl(x^T y)}{fl(w^T z)}(1+\delta), \quad |\delta| \leq u.$$

But,

$$fl(x^T y) = (x + \Delta x)^T y,\ \ fl(w^T z) = (w + \Delta w)^T z, \quad |\Delta x| \leq \zeta_n |x|,\ |\Delta w| \leq \zeta_n |w|.$$

We write

$$\frac{1}{w^T z + (\Delta w)^T z} = \frac{1}{w^T z}\left(\frac{1}{1 + \frac{(\Delta w)^T z}{w^T z}}\right), \quad \left|\frac{(\Delta w)^T z}{w^T z}\right| \leq \zeta_n \frac{|w|^T |z|}{|w^T z|}.$$

Let us assume that $\psi_n = \zeta_n |w|^T |z| / |w^T z|$ is small. Then, we can use a Taylor expansion of the denominator,

$$\frac{1}{w^T z + (\Delta w)^T z} = \frac{1}{w^T z}\left(1 - \frac{(\Delta w)^T z}{w^T z}\right) + O(\Psi_n^2), \quad |\Psi_n| \leq \psi_n.$$

We multiply with the numerator,

$$fl\left(\frac{x^T y}{w^T z}\right) = \left\{\frac{(x + \Delta x)^T y}{w^T z}\left(1 - \frac{(\Delta w)^T z}{w^T z}\right) + (x + \Delta x)^T y\ O(\Psi_n^2)\right\}(1+\delta).$$

Therefore,

$$fl\left(\frac{x^T y}{w^T z}\right) = \frac{x^T y}{w^T z} + \left[\frac{(\Delta x)^T y}{w^T z} - \frac{x^T y}{w^T z}\frac{(\Delta w)^T z}{w^T z} + \delta \frac{x^T y}{w^T z}\right] + O(u^2).$$

The absolute value of the first order term within brackets is bounded by

$$\zeta_n \frac{|x|^T|y|}{|w^T z|} + \frac{|x^T y|}{|w^T z|}\left(|\delta| + \zeta_n \frac{|w|^T|z|}{|w^T z|}\right).$$

We can specialize this result to the computation of the ratio of squares of norms. It yields

$$fl\left(\frac{\|x\|^2}{\|w\|^2}\right) = \frac{\|x\|^2}{\|w\|^2} + \left[\frac{(\Delta x)^T x}{\|w\|^2} - \frac{\|x\|^2}{\|w\|^2}\frac{(\Delta w)^T w}{\|w\|^2} + \delta\frac{\|z\|^2}{\|w\|^2}\right] + O(u^2).$$

The absolute value of the first order term is bounded by

$$\frac{\|x\|^2}{\|w\|^2}(2\zeta_n + |\delta|) \leq (2n+1)u\frac{\|x\|^2}{\|w\|^2}.$$

The rounding error analysis for a matrix vector product $y = Ax$ follows from the results for dot products since the ith component of y is the dot product of the ith row of A and the vector x. It depends on m_i, the number of nonzero entries in the ith row of A. To obtain a simple result, we usually use m, the maximum number of entries in any row of A. If the number of nonzero in a row varies greatly, the bounds cannot be very tight. The standard bound (see [633]) is

$$fl(y) = (A + \Delta A)x, \quad |\Delta A| \leq \gamma_m |A|.$$

Using the results in [675, 676], we have a better result,

$$|\Delta A| \leq \zeta_m |A|.$$

Let us consider the ith component of the matrix-vector product. We denote the ith row of A as $A_{i,:}$. Then,

$$fl(y_i) = (A_{i,:} + \Delta A_{i,:})x, \quad |\Delta A_{i,:}x| \leq \zeta_{m_i}|A_{i,:}|\,|x| \leq m_i u |A_{i,:}|\,|x|.$$

Hence, we can write

$$fl(y) = Ax + uc^A, \quad |c_i^A| \leq m_i |A_{i,:}|\,|x|,$$

where c^A is a vector for which we have $|c^A| \leq m|A|\,|x|$ componentwise.

For the ℓ_2 norm, we have

$$\|y - fl(y)\| \leq \|\Delta A\|\,\|x\|.$$

We can bound $\|\Delta A\|$ either by $\|A\|$ or by $\|\,|A|\,\|$. This leads to

$$\|y - fl(y)\| \leq n^{\frac{1}{2}}\zeta_m \|A\|,$$

or

$$\|y - fl(y)\| \leq \zeta_m \|\,|A|\,\|.$$

We are also interested in dot products involving a matrix-vector product like $(Ax, x) = x^T Ax \neq 0$. This dot product can be written as

$$\begin{aligned} fl(x^T Ax) &= x^T(Ax + uc^A) + \omega n u, \quad \omega \leq |x^T|\,|Ax + uc^A|, \;\; |c^A| \leq m|A|\,|x|, \\ &= x^T Ax + u(\omega n + x^T c^A). \end{aligned}$$

Since ω is multiplied by u, we can write

$$fl(x^T Ax) = x^T Ax + u\xi + O(u^2), \quad |\xi| \leq (m+n)|x^T|\,|A|\,|x|.$$

Let $x \neq 0$ and y be two vectors such that $y^T Ay > 0$. We will also be interested in the ratio of $\|x\|^2$ and $y^T Ay$. Using our previous results, with $|\omega| \leq \|x\|^2$ and $|\xi| \leq (m+n)|y^T|\,|A|\,|y|$, we can write

$$\begin{aligned} fl\left(\frac{\|x\|^2}{y^T Ay}\right) &= \frac{\|x\|^2 + \omega nu}{y^T Ay + \xi u + O(u^2)}\,(1+\delta), \quad |\delta| \leq u, \\ &= \frac{\|x\|^2}{y^T Ay}\,\frac{1 + \delta + \frac{\omega nu}{\|x\|^2} + O(u^2)}{1 + \frac{\xi u}{y^T Ay} + O(u^2)}. \end{aligned}$$

Assuming that $\xi u / y^T Ay$ is small and using a Taylor expansion of the denominator, we obtain

$$fl\left(\frac{\|x\|^2}{y^T Ay}\right) = \frac{\|x\|^2}{y^T Ay}(1 + \rho u + O(u^2)), \quad |\rho| \leq (m+n)\frac{|y^T|\,|A|\,|y|}{y^T Ay} + n + 1.$$

In several of the algorithms we will study, we have to do linear combinations of vectors. Let z and y be two vectors and α and β be two real numbers. We have

$$fl(\alpha x + \beta y) = (fl(\alpha x) + fl(\beta y))(1+\delta), \quad |\delta| \leq u.$$

Then,

$$fl(\alpha x) = \alpha x(1+\delta_1), \quad fl(\beta x) = \beta x(1+\delta_2), \quad |\delta_i| \leq u,\ i = 1, 2,$$

and

$$fl(\alpha x + \beta y) = \alpha x + \beta y + \alpha x(\delta_1 + \delta) + \beta y(\delta_2 + \delta) + O(u^2).$$

Therefore,

$$fl(\alpha x + \beta y) = \alpha x + \beta y + u\,c + O(u^2), \quad |c| \leq 2(|\alpha x| + |\beta y|).$$

We observe that, using the results in [676], and when the rounding mode is "round to nearest", we can get rid of the $O(u^2)$ terms with still the same bound for c. This yields

$$|fl(\alpha x + \beta y) - (\alpha x + \beta y)| = u\,|c| \leq 2u(|\alpha|\,|x| + |\beta|\,|y|),$$

and

$$\|fl(\alpha x + \beta y) - (\alpha x + \beta y)\| = u\,\|c\| \leq 2u(|\alpha|\,\|x\| + |\beta|\,\|y\|).$$

There are algorithms where x is going to be replaced by the result of the computation of Ax. Then,

$$fl(\alpha fl(Ax) + \beta y) = \alpha fl(Ax) + \beta y + u\,c, \quad |c| \leq 2(|\alpha fl(Ax)| + |\beta y|).$$

Using the results for the matrix-vector product, in the end we obtain,

$$fl(\alpha Ax + \beta y) = \alpha Ax + \beta y + u\,c', \quad |c'| \leq m\alpha|A|\,|x| + 2(|\alpha Ax| + |\beta y| + um|A|\,|x|).$$

Note that the last term in the upper bound has a factor u and thus gives a second order term in u.

1.11 ▪ Orthogonalization

With the dot product of Definition 1.3, we can generalize the usual definition of orthogonality in $\mathbb{R}^3$.

Definition 1.15. *Two real vectors x and y are said to be orthogonal if and only if*

$$(x, y) = x^T y = 0.$$

Every set of linearly independent vectors $\{x_1, x_2, \ldots, x_n\}$ can be orthogonalized in the following way.

Proposition 1.16. *We can find a set of orthogonal vectors $\{q_1, q_2, \ldots, q_n\}$ of unit norm such that span $(q_1, q_2, \ldots, q_n)$ = span $(x_1, x_2, \ldots, x_n)$, where span $(x_1, x_2, \ldots, x_n)$ is the subspace spanned by the vectors $x_1, x_2, \ldots, x_n$.*

Proof. The method to obtain q_i, is known as the Gram-Schmidt orthogonalization process. Let us first consider only two vectors, that is, $n = 2$. Let x_1 and x_2 be given and define

$$\begin{aligned} q_1 &= \frac{x_1}{\|x_1\|}, \\ z_2 &= x_2 - \frac{(x_1, x_2)}{\|x_1\|^2} x_1 = x_2 - (q_1, x_2) q_1, \\ q_2 &= \frac{z_2}{\|z_2\|}. \end{aligned}$$

Note that $\frac{(x_1,x_2)}{\|x_1\|^2} x_1$ is the component of x_2 in the direction x_1. If we subtract this component from x_2 we obtain a vector q_2 which is orthogonal to x_1. The vectors q_1 and q_2 are independent, linear combinations of x_1 and x_2 and span the same subspace. This process can be easily generalized to n vectors giving

$$\begin{aligned} q_1 &= \frac{x_1}{\|x_1\|}, \\ z_i &= x_i - \frac{(q_1, x_i)}{\|q_1\|^2} q_1 - \cdots - \frac{(q_{i-1}, x_i)}{\|q_{i-1}\|^2} q_{i-1}, \\ q_i &= \frac{z_i}{\|z_i\|}, \quad i = 2, \ldots, n. \end{aligned}$$

One can check that the q_i are orthogonal and by induction that the spannned subsets are the same. □

Being orthogonal and of unit norm, the vectors q_i are said to be *orthonormal*. Note that for the Gram-Schmidt algorithm we need all the previous vectors to compute q_i.

Definition 1.17. *A matrix Q is an orthonormal matrix if and only if the columns of Q are orthonormal vectors.*

For an orthonormal matrix Q, we have $Q^T Q = I$. If we just have the orthogonality of the columns of Q, the matrix is said to be *orthogonal*. Note that, if we think of the vectors $\{x_1, \ldots, x_n\}$ as columns of a matrix A, the Gram-Schmidt process is nothing other than computing a factorization $A = QR$ where Q (whose columns are the vectors q_i) is orthonormal and R is upper triangular. In the Gram-Schmidt algorithm Q and R are computed one column at a time. Unfortunately, this classical Gram-Schmidt algorithm (`CGS`) has poor numerical properties in finite precision arithmetic. Typically, there is a loss of orthogonality in the computed vectors.

However, there is a rearrangement of the computation, called the modified Gram-Schmidt (`MGS`) algorithm, that leads to much better results, see [966], [129], [548], [631]. In `MGS`, as soon

as one q_i is computed, all the remaining vectors are orthogonalized against it. The algorithm is the following. We introduce partially orthogonalized vectors $q_i^{(j)}$ whose initial values are $q_i^{(1)} = x_i$, $i = 1, \ldots, n$. Then, for $i = 1, \ldots, n$,

$$q_i = \frac{q_i^{(i)}}{\|q_i^{(i)}\|},$$

and for $j = i + 1, \ldots, n$,

$$q_j^{(i+1)} = q_i^{(i)} - (q_i, q_j^{(i)}) q_i.$$

For stability results on MGS, see N.J. Higham [631, 633].

Gram-Schmidt orthogonalization algorithms in finite precision arithmetic were first studied to compute a QR factorization; see, for instance, [128, 129]. The main application was the computation of the solution of least squares problems, see [725, 130]. As we said, because of rounding errors, there is quite often a gradual loss of orthogonality in the computed columns of the matrix Q. The differences between CGS and MGS were first shown by numerical experiments in [943] where it was observed that orthogonality is better preserved in MGS than in CGS.

The reason for the loss of orthogonality can be seen with only two vectors, see [129]. Let q_1 of unit norm and b_2 be two vectors. We orthogonalize b_2 against q_1,

$$\hat{q}_2 = b_2 - r_{1,2}\, q_1, \quad r_{1,2} = q_1^T b_2, \quad q_2 = \frac{\hat{q}_2}{\|\hat{q}_2\|}.$$

Let us assume that q_1 and b_2 are known exactly and that the norms are computed exactly. We compute $fl(\hat{q}_2)$ and it can be shown that

$$\|fl(\hat{q}_2) - \hat{q}_2\| \leq \alpha u \|b_2\|,$$

where α is equal to the length of the vectors times a small constant. Then,

$$I - \begin{pmatrix} q_1 & \frac{fl(\hat{q}_2)}{\|fl(\hat{q}_2)\|} \end{pmatrix}^T \begin{pmatrix} q_1 & \frac{fl(\hat{q}_2)}{\|fl(\hat{q}_2)\|} \end{pmatrix} = - \begin{pmatrix} 0 & \frac{q_1^T fl(\hat{q}_2)}{\|fl(\hat{q}_2)\|} \\ \frac{q_1^T fl(\hat{q}_2)}{\|fl(\hat{q}_2)\|} & 0 \end{pmatrix}.$$

Let us define the level of orthogonality as the norm of that matrix. It is

$$\frac{|q_1^T fl(\hat{q}_2)|}{\|fl(\hat{q}_2)\|}.$$

Since q_1 and $\hat{q}_2$ are exactly orthogonal,

$$|q_1^T fl(\hat{q}_2)| = |q_1^T (fl(\hat{q}_2) - \hat{q}_2)| \leq \|fl(\hat{q}_2) - \hat{q}_2\| \leq \alpha u \|b_2\|.$$

The bound on the level of orthogonality is proportional to

$$u \frac{\|b_2\|}{\|fl(\hat{q}_2)\|}.$$

This bound can be large if $\|fl(\hat{q}_2)\|$ is small. This happens if the angle between q_1 and b_2 is small.

Let us consider a small example,

$$A = \begin{pmatrix} 0.7071067812 & 0.7071067883 \\ 0.7071067812 & 0.7071067741 \end{pmatrix}.$$

This matrix is of rank 2. The second column is equal to the first column slightly rotated. With CGS and MGS (which are the same in this simple case) the computed matrix Q is

$$Q = \begin{pmatrix} 0.7071067812 & 0.7071067923 \\ 0.7071067812 & -0.7071067701 \end{pmatrix},$$

and

$$Q^T Q = \begin{pmatrix} 1 & 1.570092452\ 10^{-8} \\ 1.570092452\ 10^{-8} & 1 \end{pmatrix}$$

which is far from the identity matrix. If we orthogonalize a second time, we obtain a matrix Q_2 such that

$$Q_2^T Q_2 = \begin{pmatrix} 1 & -8.865115929\ 10^{-17} \\ -8.865115929\ 10^{-17} & 1 \end{pmatrix}$$

which is the identity matrix up to working precision.

The loss of orthogonality in CGS was studied in [534, 535, 1027] when orthogonalizing the columns of a matrix A. Note that in [1027] the normalization is done in a different way. In these papers it is proved that the computed quantities satisfy

$$\|I - Q^T Q\| \leq \frac{\alpha}{1 - \beta u \kappa^2(A)} u \kappa^2(A),$$

when $\beta u \kappa^2(A) < 1$ and α depends on the dimension of A. This condition corresponds to $A^T A$ being numerically nonsingular.

In [128] it was proved that, with MGS, the computed matrix Q satisfies

$$\|I - Q^T Q\| \leq \frac{\gamma}{1 - \delta \kappa(A) u} u \kappa(A),$$

if $\delta u \kappa(A) < 1$ where γ and δ are constants depending on the number of columns of A. For MGS the bound depends on $\kappa(A)$ whence for CGS it depends on $\kappa^2(A)$. The rank and the condition number of the computed Q in MGS were studied in [533] where, under some restrictions on the condition number of A and the arithmetic, it was shown that $\kappa(Q) \leq 1.3$.

1.12 ▪ Eigenvalues and singular values

Even though we do not study algorithms to compute eigenvalues in this book, it is useful to recall some results about eigenvalues and singular values since there are involved in the convergence of some iterative algorithms to solve linear systems.

Definition 1.18. *$\lambda \in \mathbb{C}$ is an eigenvalue of a square matrix A if and only if there exists at least one vector $x \neq 0$ such that*

$$Ax = \lambda x.$$

The vector x is said to be an eigenvector of A.

For every real or complex matrix there is at least one eigenvalue. The matrix $A - \lambda I$ being singular, its determinant is zero. But this determinant is a polynomial of degree n in λ. Consequently, from the fundamental theorem of algebra, it has at least one root which is an eigenvalue of A. Of course, some of the n roots can be equal. The eigenvectors can be normalized as we wish. Usually, they are taken as being of unit ℓ_2 norm.

Definition 1.19. *If $\lambda_i = \lambda_{i+1} = \cdots = \lambda_{i+m-1} = \lambda$ we say that λ is of algebraic multiplicity m. The geometric multiplicity is defined as the dimension of the subspace spanned by the eigenvectors associated with λ.*

The two multiplicities may be different, as shown in the following example,

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \lambda_1 = \lambda_2 = 0.$$

The algebraic multiplicity is 2 but the geometric multiplicity is 1.

An eigenvalue λ_i of algebraic multiplicity $m_i = 1$ is said to be *simple*. If $m_i > 1$ and there are m_i associated independent eigenvectors, it is said to be *semi-simple*.

What completely describes the eigenstructure of a matrix is the *Jordan canonical form*. For every square matrix with real or complex entries there exists a nonsingular matrix X such that

$$X^{-1}AX = J$$

where J is a block diagonal matrix with blocks of a simple bidiagonal structure. The diagonal blocks are of the form

$$J^{(i,j)} = \begin{pmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix}, \; i = 1, \ldots, \ell, \; j = 1, \ldots, g_i, \tag{1.1}$$

where ℓ is the number of distinct eigenvalues of A and g_i is the geometric multiplicity of λ_i. The matrices $J^{(i,j)}$ are called *Jordan blocks* and have only two nonzero constant diagonals.

Let $d_{i,j}$ be the order of $J^{(i,j)}$. Then, the characteristic (and irreducible) polynomial of $J^{(i,j)}$ is $(\lambda - \lambda_i)^{d_{i,j}}$. There may be several Jordan blocks associated with one eigenvalue and the block diagonal matrix whose diagonal blocks are all these Jordan blocks $J^{(i,j)}$, $j = 1, \ldots, g_i$ is called a *Jordan box* and denoted as $B^{(i)}$ when these Jordan blocks are numbered sequentially. The matrix $B^{(i)}$ is of order m_i and contains g_i blocks. The Jordan canonical form is unique up to the order of the diagonal Jordan blocks.

If A has only simple or semi-simple eigenvalues (which means that $\sum_i m_i = n$) or, equivalently, if there exists a basis of $\mathbb{C}^n$ consisting of eigenvectors, the matrix is *diagonalizable*. There exists a nonsingular matrix X such that

$$X^{-1}AX = \Lambda, \tag{1.2}$$

where Λ is a diagonal matrix whose diagonal entries are the eigenvalues λ_i of A. The factorization $A = X\Lambda X^{-1}$ is called the *spectral factorization* of A. The most general result that can be proved is that A is similar to a diagonal matrix if the geometric multiplicity of each eigenvalue is the same as its algebraic multiplicity. If A is not diagonalizable, it is said to be *defective* and one has to use the Jordan canonical form.

A matrix A is *normal* if $A^H A = AA^H$ where A^H (which is sometimes written as A^*) denotes the conjugate transpose of A. It is known that A is normal if and only if it can be diagonalized by a unitary matrix. In other words, the matrix X in (1.2) can be chosen to be unitary (see the definition below) and the spectral factorization of A is $A = X\Lambda X^H$. Examples of normal matrices are Hermitian, symmetric and skew-symmetric matrices.

To state an important theorem, we now consider complex numbers. As we have seen above, the dot product of two vectors x and y with complex components is defined by

$$(x, y) = \sum_{i=1}^{n} \overline{x_i} y_i,$$

where $\overline{x_i}$ denotes the conjugate of $x_i \in \mathbb{C}$. The matrix A^H is given by $(A^H)_{i,j} = \overline{a_{j,i}}$. A complex matrix A which is orthogonal is called *unitary*. A matrix A is *Hermitian* if and only if $A^H = A$.

Theorem 1.20. *For every matrix A there exists a unitary matrix U such that*

$$U^H A U = T,$$

where T is an upper triangular matrix.

Proof. For the proof we follow G. Strang [1054]. As we said above, every matrix A has at least one eigenvalue, say λ_1 (which may be of algebraic multiplicity n) and at least one unit norm eigenvector x_1. With the Gram-Schmidt orthogonalization process, we can find $n-1$ vectors $u_2^{(1)}, \ldots, u_n^{(1)}$ such that

$$U_1 = (x_1, u_2^{(1)}, \ldots, u_n^{(1)})$$

is a unitary matrix. Then,

$$AU_1 = U_1 \begin{pmatrix} \lambda_1 & * & \cdots & \cdots & * \\ 0 & & & & \\ \vdots & & A^{(2)} & & \\ 0 & & & & \end{pmatrix}.$$

This process can be repeated on $A^{(2)}$ which has at least one eigenvalue λ_2 and an eigenvector x_2. So there exists a unitary matrix V_2 of order $n-1$ such that

$$A^{(2)} V_2 = V_2 \begin{pmatrix} \lambda_2 & * & \cdots & * \\ 0 & & & \\ \vdots & & A^{(3)} & \\ 0 & & & \end{pmatrix}.$$

Let us denote

$$U_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & V_2 & \\ 0 & & & \end{pmatrix},$$

then,

$$\begin{pmatrix} \lambda_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & A^{(2)} & \\ 0 & & & \end{pmatrix} U_2 = \begin{pmatrix} \lambda_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & A^{(2)} V_2 & \\ 0 & & & \end{pmatrix}$$

$$= U_2 \begin{pmatrix} \lambda_1 & * & * & \cdots & * \\ 0 & \lambda_2 & * & \cdots & * \\ 0 & 0 & & & \\ \vdots & \vdots & & A^{(3)} & \\ 0 & 0 & & & \end{pmatrix}.$$

The general result follows by induction. □

The relation $A = UTU^H$ is called the *Schur factorization* of A.

Corollary 1.21. *Let A be an Hermitian matrix. Then, there exists a unitary matrix U such that*

$$U^H AU = \Lambda,$$

where Λ is a diagonal matrix whose diagonal entries are the eigenvalues of A.

□

Let us now return to real matrices. We have already seen that a symmetric matrix is diagonalizable. The following useful result is used in many proofs in numerical linear algebra.

Theorem 1.22. *Let A be a real symmetric matrix. There exists an orthogonal matrix Q such that*

$$Q^H AQ = Q^T AQ = \Lambda,$$

where Λ is a diagonal matrix whose diagonal entries are the eigenvalues of A (which are real numbers).

□

We have the following important definition.

Definition 1.23. *The spectral radius $\rho(A)$ of A is defined as*

$$\rho(A) = \max_{1 \le i \le n} |\lambda_i|,$$

where λ_i are the eigenvalues of A.

Since the eigenvalues of a real square matrix may be complex numbers, it is sometimes useful to associate real numbers to a matrix. The *singular values* σ_i of an $n \times m$ complex rectangular matrix A are the square roots of the nonzero eigenvalues of $A^H A$. It can be shown that they are also the nonzero eigenvalues of AA^H. If A is of rank r, there are r nonzero eigenvalues of $A^H A$ and of AA^H and thus r singular values which are usually ordered according to $\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_r > 0$.

The *right singular vectors* are the eigenvectors of $A^H A$. Let us collect them in the columns of a unitary matrix V of order m. The *left singular vectors* are the eigenvectors of AA^H that we collect in the columns of a unitary matrix W of order n. Then, the matrix A can be factorized into its *singular value decomposition* (SVD),

$$A = W\Sigma V^H,$$

where the $n \times m$ matrix Σ is zero except for its leading principal submatrix of size r which is diagonal and contains the singular values σ_i in descending order.

It is easy to see that the ℓ_2 norm of A is $\|A\| = \sigma_1$, the maximum singular value. The ℓ_2 condition number of A is therefore $\sigma_1(A)/\sigma_n(A)$. For symmetric matrices we have the following result.

Proposition 1.24. *Let A be a symmetric matrix. Then, $\|A\| = \rho(A)$.*

Proof. This is implied by the general result using the SVD. □

For studying the convergence of some iterative methods, we will have to consider the limit of the sequence of powers A^k for $k = 1, 2, \ldots$, of a given matrix A. The sequence of matrices $A^{(k)} = A^k$ converges to a limit A^∞ as k tends to infinity if and only if

$$\lim_{k\to\infty} a_{i,j}^{(k)} = a_{i,j}^{\infty}, \quad \text{for all } i, j.$$

This definition is equivalent to

$$\lim_{k\to\infty} \|A^{(k)} - A^\infty\| = 0.$$

The main result is the following.

Theorem 1.25. *We have $\lim_{k\to\infty} A^k = 0$ if and only if the eigenvalues of A are in the unit circle, that is,*

$$\lim_{k\to\infty} A^k = 0 \Leftrightarrow \rho(A) < 1.$$

Proof. This was proved in 1940 by R. Oldenburger [873] using the SVD. It is not too difficult to write powers of the Jordan blocks and to study their limits. Another proof was given by A.S. Householder [647]. □

It is clear that $\rho(A) \leq \|A\|$. In fact, this is true for all norms. Hence, $\|A\| < 1$ is a sufficient condition to have $\lim_{k\to\infty} A^k = 0$.

Now we consider series of matrices. We have the following result.

Theorem 1.26. *The series $I + A + A^2 + \cdots$ converges and the limit is $(I - A)^{-1}$ if and only if $\rho(A) < 1$.*

Proof. Assume that $\rho(A) < 1$ and let λ be an eigenvalue of $I - A$. Then $1 - \lambda$ is an eigenvalue of A and $1 - \lambda < 1$ because $\rho(A) < 1$. Therefore, $I - A$ is non-singular. Let us denote the partial sums of the series by S_k.

$$\begin{aligned} S_k &= I + A + A^2 + \cdots + A^k, \\ (I - A)S_k &= I - A^{k+1}, \\ S_k &= (I - A)^{-1}(I - A^{k+1}), \\ S_k - (I - A)^{-1} &= -(I - A)^{-1}A^{k+1}, \\ \|S_k - (I - A)^{-1}\| &\leq \|(I - A)^{-1}\| \, \|A^{k+1}\|. \end{aligned}$$

As $\rho(A) < 1$, $\lim_{k\to\infty} \|A^{k+1}\| = 0$ and

$$\lim_{k\to\infty} \|S_k - (I - A)^{-1}\| = 0.$$

For the converse we observe that if $\lim_{k\to\infty} S_k$ exists, it implies $\lim_{k\to\infty} A^k = 0$ and so, $\rho(A) < 1$ by Theorem 1.25. □

The series $I + A + A^2 + \cdots$ is said to be the *Neumann series* for $(I - A)^{-1}$ and S_k (for small k) is frequently used in numerical algorithms to approximate $(I - A)^{-1}$ when $\rho(A) < 1$.

1.13 ▪ Irreducibility and diagonal dominance

Some theorems about convergence of iterative methods depend on constraints on the eigenvalues which are difficult to verify in practical problems. Therefore, since the beginning of the use of

these methods, researchers tried to find sufficient conditions for convergence, that is, classes of matrices for which they could know that the methods are convergent. In this section we study the properties of some of these classes of matrices.

Definition 1.27. *Let $\mathcal{I} = \{1, 2, \ldots, n\}$. A matrix A is irreducible if*

1) $n = 1$ and $a_{1,1} \neq 0$ or

2) $n > 1$ and for every set of integers $I_1 \in \mathcal{I}$ and $I_2 \in \mathcal{I}$

with $I_1 \cap I_2 = \emptyset$ and $I_1 \cup I_2 = \mathcal{I}$, there exists $i \in I_1$, and $j \in I_2$ such that $a_{i,j} \neq 0$.

This definition is complicated but the following result helps to understand what it means. When the matrix A is not irreducible, it is said to be *reducible*.

Proposition 1.28. *A matrix A is reducible if and only if there exists a permutation matrix P such that*

$$P^{-1}AP = \begin{pmatrix} D_1 & 0 \\ F & D_2 \end{pmatrix},$$

D_1, D_2 being square matrices.

Proof. A permutation matrix is a matrix whose columns are a permutation of the columns of the identity matrix. For a proof of this proposition, see D.M. Young [1144]. □

A being irreducible means that one cannot solve the linear system $Ax = b$ by solving two subproblems of smaller size.

Definition 1.29. *A matrix A is*

• *diagonally dominant (by rows) if*

$$|a_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|, \quad \forall i,$$

• *strictly diagonally dominant if*

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|, \quad \forall i,$$

• *irreducibly diagonally dominant if*

1) A is irreducible,

2) A is diagonally dominant,

3) there exists an index i such that

$$|a_{i,i}| > \sum_{j=1, j \neq i}^{n} |a_{i,j}|.$$

Theorem 1.30. *Let A be irreducibly diagonally dominant. Then, A is nonsingular and $a_{i,i} \neq 0$ for all i.*

Proof. Let us prove by contradiction that $a_{i,i} \neq 0$. Assume that $n > 1$ and $a_{i,i} = 0$ for some i. Diagonal dominance implies that $a_{i,j} = 0$ for all j. But, this contradicts the irreducibility of A because we can choose $I_1 = i$, $I_2 = \mathcal{I} - \{i\}$ and have $a_{i,j} = 0$ for all $i \in I_1$ and for all $j \in I_2$. Let us now prove that A is nonsingular. Since $a_{i,i} \neq 0$ for all i, we can define

$$B = I - D^{-1}A,$$

where D is a diagonal matrix with $d_{i,i} = a_{i,i}$ for all i (in abbreviated form $D = \text{diag}(A)$). Clearly,

$$\begin{aligned} b_{i,i} &= 0, \\ b_{i,j} &= -\frac{a_{i,j}}{a_{i,i}}, \quad i \neq j. \end{aligned}$$

Since A is irreducibly diagonally dominant, we have

$$\sum_{j=1}^{n} |b_{i,j}| \leq 1, \quad \text{for all} \quad i,$$

and there exists an index i such that $\sum_{j=1}^{n} |b_{i,j}| < 1$. Using Proposition 1.11, we obtain $\|B\|_\infty \leq 1$. Hence, $\rho(B) \leq 1$. Let us assume there exists an eigenvalue λ of modulus 1 and let x be the corresponding eigenvector,

$$\sum_{j=1}^{n} b_{i,j} x_j = \lambda x_i.$$

We end the proof as in [1144]. Let i be such that $|x_i| = \|x\|_\infty$, then

$$\sum_{j=1}^{n} |b_{i,j}| \, \|x\|_\infty \leq \|x\|_\infty \leq \sum_{j=1}^{n} |b_{i,j}| \, |x_j|.$$

The second inequality follows because

$$|\lambda x_i| = |\lambda| \, |x_i| = |x_i| = \|x\|_\infty \leq \sum_{j=1}^{n} |b_{i,j}| \, |x_j|.$$

Hence,

$$\sum_{j=1}^{n} |b_{i,j}| \big(|x_j| - \|x\|_\infty \big) \geq 0,$$

but

$$|x_j| - \|x\|_\infty \leq 0 \quad \text{for all} \quad j.$$

This implies that $|x_j| = \|x\|_\infty$ or $b_{i,j} = 0$ but B is irreducible. Therefore, from [1144], there exist indices $i_1, \ldots, i_p$ such that

$$b_{i,i_1}, b_{i_1,i_2}, \ldots, b_{i_p,j} \neq 0.$$

This shows that

$$|x_{i_1}| = |x_{i_2}| = \cdots = |x_j| = \|x\|_\infty.$$

In all cases, we have $|x_j| = \|x\|_\infty$ for all j and therefore

$$\|x\|_\infty = |x_i| \leq \sum_{j=1}^{n} |b_{i,j}| \, \|x\|_\infty,$$

for all i. Since $\|x\|_\infty \neq 0$, $\sum_{j=1}^{n} |b_{i,j}| \geq 1$ for all i. This contradicts the fact that there is at least one index i such that $\sum_{j=1}^{n} |b_{i,j}| < 1$. Thus we have proved that $\rho(B) < 1$ and by Theorem 1.26, $I - B = D^{-1}A$ is nonsingular. □

Corollary 1.31. *Let A be strictly diagonally dominant. Then, A is nonsingular and $a_{i,i} \neq 0$ for all i.*

□

The eigenvalues of diagonally dominant matrices have the following property.

Theorem 1.32. *Let A be irreducibly diagonally dominant and assume that $a_{i,i} > 0$ for all i. Then, for every eigenvalue λ, $Re(\lambda) > 0$.*

Proof. From Theorem 1.30, A is nonsingular. Let x be an eigenvector of A. Then, we have

$$\lambda x_i = \sum_{j=1}^{n} a_{i,j} x_j,$$

$$(\lambda - a_{i,i}) x_i = \sum_{j=1, j\neq i}^{n} a_{i,j} x_j.$$

Let us choose an index i such that $|x_i| = \|x\|_\infty$, then

$$|\lambda - a_{i,i}| \leq \sum_{j=1, j\neq i}^{n} |a_{i,j}|.$$

This proves that the eigenvalues belong to the union of the disks with centers $a_{i,i}$ and radius $\sum_{j=1, j\neq i}^{n} |a_{i,j}|$. These are known as *Gerschgorin disks*. We observe that these disks contain only complex numbers with positive real parts because if there is a $b > 0$ such that $-b$ belongs to the disk then

$$a_{i,i} < a_{i,i} + b = |-b - a_{i,i}| \leq \sum_{j=1, j\neq i}^{n} |a_{i,j}|.$$

This contradicts the diagonal dominance of A. Moreover, we cannot have $\lambda = 0$ because A is nonsingular. □

The same result holds if A is strictly diagonally dominant. The definition of diagonal dominance was generalized as follows.

Definition 1.33.

• *A matrix A is generalized diagonally dominant if there exists a vector d with $d_i > 0$ for all i such that*

$$|a_{i,i}| d_i \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}| d_j.$$

• *A is generalized strictly diagonally dominant if*

$$|a_{i,i}|d_i > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|d_j.$$

These definitions are equivalent to applying the classical definitions of diagonal dominance to the matrix AD where D is the diagonal matrix with diagonal entries d_i. Using what we have proved for diagonally dominant matrices, we have the following result.

Proposition 1.34. *Let A be generalized strictly diagonally dominant. Then, A is nonsingular and $a_{i,i} \neq 0$ for all i.*

□

The definition of generalized diagonal dominance can also be expressed in a different way. There exists a diagonal matrix D with strictly positive diagonal elements such that $D^{-1}AD$ is strictly diagonally dominant. To see this, we write the definition in the following form

$$|a_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|\frac{d_j}{d_i}.$$

This last definition is often more convenient. Let us now give a definition which has some formal analogy with irreducibility.

Definition 1.35. *The matrix A has property A if there exist two sets of indices I_1 and I_2 that partition the set $\mathcal{I}$ and if $i \neq j$ and $a_{i,j} \neq 0$ or $a_{j,i} \neq 0$, then $i \in I_1$ and $j \in I_2$ or $j \in I_2$ and $j \in I_1$.*

This definition was introduced by D.M. Young [1144] who showed the following characterization for matrices having property A.

Proposition 1.36. *The matrix A has property A if and only if A is diagonal or there exists a permutation matrix P such that*

$$P^{-1}AP = \begin{pmatrix} D_1 & F \\ \\ E & D_2 \end{pmatrix},$$

where D_1 and D_2 are diagonal square matrices.

□

1.14 ▪ M-Matrices and generalizations

A matrix A is positive (resp. strictly positive) and we denote $A \geq 0$ (resp. $A > 0$) if $a_{i,j} \geq 0$ (resp. $a_{i,j} > 0$) for all i and j.

Definition 1.37. *A matrix A is monotone if and only if A is nonsingular and $A^{-1} \geq 0$.*

This definition is the finite dimensional version of the maximum principle: if A is monotone and $b \geq 0$, then the solution of $Ax = b$ is such that $x \geq 0$.

Definition 1.38. *A is an L-matrix if and only if*

$$a_{i,i} > 0, \text{ for all } i, \qquad a_{i,j} \leq 0, \quad i \neq j.$$

From this definition, it is obvious that every L-matrix A can be written as $A = \sigma I - B$, with $\sigma \in \mathbb{R}$, $\sigma > 0$, and $B \geq 0$. Then, we have the following definition.

Definition 1.39. *Every matrix A which can be written as $A = \sigma I - B$ with $\sigma > 0, B \geq 0$ and such that $\rho(B) \leq \sigma$ is said to be an M-matrix.*

With this definition A can be singular. If A is nonsingular we have $\rho(B) < \sigma$. Many authors include A being nonsingular in the definition of an M-matrix. The following result relates the previous definition to the most frequently used definition of M-matrices.

Theorem 1.40. *A is a nonsingular M-matrix if and only if A is nonsingular with $a_{i,j} \leq 0$ for $i \neq j$ and $A^{-1} \geq 0$.*

Proof. Let A be a nonsingular M-matrix. Then,

$$A = \sigma I - B, \sigma \in \mathbb{R}, \quad \sigma > 0, \quad B \geq 0, \quad \rho(B) < \sigma.$$

We have to show that $A^{-1} \geq 0$. We have $\rho(\frac{B}{\sigma}) < 1$ and by Theorem 1.26 $\frac{1}{\sigma}A = I - \frac{B}{\sigma}$ is nonsingular. The Neumann series

$$I + \frac{B}{\sigma} + \frac{B^2}{\sigma^2} + \dots$$

converges and

$$\sigma A^{-1} = I + \frac{B}{\sigma} + \dots$$

Since $B \geq 0$ and $\sigma > 0$, we have $A^{-1} \geq 0$.

Conversely, let us assume that A is nonsingular with $a_{i,j} \leq 0$ for all i and j with $i \neq j$ and $A^{-1} \geq 0$. Then, we can write $A = \sigma I - B$ with $B \geq 0$ and $\sigma > 0$.

Let S_k be defined as

$$S_k = I + \frac{B}{\sigma} + \dots + \frac{B^{k+1}}{\sigma^{k+1}}.$$

Then,

$$\left(I - \frac{B}{\sigma}\right) S_k = I - \frac{B^{k+1}}{\sigma^{k+1}},$$

$$\sigma A^{-1} = \left(I - \frac{B}{\sigma}\right)^{-1} = S_k + \left(I - \frac{B}{\sigma}\right)^{-1} \frac{B^{k+1}}{\sigma^{k+1}}.$$

But $(I - \frac{B}{\sigma})^{-1} \geq 0$, $B^{k+1} \geq 0$ and $S_k \leq \sigma A^{-1}$. Every entry of S_k is a decreasing bounded sequence and so, it converges. The series $I + \frac{B}{\sigma} + \dots$ converges. By Theorem 1.26 it implies

$$\rho\left(\frac{B}{\sigma}\right) < 1,$$

which proves that A is an M-matrix. □

One can show that an M-matrix has positive diagonal entries and so is an L-matrix. Many characterizations of nonsingular M-matrices have been given. The book [123] by A. Berman and R.J. Plemmons gives 50 equivalent definitions. A sufficient condition that is often useful is given in the following theorem.

Theorem 1.41. *Let A be an irreducibly diagonally dominant L-matrix. Then, A is a nonsingular M-matrix.*

Proof. From Theorem 1.30 we know that A is nonsingular. Let D be a diagonal matrix such that $d_{i,i} = a_{i,i} > 0$ and let $B = I - D^{-1}A$. It is clear that $B \geq 0$,

$$D^{-1}A = I - B, \quad B \geq 0.$$

Theorem 1.30 shows that $\rho(B) < 1$. This proves that $D^{-1}A$ is an M-matrix and so is A. □

In fact, with the hypotheses of Theorem 1.41, we can prove that $A^{-1} > 0$. If A is a strictly diagonally dominant L-matrix, then A is a nonsingular M-matrix. Along the same lines we have the next theorem.

Theorem 1.42. *A is a nonsingular M-matrix if and only if $a_{i,j} \leq 0$ for all $i \neq j$ and A is generalized strictly diagonally dominant.*

Proof. Let A be generalized strictly diagonally dominant. Then, there exists a diagonal matrix D, $d_{i,i} > 0$ such that AD is strictly diagonally dominant. It is clear that the diagonal entries of AD are positive and the off-diagonal ones are negative. Therefore, AD is a nonsingular M-matrix and so is A.

Conversely, let us suppose that A is a nonsingular M-matrix. We have $a_{i,j} \leq 0$ for all $i \neq j$, $a_{i,i} > 0$ and $A^{-1} \geq 0$. Let e be such that $e^T = (1, \ldots, 1)$, and $d = A^{-1}e$. Then $d > 0$ because if we suppose there exists i such that $d_i = 0$ we have, $\sum_j (A^{-1})_{i,j} = 0$ which implies $(A^{-1})_{i,j} = 0$ for all j and hence, A^{-1} is singular. Let D be a diagonal matrix with diagonal entries d_i. Of course, $De = d$ so,

$$ADe = Ad = e > 0.$$

Componentwise, we have

$$a_{i,i}d_i + \sum_{\substack{j=1 \\ j\neq 1}}^{n} a_{i,j}d_j > 0.$$

But, $a_{i,j} \leq 0$ for all $i \neq j$ so $|a_{i,j}| = -a_{i,j}$. The previous inequality can be written as

$$|a_{i,i}|d_i > \sum_{j=1, j\neq i}^{n} |a_{i,j}|d_j,$$

which proves that A is generalized strictly diagonally dominant. □

Corollary 1.43. *The matrix A is a nonsingular M-matrix if and only if A is an L-matrix and there exists a diagonal matrix D with $d_{i,i} > 0$ such that $D^{-1}AD$ is strictly diagonally dominant.*

Proof. This comes from the remark following Proposition 1.34. □

The following result is a generalization for M-matrices of Theorem 1.32 for irreducibly diagonally dominant matrices.

Theorem 1.44. *Let A be a matrix with $a_{i,j} \leq 0$ for all $i \neq j$. Then, A is a nonsingular M-matrix if and only if $Re(\lambda) > 0$ for all eigenvalues λ of A.*

Proof. See A. Berman and R.J. Plemmons [123]. □

Lemma 1.45. *Let A be an M-matrix written in block form,*

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix},$$

where B and C are square matrices. Then, the Schur complement $S = C - EB^{-1}F$ is an M-matrix.

Proof. It is obvious that the principal submatrices of an M-matrix are M-matrices. Therefore, B is an M-matrix and $B^{-1} > 0$. Since, by definition, the entries of E and F are non-positive, the entries of $EB^{-1}F$ are non-negative. Therefore, the non-diagonal entries of S are non-positive.

Since A is an M-matrix, there is a diagonal matrix D with strictly positive diagonal entries such that AD is strictly diagonally dominant by row. Let

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix},$$

AD being strictly diagonally dominant means that if $e = (1 \ \ldots \ 1)^T$, then $ADe > 0$. But,

$$AD = \begin{pmatrix} BD_1 & FD_2 \\ ED_1 & CD_2 \end{pmatrix},$$

and let $e = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$. The Schur complement of AD is strictly diagonally dominant, (see [273]). This means that

$$0 < [CD_2 - ED_1(BD_1)^{-1}FD_2]e_2 = SD_2e_2.$$

This shows that S is generalized strictly diagonally dominant by row. Hence, S is an M-matrix. □

Let us consider a generalization of M-matrices. Given a matrix A, we define $M(A)$ as the matrix having entries $m_{i,j}$ such that

$$m_{i,i} = |a_{i,i}|, \quad m_{i,j} = -|a_{i,j}|$$

for all $i \neq j$. Clearly, $M(A)$ is an L-matrix. It is obvious that many different matrices A can lead to the same $M(A)$. So, we define the set

$$\Omega(A) = \{B \,|\, |b_{i,j}| = |a_{i,j}|\}.$$

$\Omega(A)$ is called the *equimodular set* for A. Then, we have the following definition.

Definition 1.46. *A is an H-matrix if and only if $M(A)$ is an M-matrix.*

Each definition of an M-matrix gives a corresponding definition for H-matrices. Let us recall the following result which will be useful later.

Theorem 1.47. *A is a nonsingular H-matrix if and only if A is generalized strictly diagonally dominant.*

□

It is obvious that strictly diagonally dominant, irreducibly diagonally dominant and nonsingular M-matrices are nonsingular H-matrices.

1.15 ▪ Splittings

Some iterative methods that we will study use a decomposition of the matrix A (which is also known as a *splitting*) in the form

$$A = M - N.$$

To prove some results about splittings, we have to be more specific about M and N.

Definition 1.48. *$A = M - N$ is a regular splitting if and only if M is nonsingular, $M^{-1} \geq 0$ and $N \geq 0$.*

Regular splittings and M-matrices are closely related as shown in the next result.

Theorem 1.49. *Let A be a matrix with $a_{i,j} \leq 0$ for all $i \neq j$. A is a nonsingular M-matrix if and only if there exists a regular splitting $A = M - N$ with $\rho(M^{-1}N) < 1$.*

Proof. A being an M-matrix, there exists $\sigma \in \mathbb{R}, \sigma > 0$ and $B \geq 0$ such that $A = \sigma I - B$ with $\rho(B) < \sigma$. This is clearly a regular splitting of A.

Conversely, assume we have a regular splitting $A = M - N$ with $\rho(M^{-1}N) < 1$. Then,

$$M^{-1}A = M^{-1}(M - N) = I - M^{-1}N$$

is nonsingular. By Theorem 1.26,

$$(I - M^{-1}N)^{-1} = I + M^{-1}N + (M^{-1}N)^2 + \cdots$$

Therefore, $(I - M^{-1}N)^{-1} \geq 0$ and $A^{-1} \geq 0$. □

It is sometimes useful to compare two regular splittings. This is done in the following theorem.

Theorem 1.50. *Let A be such that $A^{-1} \geq 0$ and consider two regular splittings of A, $A = M_1 - N_1$ and $A = M_2 - N_2$ with $N_2 \leq N_1$. Then,*

$$\rho(M_2^{-1}N_2) \leq \rho(M_1^{-1}N_1) < 1.$$

Proof. See R.S. Varga [1098]. □

Lemma 1.51. *Let $A = M - N$ be a regular splitting. The following statements are equivalent,*

1) $A^{-1} \geq 0$,

2) $A^{-1}N \geq 0$,

3) $\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1+\rho(A^{-1}N)} < 1$.

Proof. This was proved by R.S. Varga [1098]. □

Lemma 1.52. *Let $|A| \leq B$. Then, $\rho(A) \leq \rho(B)$.*

Proof. This is a consequence of the Perron-Frobenius theorem, see R.S. Varga [1098]. □

Lemma 1.53. *Let B be a nonsingular matrix and for all k, let (M_k, N_k) be a regular splitting of B. The following statements are equivalent*

1) $B^{-1} \geq 0$,

2) the sequence defined by $x_{k+1} = x_k - M_k^{-1}(Bx^k - c)$ tends to a solution of $Bx = c$.

Proof. See J. Moré [837]. □

Lemma 1.54. *With the same hypothesis as in Lemma 1.53, let $\widetilde{M}_k$ (nonsingular) and $\widetilde{N}_k$ such that*

$$|M_k^{-1}| \leq \widetilde{M}_k^{-1}, \quad |N_k| \leq \widetilde{N}_k,$$

and let $\widetilde{B} = \widetilde{M}_k - \widetilde{N}_k$ for all k. If the sequence defined by

$$x_{k+1} = x_k - \widetilde{M}_k^{-1}(\widetilde{B}x_k - c),$$

converges, then the sequence y_k such that

$$y_{k+1} = y_k - M_k^{-1}(By_k - c)$$

also converges.

Proof. Let $\widetilde{x}$ be the solution of $\widetilde{B}\widetilde{x} = c$, y be the solution of $By = c$, and $\epsilon_k = \widetilde{x} - x_k$, $\varepsilon_k = y - y_k$. Then,

$$\widetilde{M}_k \epsilon_{k+1} = \widetilde{N}_k \epsilon_k,$$
$$M_k \varepsilon_{k+1} = N_k \varepsilon_k.$$

Let us choose ε_0 such that $|\varepsilon_0| = \epsilon_0$, then

$$|\varepsilon_1| = |M_0^{-1} N_0 \varepsilon_0| \leq |M_0^{-1}||N_0||\varepsilon_0| \leq \widetilde{M}_0^{-1}\widetilde{N}_0\epsilon_0 = \epsilon_1.$$

By induction, we can show that

$$|\varepsilon_k| \leq \epsilon_k$$

But $\epsilon_k \to 0$ as $k \to \infty$, therefore $\varepsilon_k \to 0$. □

1.16 ▪ Positive definite matrices

Definition 1.55. *The matrix A is positive definite if and only if $(Ax, x) = x^T Ax > 0$ for all $x \neq 0$.*

These matrices are sometimes referred to as *positive real*. As we have seen above, if A is symmetric positive definite (SPD), we can define a vector norm $\|x\|_A$ by

$$\|x\|_A = (Ax, x)^{\frac{1}{2}}.$$

This is usually called the *energy norm* or the A-norm. For symmetric matrices we have a well known characterization of positive definiteness.

Proposition 1.56. *Let A be a symmetric matrix. Then, A is positive definite if and only if the eigenvalues of A (which are real numbers) are positive.*

Proof. This is easily proved by considering the spectral factorization of A , which is $A = X\Lambda X^H$. □

The following result for splittings is often called the Householder-John theorem. It is useful to prove the convergence of some iterative methods.

Theorem 1.57. *Let A be a nonsingular symmetric M-matrix such that $A = M - N$ with M nonsingular and $Q = M + M^T - A = M^T + N$ is positive definite. Then, $\rho(M^{-1}N) < 1$ if and only if A is positive definite.*

Proof. Let us assume that A is positive definite and let x_0 be given. We define a sequence of vectors by

$$Mx_{k+1} = Nx_k.$$

We have to prove that x_k converges to 0 as k goes to ∞, but,

$$x_{k+1} = M^{-1}Nx_k = (I - M^{-1}A)x_k.$$

Since A is positive definite,

$$\begin{aligned}\|x_{k+1}\|_A^2 &= (Ax_{k+1}, x_{k+1}),\\ &= ((A - AM^{-1}A)x_k, (I - M^{-1}A)x_k),\\ &= (Ax_k, x_k) - (AM^{-1}Ax_k, x_k)\\ &\quad -(Ax_k, M^{-1}Ax_k) + (AM^{-1}Ax_k, M^{-1}Ax_k).\end{aligned}$$

Let $y_k = M^{-1}Ax_k$. Then,

$$\|x_{k+1}\|_A^2 - \|x_k\|_A^2 = (Ay_k, y_k) - (My_k, y_k) - (y_k, My_k).$$

This holds because $(AM^{-1}Ax_k, x_k) = (M^{-1}Ax_k, Ax_k) = (y_k, My_k)$. Hence,

$$\|x_{k+1}\|_A^2 - \|x_k\|_A^2 = ((A - M^T - M)y_k, y_k) = -(Qy_k, y_k) < 0.$$

If $y_k = 0$, this implies $x_k = 0$ and then $x_{k+1} = 0$. Therefore, $\|x_k\|_A^2$ is a bounded decreasing sequence and it converges to some limit. Since we have

$$\|x_k\|_A^2 - \|x_{k+1}\|_A^2 = (Qy_k, y_k),$$

$(Qy_k, y_k) \to 0$ and then $y_k \to 0$ and $x_k \to 0$ which proves that $\rho(M^{-1}N) < 1$.

Conversely, let us assume that $\rho(M^{-1}N) < 1$ and that A is not positive definite. We have already shown that

$$(Ax_k, x_k) - (Ax_{k+1}, x_{k+1}) > 0.$$

If A is not positive definite, there exists x_0 such that $(Ax_0, x_0) < 0$. Then, by induction, the sequence x_k is such that

$$(Ax_{k+1}, x_{k+1}) < (Ax_k, x_k) < 0.$$

But this contradicts the fact that x_k tends to 0. □

We remark that the A-norms of iterates of the sequence x_k are decreasing. The previous theorem can be generalized to nonsymmetric matrices.

Theorem 1.58. *Let A be a nonsingular matrix $A = M - N$, with M nonsingular, such that $Q = M^T A^{-T} A + N$ is positive definite. Then, $\rho(M^{-1}N) < 1$ if and only if A is positive definite.*

Proof. See J.M. Ortega and R.J. Plemmons [884]. □

We now consider the relationship between M-matrices, H-matrices and positive definite matrices in the symmetric case.

Theorem 1.59. *Let A be a symmetric matrix with $a_{i,j} \leq 0$ for all $i \neq j$. Then A is a nonsingular M-matrix if and only if A is positive definite.*

Proof. The result follows directly from Theorem 1.44. □

Theorem 1.60. *Let A be a symmetric nonsingular H-matrix with $a_{i,i} > 0$ for all i. Then, A is positive definite.*

Proof. There exists a diagonal matrix D with $d_{i,i} > 0$ such that $D^{-1}AD$ is strictly diagonally dominant and $(D^{-1}AD)_{i,i} = a_{i,i} > 0$. From Theorem 1.32 $D^{-1}AD$ is positive definite. Since $D^{-1}AD$ has the same eigenvalues as A, this proves the result. □

The converse of this theorem is not true as shown by the following simple example.

Let

$$A = \begin{pmatrix} a & e & e \\ e & a & e \\ e & e & a \end{pmatrix}.$$

The eigenvalues of A are $a + 2e, a - e, a + e$. Let $a > e > 0$. Then, A is positive definite. Since

$$M(A) = \begin{pmatrix} a & -e & -e \\ -e & a & -e \\ -e & -e & a \end{pmatrix},$$

the eigenvalues of $M(A)$ are $a + e$, $a + e$ and $a - 2e$. If we choose $0 < e < a < 2e$, $M(A)$ is not positive definite and by Theorem 1.59 is not an M-matrix. This matrix A is an example of a positive definite matrix which is not an H-matrix.

The following result is important for the study of direct and iterative methods.

Lemma 1.61. *Let A be a symmetric positive definite matrix partitioned as*

$$A = \begin{pmatrix} A_{1,1} & A_{2,1}^{\mathrm{T}} \\ A_{2,1} & A_{2,2} \end{pmatrix},$$

where the blocks $A_{1,1}$ and $A_{2,2}$ are square. Then, the Schur complement,

$$S_{2,2} = A_{2,2} - A_{2,1} A_{1,1}^{-1} A_{2,1}^{\mathrm{T}},$$

is symmetric and positive definite.

Proof. This result can be proved in many different ways. Perhaps, the simplest one is to consider A^{-1} and to compute the bottom right-hand block of A^{-1}. Let

$$\begin{pmatrix} A_{1,1} & A_{2,1}^{\mathrm{T}} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Then,

$$A_{1,1}x_1 + A_{2,1}^{\mathrm{T}}x_2 = b_1 \quad \Rightarrow x_1 = A_{1,1}^{-1}(b_1 - A_{2,1}^{\mathrm{T}}x_2).$$

Therefore,

$$(A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{2,1}^{\mathrm{T}})x_2 = b_2 - A_{2,1}A_{1,1}^{-1}b_1.$$

The inverse of A can be written as

$$A^{-1} = \begin{pmatrix} X & Y \\ Z & S_{2,2}^{-1} \end{pmatrix}.$$

Since A is positive definite, the diagonal blocks of A and A^{-1} are also positive definite, and $S_{2,2}$ is positive definite. □

We also have the useful following result.

Lemma 1.62. *Let A be a symmetric positive definite matrix. Then,*

$$\max_{i,j} |a_{i,j}| = \max_{i}(a_{i,i}).$$

Proof. It is clear that the diagonal entries of A are positive. Suppose there are indices i_0, j_0 such that $|a_{i_0,j_0}| > |a_{i,j}|, i_0 \neq j_0, \ \forall i, j$ different from i_0, j_0. There are two cases:
i) assume $a_{i_0,j_0} > 0$, then let

$$x = (0 \ldots, 0, 1, 0, \ldots, 0, -1, 0, \ldots, 0)^T,$$

the 1 being in position i_0 and the -1 in position j_0. We have,

$$(x, Ax) = a_{i_0,i_0} + a_{j_0,j_0} - 2a_{i_0,j_0} < 0.$$

Therefore, A is not positive definite and we have a contradiction;
ii) assume $a_{i_0,j_0} < 0$, we choose

$$x = (0 \ldots, 0, 1, 0, \ldots, 0, 1, 0, \ldots, 0)^T,$$

and obtain

$$(x, Ax) = a_{i_0,i_0} + a_{j_0,j_0} + 2a_{i_0,j_0} < 0,$$

which, again, is a contradiction. □

1.17 ▪ The graph of a matrix

In the following chapters, we will be interested in matrices with many zero entries. Such matrices are called *sparse matrices*. For these matrices it is often useful to consider their graphs.

To a general nonsymmetric square matrix A of order n, we associate a directed graph (or digraph).

Definition 1.63. *A digraph is a couple $G = (X, E)$ where X is a set of n nodes (or vertices) and E is a set of directed edges.*

For a given matrix A, there is a directed edge from node i to node j if $a_{i,j} \neq 0$. Usually, self loops corresponding to $a_{i,i} \neq 0$ are not included. Let us consider a small example,

$$A = \begin{pmatrix} x & x & 0 & x \\ 0 & x & 0 & 0 \\ x & 0 & x & 0 \\ 0 & x & x & x \end{pmatrix}.$$

Then, the associated digraph is shown in Figure 1.10.

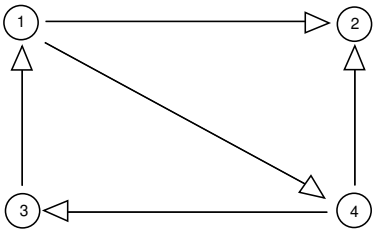


Figure 1.10. *A directed graph*

Graphs are more commonly used in problems involving symmetric matrices or matrices having a symmetric nonzero structure (also called a nonzero pattern). If $a_{i,j} \neq 0$, then $a_{j,i} \neq 0$. Therefore, we can consider only undirected graphs and drop the arrows on the edges. Let

$$A = \begin{pmatrix} x & x & x & 0 \\ x & x & 0 & x \\ x & 0 & x & x \\ 0 & x & x & x \end{pmatrix},$$

then, the graph of A is shown in Figure 1.11.

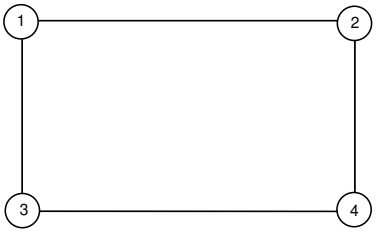


Figure 1.11. *An undirected graph*

Let $G = (X, E)$ be a (undirected) graph. We denote the nodes of the graph by x_i or sometimes i and consider some definitions:

• $G' = (X', E')$ is a *subgraph* of G if $X' \subset X$ and $E' \subset E$.

• Two nodes x and y of G are *adjacent* if $\{x, y\} \in E$. The *adjacency set* of a node y is defined as

$$Adj(y) = \{x \in X \mid x \text{ is adjacent to } y\}.$$

If $Y \subset X$, then

$$Adj(Y) = \{x \in X \mid x \in Adj(y), x \notin Y, y \in Y\}.$$

• The *degree* of a node x of G is the number of its adjacent nodes in G,

$$deg(x) = |Adj(x)|.$$

• Let x and $y \in X$. A *path* of length ℓ from x to y is a set of nodes $\{x_1, x_2, \ldots, x_{\ell+1}\}$ such that $x = x_1$, $y = x_{\ell+1}$ and $\{x_i, x_{i+1}\} \in E,\ 1 \leq i \leq \ell$. A path $\{x_0, x_1, \ldots, x_\ell, x_0\}$ is a (simple) *cycle* of length $\ell + 1$.

• A graph is *connected* if for every $x, y \in X$, there exists a path from x to y. This corresponds to the matrix being irreducible.

• A *chord* of a path is any edge joining two non-consecutive vertices in the path. A graph is *chordal* if every cycle of length greater than three has a chord, see [133].

• An important kind of graph is when there are no closed paths. A particular node is labeled as the root. Then, there is a path from any node to the root. Such a (connected) graph is called a *tree*. If it is not connected, we have a set of trees which is called a *forest*.

• Let $Y \subset X$, the *section graph* $G(Y)$ is a subgraph $(Y, E(Y))$ with

$$E(Y) = \{\{x, y\} \in E \mid x \in Y, y \in Y\}.$$

• A set $Y \subset X$ is a *separator* for G, a connected graph, if $G(X/Y)$ has two or more connected components.

• The *distance* $d(x, y)$ between two nodes x and y of G is the length of the shortest path between x and y. The *eccentricity* $e(x)$ of a node x is

$$e(x) = \max\{d(x, y) \mid y \in X\}.$$

The *diameter* δ of G is

$$\delta(G) = \max\{e(x) \mid x \in X\}.$$

A node x is *peripheral* if $e(x) = \delta(G)$.

• A *clique* is a subset of nodes which are all pairwise connected.

• A *level structure* of a graph G is a partition $\mathcal{L} = \{L_0, L_1, \ldots, L_\ell\}$ of X such that

$$\begin{aligned} &Adj(L_i) \subset L_{i-1} \cup L_{i+1},\ i = 1, \ldots, \ell - 1, \\ &Adj(L_0) \subset L_1, \\ &Adj(L_\ell) \subset L_{\ell-1}. \end{aligned}$$

Each $L_i, i = 1, \ldots, \ell - 1$ is a separator for G. For each node $x \in X$, a level structure $\mathcal{L}(x)$ can be defined as

$$\mathcal{L}(x) = \{L_0(x), \ldots, L_{e(x)}(x)\},$$

$$\begin{aligned} L_0(x) &= \{x\} \\ L_i(x) &= Adj(\cup_{k=0}^{i-1} L_k(x)),\ 1 \leq i \leq e(x) \end{aligned}$$

where $e(x)$ is the eccentricity of x. The width of a level structure $\mathcal{L}(x)$ is

$$w(x) = \max\{|L_i(x)|, 0 \leq i \leq e(x)\}.$$

1.18 ▪ Hessenberg and tridiagonal matrices

An upper Hessenberg matrix H is such that $h_{i,j} = 0$ for $i > 2$, $j = 1, \ldots, i-2$, that is, only the entries of the upper triangular part of H are (possibly) nonzero as well as the entries on the first subdiagonal. A lower Hessenberg matrix is a transposed upper Hessenberg matrix. A symmetric Hessenberg matrix is tridiagonal.

The matrix H of order n is *unreduced* (or *irreducible*) if $h_{j+1,j} \neq 0$, $j = 1, \ldots, n-1$, that is, the subdiagonal entries are nonzero. It implies that the matrix H is *nonderogatory*, which means that up to a scalar multiply the characteristic polynomial and the minimal polynomial are the same; see, for instance, [425] for a proof.

In some iterative methods we have to solve linear systems $H_k y = c$ where H_k is an unreduced upper Hessenberg matrix of order k. This can be done by reducing H_k to upper triangular form with Givens rotations to successively eliminate the subdiagonal entries. Let us assume that the current matrix is

$$\begin{pmatrix} x & x & \ldots & & x \\ & \ddots & & & \vdots \\ & & \ddots & & \\ & & & x & x \\ & & & 0 & r \\ & & & 0 & h \end{pmatrix}.$$

We have to eliminate the h entry to obtain an upper triangular matrix. This is done by left multiplying with the Givens matrix

$$\begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & c & -s \\ & & & s & c \end{pmatrix}.$$

The coefficients s and c, which are sine and cosine of the angle of rotation, are given by

$$c = \frac{r}{\sqrt{r^2 + h^2}}, \quad s = -\frac{h}{\sqrt{r^2 + h^2}}.$$

Let Q_k^T be the product of all the Givens rotation matrices for eliminating successively all the subdiagonal entries in rows 2 to k. Then, $Q_k^T H_k = R_k$ and the solution is obtained by solving the triangular system $R_k y = Q_k^T c$. To compute and use R_k we have to store and apply all the previous rotations to the right-hand side.

A particular kind of Hessenberg matrix is a *companion matrix*,

$$C = \begin{pmatrix} 0 & 0 & \ldots & 0 & -\alpha_0 \\ 1 & 0 & \ldots & 0 & -\alpha_1 \\ 0 & 1 & \ldots & 0 & -\alpha_2 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 & -\alpha_{n-1} \end{pmatrix}. \tag{1.3}$$

The matrix C is square of order n and nonderogatory. The eigenvalues of C are the roots of the monic polynomial p of degree n,

$$p(\lambda) = \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1 \lambda + \alpha_0.$$

Theorem 1.64. *An unreduced upper Hessenberg matrix H of order n can be factored as*

$$H = UCU^{-1}, \tag{1.4}$$

where U is a nonsingular upper triangular matrix of order n with $u_{1,1} = 1$ and diagonal entries $u_{k,k} = \prod_{j=1}^{k-1} h_{j+1,j}$ for $k > 1$ and C is the companion matrix of H, that is, corresponding to the characteristic polynomial p_C of H.

Proof. Define $U = (\, e_1 \quad He_1 \quad H^2e_1 \quad \dots \quad H^{n-1}e_1 \,)$. Then, U is upper triangular and nonsingular since H is unreduced and $u_{k,k} = \prod_{j=1}^{k-1} h_{j+1,j}$. Moreover, we have $HU = UC$ as it is easy to see for the first $n-1$ columns, and because of the Cayley-Hamilton theorem (which says that $p_C(H) = 0$) for the last column. Therefore, $H = UCU^{-1}$. This decomposition of H is unique. □

Inverses of unreduced upper Hessenberg matrices possess interesting properties. In [663] and [438], it was proved that the entries of the lower triangular part of H^{-1} are the product of components of two vectors, say x and y, $(H^{-1})_{i,j} = x_j y_i$; see also [55, 99]. The lower triangular part of H^{-1} is the lower triangular part of a rank-one matrix. This result about H^{-1} can be easily proved from the factorization (1.4) as follows.

Theorem 1.65. *Let $p_C(\lambda) = \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0$ be the characteristic polynomial of the unreduced Hessenberg matrix H factored as $H = UCU^{-1}$ as in Theorem 1.64. We define* $\beta_1 = -\alpha_1/\alpha_0$,

$$\hat{\beta} = -\left(\, -\alpha_2/\alpha_0 \quad \cdots \quad \alpha_{n-1}/\alpha_0 \quad 1/\alpha_0 \,\right)^T,$$

and we partition the upper triangular matrix U^{-1} as

$$U^{-1} = \begin{pmatrix} 1 & \hat{\vartheta}^T \\ 0 & \hat{U}^{-1} \end{pmatrix}.$$

Then, the inverse of H can be written as

$$H^{-1} = \begin{pmatrix} \beta_1 - \hat{\vartheta}^T\hat{U}\hat{\beta} & (\beta_1 - \hat{\vartheta}^T\hat{U}\hat{\beta})\hat{\vartheta}^T + (e_1^T - \hat{\vartheta}^T\hat{U}F)\hat{U}^{-1} \\ \hat{U}\hat{\beta} & \hat{U}\hat{\beta}\hat{\vartheta}^T + \hat{U}F\hat{U}^{-1} \end{pmatrix}, \tag{1.5}$$

where F is the zero matrix except for the entries on the first upper diagonal which are equal to 1.

Proof. We have obviously $H^{-1} = UC^{-1}U^{-1}$. With our notation the inverse of C can be written as

$$C^{-1} = \begin{pmatrix} \beta_1 & e_1^T \\ \hat{\beta} & F \end{pmatrix}.$$

The inverse of the matrix U^{-1} is

$$U = \begin{pmatrix} 1 & -\hat{\vartheta}^T\hat{U} \\ 0 & \hat{U} \end{pmatrix}.$$

We obtain the result by the multiplication $UC^{-1}U^{-1}$. □

The matrix $\hat{U}F\hat{U}^{-1}$ is strictly upper triangular. Therefore, the lower triangular part of the principal trailing block of H^{-1} is the lower triangular part of $\hat{U}\hat{\beta}\hat{\vartheta}^T$. We easily see that the lower triangular part of H^{-1} is the lower triangular part of a rank-one matrix.

From the previous result about H, we can obtain the factorization of all its principal matrices H_k.

Theorem 1.66. *Let $H = UCU^{-1}$ where U is nonsingular with $u_{1,1} = 1$ and upper triangular and C is a companion matrix. For $k < n$ the principal submatrix H_k can be written as $H_k = U_k C^{(k)} U_k^{-1}$, U_k being the leading principal submatrix of order k of U and the companion matrix of H_k is $C^{(k)} = E_k + (\,0 \quad U_k^{-1} U_{1:k,k+1}\,)$ where E_k is a square down-shift matrix of order k,*

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

Proof. Clearly,

$$H_k = (\,I_k \quad 0\,)\, H \begin{pmatrix} I_k \\ 0 \end{pmatrix} = (\,I_k \quad 0\,)\, UCU^{-1} \begin{pmatrix} I_k \\ 0 \end{pmatrix},$$

where I_k is the identity matrix of order k. Let

$$(\,I_k \quad 0\,)\, U = (\,U_k \quad Z\,), \quad U^{-1} \begin{pmatrix} I_k \\ 0 \end{pmatrix} = \begin{pmatrix} U_k^{-1} \\ 0 \end{pmatrix}.$$

Since $C = E_n + (\,0 \quad \cdots \quad 0 \quad v\,)$ where v is a given vector, we have

$$H_k = (\,U_k \quad Z\,) \begin{pmatrix} E_k U_k^{-1} \\ e_1 e_k^T U_k^{-1} \end{pmatrix} = U_k E_k U_k^{-1} + Z e_1 e_k^T U_k^{-1}.$$

The vector Ze_1 of length k is made of the k first components of the $k+1$st column of U, that is, $U_{1:k,k+1}$. Factoring U_k on the left and U_k^{-1} on the right gives the result. □

We will see later that the entries of the first row of U_k^{-1} are linked to the convergence of some iterative methods. They can be computed recursively from the entries of H.

Lemma 1.67. *The entries $\vartheta_{1,j}$ of the first row of the inverse of U in $H = UCU^{-1}$ are related to the entries $h_{i,j}$ of H by the relation*

$$\vartheta_{1,k+1} = -\frac{1}{h_{k+1,k}} \sum_{j=1}^{k} \vartheta_{1,j} h_{j,k}, \; k = 1, \ldots, n-1. \tag{1.6}$$

Proof. We have seen that $H = UCU^{-1}$ where C is the companion matrix defined in (1.3). Multiplying on the left by U^{-1}, we have $U^{-1}H = CU^{-1}$. From the structure of C and the fact that U^{-1} is upper triangular, the entries of the first row of CU^{-1} are zero except for the last one in position $(1, n)$. Writing the entry $(1, k)$ of $U^{-1}H$ for $k < n$, we obtain

$$\sum_{j=1}^{k+1} \vartheta_{1,j} h_{j,k} = 0 \;\Rightarrow\; \vartheta_{1,k+1} = -\frac{1}{h_{k+1,k}} \sum_{j=1}^{k} \vartheta_{1,j} h_{j,k},$$

which proves the claim. □

We can specialize some of the previous results to symmetric tridiagonal matrices.

Let

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}, \tag{1.7}$$

where the values β_j, $j = 2, \ldots, n$ are assumed to be nonzero. From [97] or [809] and the references therein, as well as Theorem 1.65, we know that there exist two sequences of nonzero numbers $\{\nu_i\}, \{\sigma_i\}, i = 1, \ldots, n$ such that

$$T^{-1} = \begin{pmatrix} \nu_1\sigma_1 & \nu_1\sigma_2 & \nu_1\sigma_3 & \ldots & \nu_1\sigma_n \\ \nu_1\sigma_2 & \nu_2\sigma_2 & \nu_2\sigma_3 & \ldots & \nu_2\sigma_n \\ \nu_1\sigma_3 & \nu_2\sigma_3 & \nu_3\sigma_3 & \ldots & \nu_3\sigma_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \nu_1\sigma_n & \nu_2\sigma_n & \nu_3\sigma_n & \ldots & \nu_n\sigma_n \end{pmatrix}. \tag{1.8}$$

Moreover, ν_1 can be chosen as $\nu_1 = 1$. The strictly triangular part corresponds to what we have seen for the inverse of the upper Hessenberg matrix H. Even though this seems implied by the notation, we observe that the matrix T^{-1} is nonsingular if and only if $\nu_i \neq 0, i = 1, \ldots, n$ and $\nu_i\sigma_{i+1} - \nu_{i+1}\sigma_i \neq 0, i = 1, \ldots, n-1$.

We can relate the entries of T^{-1} to those of T by identification in the relation $T^{-1}T = I$,

$$\alpha_1 = -\frac{\nu_2}{\sigma_2 - \nu_2\sigma_1}, \quad \beta_2 = \frac{1}{\sigma_2 - \nu_2\sigma_1} = -\frac{\alpha_1}{\nu_2},$$

and for $i = 2, \ldots, n-1$, let

$$\begin{aligned} \chi_i &= \nu_i(\nu_i\sigma_{i+1} - \nu_{i+1}\sigma_i), \\ \alpha_i &= -\frac{\nu_{i+1}}{\chi_i} - \beta_i\frac{\nu_{i-1}}{\nu_i}, \quad \beta_{i+1} = \frac{\nu_i}{\chi_i}, \end{aligned} \tag{1.9}$$

the last diagonal entry being equal to

$$\alpha_n = \frac{1}{\nu_n\sigma_n} - \beta_n\frac{\nu_{n-1}}{\nu_n}.$$

Of course, we have to assume that $\nu_i \neq 0$ for $i = 1, \ldots, n$ and $\chi_i \neq 0$ for $i = 1, \ldots, n-1$. It can be easily proved that $\nu^T T e_i = 0$ for $i = 1, \ldots, n-1$, where $\nu^T = \begin{pmatrix} \nu_1 & \nu_2 & \cdots & \nu_n \end{pmatrix}$ and e_i is the ith column of the identity matrix. As a consequence, $\nu^T U = e_1^T$, where $T = UCU^{-1}$.

We can see that the last column of T^{-1} is proportional to the transpose of the first row of U^{-1}. Since $T = UCU^{-1}$ is symmetric as well as T^{-1},

$$T^{-1} = T^{-T} = U^{-T}C^{-T}U^T.$$

The last column of U^T is $u_{n,n}e_n$. The last column of C^{-T} is $-(1/\alpha_0)e_1$, where α_0 is the coefficient of the characteristic polynomial of T, that is, the determinant of T. Finally, we obtain

$$T^{-1}e_n = -\frac{u_{n,n}}{\det(T)}U^{-T}e_1,$$

which proves the result.

For $k < n$, we can characterize the inverses of the principal matrices T_k of T. Let us start with T_{n-1} which is obtained from T by removing the last row and the last column. Let $\mathcal{B}$ be a nonsingular matrix of order n, partitioned as

$$\mathcal{B} = \begin{pmatrix} B & a \\ c^T & \alpha \end{pmatrix},$$

with B nonsingular. We would like to make a rank-one modification to $\mathcal{B}$ to find the inverse B^{-1}. Let $\mathcal{B}_{:,n}$ be the last column of $\mathcal{B}$. We define two vectors

$$u = \mathcal{B}_{:,n} - e_n = \begin{pmatrix} a \\ \alpha - 1 \end{pmatrix}, \quad v = e_n,$$

and the rank-one modification,

$$\mathcal{B} - uv^T = \begin{pmatrix} B & 0 \\ c^T & 1 \end{pmatrix}.$$

Clearly, if we are able to compute the inverse of $\mathcal{B} - uv^T$, we obtain the inverse of B. The inverse of the rank-one modification is

$$(\mathcal{B} - uv^T)^{-1} = \mathcal{B}^{-1} + \frac{1}{1 - v^T\mathcal{B}^{-1}u}\mathcal{B}^{-1}uv^T\mathcal{B}^{-1},$$

according to the Sherman-Morrison formula, see [548] and Section 1.19. First, we have

$$v^T\mathcal{B}^{-1}u = e_n^T\mathcal{B}^{-1}(\mathcal{B}_{:,n} - e_n) = e_n^Te_n - e_n^T\mathcal{B}^{-1}e_n = 1 - e_n^T\mathcal{B}^{-1}e_n.$$

It yields

$$1 - v^T\mathcal{B}^{-1}u = e_n^T\mathcal{B}^{-1}e_n = [\mathcal{B}^{-1}]_{n,n}.$$

$v^T\mathcal{B}^{-1}$ is the last row of $\mathcal{B}^{-1}$ that we denote as $[\mathcal{B}^{-1}]_{n,:}$. The other vector is

$$\mathcal{B}^{-1}u = \mathcal{B}^{-1}(\mathcal{B}_{:,n} - e_n) = e_n - [\mathcal{B}^{-1}]_{:,n}.$$

Therefore, the inverse of the rank-one modification is

$$(\mathcal{B} - uv^T)^{-1} = \mathcal{B}^{-1} + \frac{1}{[\mathcal{B}^{-1}]_{n,n}}\left[e_n - [\mathcal{B}^{-1}]_{:,n}\right]\left[[\mathcal{B}^{-1}]_{n,:}\right].$$

If we know the inverse of $\mathcal{B}$, by taking the indices (i, j) such that $1 \le i, j \le n-1$ in the right-hand side, we obtain the inverse of B. Note that the last term on the right-hand side depends only on the last row and last column of $\mathcal{B}^{-1}$.

Lemma 1.68. *Let T be the tridiagonal matrix whose inverse is defined by (1.8). The inverse of T_{n-1} is*

$$T_{n-1}^{-1} = \begin{pmatrix} \sigma_1^{(n-1)} & \sigma_2^{(n-1)} & \cdots & \sigma_{n-1}^{(n-1)} \\ \sigma_2^{(n-1)} & \sigma_2^{(n-1)}\nu_2 & \cdots & \sigma_{n-1}^{(n-1)}\nu_2 \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n-1}^{(n-1)} & \sigma_{n-1}^{(n-1)}\nu_2 & \cdots & \sigma_{n-1}^{(n-1)}\nu_{n-1} \end{pmatrix}.$$

with

$$\sigma_i^{(n-1)} = \sigma_i - \sigma_n\frac{\nu_i}{\nu_n},\ i = 1, 2, \ldots, n-1. \tag{1.10}$$

Proof. We apply the previous results with $\mathcal{B} = T$. We start with the first column. We have

$$\begin{pmatrix} \sigma_1^{(n-1)} \\ \sigma_2^{(n-1)} \\ \vdots \\ \sigma_{n-1}^{(n-1)} \end{pmatrix} = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_{n-1} \end{pmatrix} - \frac{1}{\nu_n} \begin{pmatrix} \sigma_n \\ \sigma_n\nu_2 \\ \vdots \\ \sigma_n\nu_{n-1} \end{pmatrix}.$$

It yields relation (1.10). Now, we consider the last column of T_{n-1}^{-1},

$$\sigma_{n-1}^{(n-1)} \begin{pmatrix} 1 \\ \nu_2^{(n-1)} \\ \vdots \\ \nu_{n-1}^{(n-1)} \end{pmatrix} = \sigma_{n-1} \begin{pmatrix} 1 \\ \nu_2 \\ \vdots \\ \nu_{n-1} \end{pmatrix} - \frac{\nu_{n-1}}{\nu_n} \begin{pmatrix} \sigma_n \\ \sigma_n\nu_2 \\ \vdots \\ \sigma_n\nu_{n-1} \end{pmatrix}.$$

Hence,

$$\sigma_{n-1}^{(n-1)}\nu_i^{(n-1)} = \sigma_{n-1}\nu_i - \sigma_n\frac{\nu_{n-1}}{\nu_n}\nu_i, \quad 1 \le i \le n-1.$$

Using the value of $\sigma_{n-1}^{(n-1)}$, we obtain

$$\sigma_{n-1}^{(n-1)}\nu_i^{(n-1)} = \sigma_{n-1}\nu_i - \sigma_n\frac{\nu_{n-1}}{\nu_n}\nu_i = \left(\sigma_{n-1} - \sigma_n\frac{\nu_{n-1}}{\nu_n}\right)\nu_i^{(n-1)}.$$

If $\sigma_{n-1}^{(n-1)} \neq 0$, it yields $\nu_i^{(n-1)} = \nu_i$, for all $1 \le i \le n-1$. □

To obtain the inverse of T_k, with $k = 1, \ldots, n-2$ we apply Lemma 1.68 recursively.

Theorem 1.69. *Let T be the tridiagonal matrix whose inverse is defined by (1.8). The inverse of T_k for $k = 1, \ldots, n-1$ is*

$$T_k^{-1} = \begin{pmatrix} \sigma_1^{(k)} & \sigma_2^{(k)} & \cdots & \sigma_k^{(k)} \\ \sigma_2^{(k)} & \sigma_2^{(k)}\nu_2 & \cdots & \sigma_k^{(k)}\nu_2 \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_k^{(k)} & \sigma_k^{(k)}\nu_2 & \cdots & \sigma_k^{(k)}\nu_k \end{pmatrix}.$$

with

$$\sigma_i^{(k)} = \sigma_i^{(k+1)} - \sigma_{k+1}^{(k+1)}\frac{\nu_i}{\nu_{k+1}}, \ i = 1, 2, \ldots, k. \tag{1.11}$$

Moreover,

$$\sigma_i^{(k)} = \sigma_i - \sigma_{k+1}\frac{\nu_i}{\nu_{k+1}}, \ i = 1, 2, \ldots, k. \tag{1.12}$$

Proof. Relation (1.11) is obtained by applying Lemma 1.68 to T_{k+1}. To prove relation (1.12), we proceed by induction from $k = n-1$ to $k = 1$. From Lemma 1.68, the result holds for $k = n-1$. Let us assume that it holds for $k+1$. Then, for $i = 1, \ldots, k$,

$$\begin{aligned} \sigma_i^{(k)} &= \sigma_i^{(k+1)} - \sigma_{k+1}^{(k+1)}\frac{\nu_i}{\nu_{k+1}}, \\ &= \sigma_i - \sigma_{k+2}\frac{\nu_i}{\nu_{k+2}} - \left(\sigma_{k+1} - \sigma_{k+2}\frac{\nu_{k+1}}{\nu_{k+2}}\right)\frac{\nu_i}{\nu_{k+1}}, \\ &= \sigma_i - \sigma_{k+1}\frac{\nu_i}{\nu_{k+1}}. \end{aligned}$$

□

1.19 ▪ Sherman-Morrison and Cauchy-Binet formulas

The *Sherman-Morrison formula* (see [100, 1005]) shows what is the inverse of a rank-one modification of a nonsingular square matrix A of order n. Let u and v be two vectors with n components. The matrix $A + uv^T$ is nonsingular if and only if $1 + v^T A^{-1} u \neq 0$ and

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1} uv^T A^{-1}}{1 + v^T A^{-1} u}. \tag{1.13}$$

This has been generalized to the *Sherman-Morrison-Woodbury formula* (see [1122]). Let U, V be two matrices of arbitrary rank r, $r \leq n$, and let C be nonsingular such that the product UCV exists and is square of order n. Then,

$$(A + UCV)^{-1} = A^{-1} - A^{-1} U (C^{-1} + VA^{-1}U)^{-1} VA^{-1}. \tag{1.14}$$

The *Cauchy-Binet formula* (see, for instance, [494]) is a relation giving the determinant of the product of two rectangular matrices of compatible dimension. Let $k \leq n$, A be $k \times n$ and B be $n \times k$. Then,

$$\det(AB) = \sum_{\mathcal{I}_k} \det(A_{:,\mathcal{I}_k}) \det(B_{\mathcal{I}_k,:}), \tag{1.15}$$

where the summation is over all the ordered subsets $\mathcal{I}_k$ of k integers, $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. Note that if $A = B^*$, the conjugate transpose of B, we have

$$\det(B^* B) = \sum_{\mathcal{I}_k} |\det(B_{\mathcal{I}_k,:})|^2.$$

1.20 ▪ Chebyshev polynomials

In this section we review some results about Chebyshev polynomials that will be useful in studying some iterative methods.

Definition 1.70. *The Chebyshev polynomials (of the first kind) C_n are defined for an integer n and $x \in \mathbb{R}$, $|x| \leq 1$, by*

$$C_n(x) = \cos(n \arccos x).$$

This definition can be extended to an interval $[a, b]$ by setting $t = \frac{a+b}{2} + \frac{b-a}{2} x$. Then, $t \in [a, b] \Leftrightarrow x \in [-1, +1]$.

From this definition, it is not immediately obvious that C_n is a polynomial. This follows from the properties in the following theorem.

Theorem 1.71. *Let C_n be a Chebyshev polynomial. Then,*

1) $C_0(x) = 1$, $C_1(x) = x$, $C_{n+1}(x) = 2xC_n(x) - C_{n-1}(x)$.

2) For $n \geq 1$, C_n is a polynomial of degree n whose leading coefficient is 2^{n-1}.

3) $C_n(-x) = (-1)^n C_n(x)$.

4) C_n has n zeros in $[-1, +1]$, namely

$$x_k = \cos\left(\frac{2k+1}{n} \frac{\pi}{2}\right), \quad k = 0, 1, \ldots, n-1,$$

and $n+1$ *extremas*

$$x'_k = \cos\left(\frac{k\pi}{n}\right) \quad \textit{with} \quad C_n(x'_k) = (-1)^k, \quad k = 0, 1, \ldots, n.$$

5) The Chebyshev polynomials are orthogonal with respect to the dot product

$$((f,g)) = \int_{-1}^{+1} \frac{f(x)g(x)}{\sqrt{1-x^2}}\, dx.$$

Moreover,

$$((C_i, C_j)) = \begin{cases} 0 & i \neq j \\ \frac{\pi}{2} & i = j \neq 0 \\ \pi & i = j = 0 \end{cases}$$

6) For all polynomials of degree n *with leading coefficient* 1, $C_n/2^{n-1}$ *has the smallest maximum norm, namely* $1/2^{n-1}$.

Proof. See, for instance, G. Dahlquist and Å. Björck [302]. □

The most interesting property of Chebyshev polynomials for studying iterative methods is that they have the smallest maximum norm (item 6). It helps in solving the following problem:

Let $\pi_n^1 = \{$ polynomials of degree n in t whose value is 1 at the origin $\}$. We will frequently require the solution of the minimization problem

$$\min_{Q_n \in \pi_n^1} \max_{t \in [a,b]} |Q_n(t)|.$$

A solution to this problem is given by the shifted and normalized Chebyshev polynomial

$$\min_{Q_n \in \pi_n^1} \max_{t \in [a,b]} |Q_n(t)| = \max_{t \in [a,b]} \left| \frac{C_n\left(\frac{2t-(a+b)}{b-a}\right)}{C_n\left(\frac{a+b}{b-a}\right)} \right| = \frac{1}{\left|C_n\left(\frac{a+b}{b-a}\right)\right|}.$$

This solution is of interest because we know the roots of Q_n,

$$\tau_\ell = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2\ell-1}{n}\frac{\pi}{2}\right) \quad \ell = 1, \ldots, n.$$

If $a = -b$, then $C_n\left(\frac{a+b}{b-a}\right) = 1$. In more general cases we need an upper bound for

$$\frac{1}{\left|C_n\left(\frac{a+b}{b-a}\right)\right|}.$$

We use the following characterization.

Lemma 1.72.

$$C_n(x) = \frac{1}{2}[(x + \sqrt{x^2-1})^n + (x - \sqrt{x^2-1})^n], \quad |x| \leq 1.$$

Proof. Let $\varphi = \arccos(x)$, so $C_n(x) = \cos(n\varphi)$. We can write

$$\cos(n\varphi) = \frac{1}{2}(\cos(n\varphi) + i\sin(n\varphi) + \cos(n\varphi) - i\sin(n\varphi)),$$

with $i = \sqrt{-1}$. Therefore,

$$\cos(n\varphi) = \frac{1}{2}[(\cos(\varphi) + i\sin(\varphi))^n + (\cos(\varphi) - i\sin(\varphi))^n],$$

but,

$$x = \cos(\varphi) \text{ and } i\sin(\varphi) = \sqrt{-\sin(\varphi)^2} = \sqrt{x^2 - 1},$$

proving the result. □

Using Lemma 1.72, we obtain the bound we are looking for.

Theorem 1.73. *If* $0 < a < b$ *then*

$$\min_{Q_n \in \pi_n^1} \max_{t \in [a,b]} |Q_n(t)| \leq 2\left(\frac{1 - \sqrt{\frac{a}{b}}}{1 + \sqrt{\frac{a}{b}}}\right)^n = 2\left(\frac{\sqrt{\frac{b}{a}} - 1}{\sqrt{\frac{b}{a}} + 1}\right)^n.$$

Proof. We have $x = \frac{b+a}{b-a}$, so

$$\begin{aligned} x^2 - 1 &= \frac{2\sqrt{ab}}{b-a}, \\ x + \sqrt{x^2 - 1} &= \frac{(\sqrt{b} - \sqrt{a})^2}{b-a}, \\ &= \frac{\sqrt{b} + \sqrt{a}}{\sqrt{b} - \sqrt{a}}, \end{aligned}$$

and

$$x - \sqrt{x^2 - 1} = \frac{\sqrt{b} - \sqrt{a}}{\sqrt{b} + \sqrt{a}}.$$

Therefore,

$$C_n\left(\frac{b+a}{b-a}\right) \geq \frac{1}{2}\left(\frac{\sqrt{b} + \sqrt{a}}{\sqrt{b} - \sqrt{a}}\right)^n.$$

□

1.21 ▪ Computer architecture

Let us very briefly review the history of computers. For more details, see Chapter 7 of [160]. Since the first few years after World War II with the advent of the first digital computers with stored programs, there has been tremendous progress in the floating-point performance of scientific computers.

In the early years of computers, people have considered the number of floating-point operations involved in their algorithms as a measure of efficiency. This was supposed to give a good account of what is the computing time used when running the codes. A nice feature was that this operation count was computer independent. But we cannot use the same technique anymore on today's parallel computers.

Up to the beginning of the 1970s, computers can be considered as serial machines. Roughly speaking, one can say that these computers execute basic operations in a serial way, waiting for the completion of an operation before beginning the execution of the next one. For this kind of

computer, that is to say the ones people used in the 1950s, 1960s and the beginning of the 1970s, the operation count gives a good idea of what is going on.

This was already no longer true for some computers that were introduced in the beginning of the 1970s, for example, the CDC 7600. The 7600 was a pipelined computer. This means that a basic operation, like addition or multiplication, is divided into a certain number of steps, say n. When one pair of operands has moved from step 1 to step 2, step 1 is free to accept a new pair of operands. Of course, the second pair must be independent of the results of the first operation. If we (or the compiler) are able to continuously feed the pipeline with independent operations then the first result will take n cycles to be produced but after that we must get a new result every cycle. Unfortunately, at that time the available compilers were not smart enough to automatically recognize that they had to do consecutive independent operations.

The 1970s saw the advent of vector computers, the first one commercially successful being the CRAY-1. Among other features, the CRAY-1 had 8 vector registers of 64 words each. The compiler was able to recognize that we were doing, for example, the addition of two one-dimensional arrays, that is, two vectors. It was then able by using vector instructions to feed the addition pipeline efficiently by fetching the operands from the memory into the vector registers. Moreover, this machine had independent functional units that can operate in parallel, of course on non-related data.

It is clear that with this kind of computer, the classical operation count means nothing. On typical codes, there was a ratio of 10 to 40 between scalar and vectorized codes. To run fast, an algorithm had to be expressed (if possible) in terms of vectors. The memory traffic issue was also very important in order to get good performance. The fastest vector computers were those with the largest memory bandwidth and the speed of computation depended also on the ratio of the number of floating-point operations to the number of memory references.

The next step to get more computer power was to use multiprocessor computers. Vector computers evolved into machines with several vector processors sharing a large memory. Examples of these machines were the CRAY Y-MPs, as well as computers from Japanese manufacturers like Fujitsu, NEC or Hitachi. Most of these machines were running by the mid-1190s at Gigaflops (10^9 floating-point operations per second) speeds on well designed programs.

At the end of the 1980s and beginning of the 1990s a new kind of scientific computers appeared on the market: parallel computers with a distributed memory. Almost all these machines used off-the-shelf microprocessors rather than proprietary designs. The rise of the microprocessors was the demise of the vector processors. These new machines had from a few tens to a few thousand processors. Teraflops (10^{12} flops) speeds were reached on some applications.

Since the increase of the computational speed of microprocessors was slowing down, at the beginning of the 2000s manufacturers started to increase the number of processors in reach of the Petaflops (10^{15} flops). Moreover, several computing engines, named as "cores" were put on a single chip. In 2011, a Blue Gene/Q IBM computer using 18-core chips with 1.47 billion transistors had a peak performance of 20 Petaflops. It reached 17 Petaflops on the Linpack benchmark in the TOP500 list.

To increase the performance again in reach of the Exaflops (10^{18} flops), computer designers turned to another type of computing engines known as general-purpose graphics processing units (GPGPU). A graphics processing unit (GPU) is a type of specialized processor for the fast creation and manipulation of images for output on a display device. They were first developed in the 1970s and 1980s. They were in particular used for video games. With time, they became more sophisticated and efficient, also being able to do floating-point arithmetic. The GPUs use a form of parallelism which may somehow be assimilated to vectorization. They are now used in many supercomputers which become hybrid machines with traditional multicore processors and GPUs. These computing devices also use 16-bit floating-point arithmetic which was useful for the applications they were developed for. Since the computations are faster with 16 bits rather

than with single or double precision and this also reduce the communication times, it attracted the attention of numerical analysts. In recent years we saw a flowering of so-called *mixed precision* algorithms.

Parallel computers put more constraints on numerical analysts and programmers. It seems obvious that to obtain good performances on a parallel computer, the program implementing an algorithm must have as many operations as possible that can be executed in parallel. Sometimes only a few modifications have to be made to classical algorithms to use them on parallel computers. But, most of the time, new algorithms have to be derived. This is not so easy to achieve, for instance, in algorithms for solving linear systems. By definition some or all the unknowns of a linear system are coupled, except if the matrix is diagonal in which case the solution is trivially obtained. In distributed memory computers these algorithms have to exchange data between the processors' memories. This is often the bottleneck of the computation. Quite often it is also necessary to compute the dot product of two vectors. If the multiplication of the components is parallel, the summation is a reduction operation that can be costly. This is why in recent years, researchers have been looking for algorithms minimizing the communications. They are sometimes called *communication-avoiding* algorithms, even though some communications are always needed when solving linear systems.

Finding efficient and robust parallel algorithms is a serious challenge for numerical analysts. Another important issue that has to be considered is scalability. The most powerful scientific computers are designed to solve large problems. The size of the problems that scientists and engineers want to solve is, and probably will be, always too large for the available computers. Nevertheless, as the size of the problem grows we would like, if we increase proportionally the number of cores, to keep the (elapsed) computer time constant. To obtain this, we need to have algorithms whose serial complexity is proportional to the number of unknowns. Unfortunately, most of the well known algorithms are not scalable.

Figure 1.12 displays the performance (in Tflops) of the fastest computer in the TOP500 list up to June 2021. On the x-axis, one unit corresponds to six months. Note that we have a logarithmic scale on the y-axis.

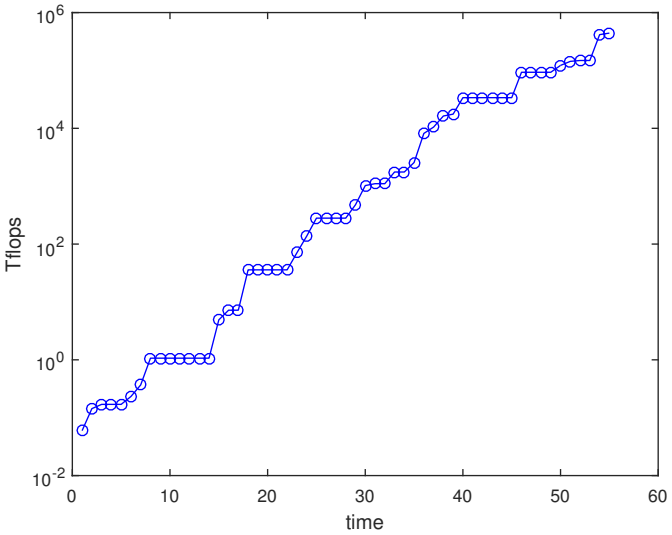


Figure 1.12. *Performance of the fastest computer in the TOP500 HPL list as a function of time*

During all the years since World War II, researchers in numerical linear algebra had to face many challenges like fixed-point arithmetic, floating-point arithmetic, different and sometimes weird arithmetics, rounding errors, portability issues, vectorization of algorithms, cache-aware algorithms, parallelism, communication-avoiding algorithms, mixed precision algorithms and so on. But, we must note that, except for the definition of the IEEE 754 standard for floating-point arithmetic, numerical analysts did not have much influence on the architecture and design of computers. They have to deal with what they get.

1.22 ▪ Historical and bibliographical comments

Many of the results given in this chapter can be found in the classical books of A.S. Householder [647], J.H. Wilkinson [1120], R.S. Varga [1098], D.M. Young [1144] and G. Strang [1054].

For discretization methods, one can see, for instance, R.J. LeVeque [731] for finite differences and P.G. Ciarlet [264] for finite elements.

The design of the IEEE standard for floating-point arithmetic was largely influenced by W.M. Kahan. He received the Turing Award in 1989 for his work. On this topic, see D. Golberg [537], M.L. Overton , N.J. Higham [631, 633], the handbook [846], or the norm IEEE-754 itself.

Norms of vectors and matrices were not much used in numerical linear algebra before the 1940s. One can find them in papers by J. von Neumann and H.H. Goldstine [1105] in 1947 and by A.M. Turing [1078] in 1948. Later on, they were studied intensively by A.S. Householder and F.L. Bauer [648, 649, 104, 105]. Properties of matrix norms and their relations to eigenvalues were studied by A.M. Ostrowski [888] in 1955.

The Gram-Schmidt orthogonalization is named after J.P. Gram [554] in 1883 and E. Schmidt [994, 995, 997, 996] in 1907-1908. It was used later on to orthogonalize sets of vectors.

The Schur factorization originated from I. Schur [999] in 1909.

The singular value decomposition is due to E. Beltrami in 1873, C. Jordan in 1874, and J.J. Sylvester in 1889 The name "singular value" was introduced by H. Bateman in 1908 and E. Picard in 1909.

Theorem 1.25, giving a necessary and sufficient condition for convergence of a matrix power sequence, goes back to K.W. Hensel in 1926 and R. Oldenburger in 1940. Most of the proofs in the literature use the Jordan canonical form.

Irreducibility of a matrix was introduced by F.G. Frobenius in 1912 and reintroduced by H. Geringer in 1949.

Diagonally dominant matrices have been studied for a very long time. Corollary 1.31 saying that a strictly diagonally dominant matrix is nonsingular was proved by many mathematicians. However, its origin seems to be in the works of L. Levy in 1881, J. Desplanques in 1887, and J. Hadamard in 1898. This result was then forgotten and rediscovered by S. Gerschgorin in 1931. Olga Taussky proved in 1949 that a strictly diagonally dominant matrix has eigenvalues with positive real parts.

A.M. Ostrowski introduced M- and H-matrices in 1937. It is very difficult to trace where all the characterizations of M-matrices come from. The book [123] by A. Berman and R.J. Plemmons is a good summary of all these results.

Theorem 1.57 originated from A.S. Householder in 1955 and F. John in 1956. It is an extension of the Ostrowski-Reich theorem (1949-1954).

The use of graphs for Gaussian elimination is due to S.V. Parter [906] in 1961 and D.J. Rose [951] in 1970; see also F. Harary [616] in 1959 and [617] in 1971.

Hessenberg matrices were introduced in the work of K. Hessenberg [624, 625] on the computation of eigenvalues in 1940. The characterization of the lower triangular part of inverses of Hessenberg matrices was published by Y. Ikebe [663] in 1979 and by D.K. Faddeev in 1981 in Russian and in 1984 in English [438]. An early paper considering the explicit inverse of tridiagonal matrices is by D. Moskovitz [845] in 1944. An important paper for the inverses of band matrices was the seminal work by E. Asplund [55] in 1959. In 1969, the physicist C.F. Fischer and her student R.A. Usmani gave a general analytical formula for inverses of symmetric Toeplitz tridiagonal matrices [458]. In 1971, J. Baranger and M. Duc-Jacquet considered symmetric factorizable matrices (whose elements are $a_i b_j$ for $i \leq j$) and proved that the inverse is tridiagonal (this is Asplund's result) and conversely [97]. For a review about inverses of tridiagonal and block tridiagonal matrices, see [809] in 1992.

2

Gaussian elimination for general linear systems

The two main classes of rounding error analysis are not, as my audience might imagine, 'backwards' and 'forwards', but rather 'one's own' and 'other people's'. One's own is, of course, a model of lucidity; that of others serves only to obscure the essential simplicity of the matter in hand.

– J.H. Wilkinson, *NAG 1984 Annual General Meeting*

In this chapter we consider elimination methods for solving linear systems with general matrices. This means that the matrices are generally stored as a two-dimensional array without any particular assumption about the value of the entries. Elimination methods correspond to a factorization of the matrix as the product of lower an upper triangular matrices. This is why we start by considering triangular systems. Then, we turn to general systems and LU factorization. A particular attention is given to symmetric systems and then to special classes of matrices. After discussing rounding error analyses, we consider elimination methods on parallel computers. We end this chapter with some variants of Gaussian elimination.

2.1 ▪ Triangular systems

The easiest linear systems to solve are those with a diagonal matrix. All the equations are independent and the solution is trivially obtained by n divisions. Of course, if the matrix is diagonal we cannot really speak of a linear system. Among linear systems whose solution is easily obtained, triangular systems are of particular interest since we will see that they are involved in solving general linear systems. A matrix L is *lower triangular* is all the entries above the main diagonal are zero, that is, $\ell_{i,j} = 0$ for $j > i$. The matrix L is nonsingular if and only if all the diagonal entries are nonzero and the determinant of L is the product of the diagonal entries. An *upper triangular matrix* U is the transpose of a lower triangular matrix.

A linear system $Lx = b$ is solved straightforwardly. The first equation is $\ell_{1,1}x_1 = b_1$, and it gives immediately the value of x_1. Then, we use the second equation to obtain x_2, and so on, up to the last component x_n. For an upper triangular system, we start with the computation of x_n, then x_{n-1} and so on, up to x_1.

For solving $Lx = b$ on a sequential computer, there are basically two algorithms. In the first one, the entries of the matrix L are accessed by rows and components of the solution x are computed in a natural way, one after the other from 1 to n,

```
for i = 1:n
 for j = 1:i-1
  b(i) = b(i) - L(i,j) * x(j);
 end
 x(i) = b(i) / L(i,i);
end
```

This is known as the *dot product algorithm* since the operation in the central loop on j is computing the dot product of the nonzero entries of the ith row (except the diagonal entry) with the first $i-1$ components of x. We observe that if we want to improve the accuracy, we can use one of the summation algorithms described in Section 1.7.

The other algorithm modifies the right-hand side as soon as each component of x has been computed and the matrix entries are accessed by columns,

```
for j = 1:n
 x(j) = b(j) / L(j,j);
 for i = j+1:n
  b(i) = b(i) - L(i,j) * x(j);
 end
end
```

This is known as the *axpy algorithm* by reference to the operation in the central loop which computes a vector plus (minus) a scalar x_j times a vector. It corresponds to switching the loops in the first implementation. Which one is faster depends on the computer architecture and the programming language.

Let us experiment with these two algorithms in Matlab. We construct a lower triangular matrix L of order 300 with ones on the diagonal and random numbers in $[-1, 1]$ in the strictly lower part. We solve linear systems with the principal submatrices of L and a random right-hand side. It turns out that the fastest algorithm is the axpy variant. Figure 2.1 shows ratios of computing times relative to the axpy runs as a function of the size of the matrix. We use the standard dot product variant and another one where the dot product is computed with the DDsum algorithm of Section 1.7. Figure 2.2 displays the relative error norms. The "exact" solution is computed in variable precision with 64 decimal digits.

The dot product algorithm is a little more expensive than the axpy algorithm, probably because of the way the matrix is accessed. Using the DDsum algorithm is much more expensive, but the error is smaller. However, for this example, the dot product and axpy algorithms give a satisfactory accuracy.

2.2 ▪ Gaussian elimination for general systems

The problem we are concerned with is obtaining the numerical solution of a linear system,

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

where A is a square nonsingular matrix (that is, $\det(A) \neq 0$) of order n with real entries and b is a given vector. Of course, the solution x of (2.1) is given by

$$x = A^{-1}b$$

where A^{-1} denotes the inverse of A. Unfortunately, in most cases A^{-1} is not explicitly known, except for some special problems and/or for small values of n. Moreover, it is generally not a good idea to compute the inverse to solve a linear system.

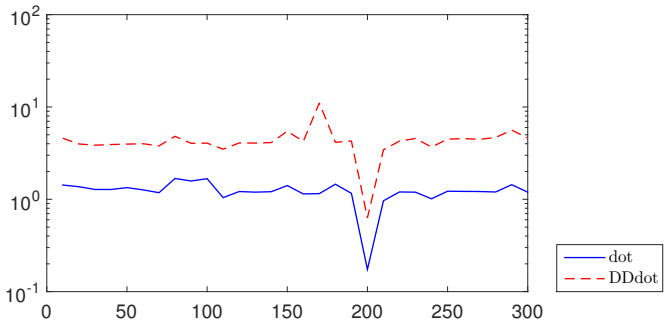


Figure 2.1. *Relative computing times for solving* $Lx = b$

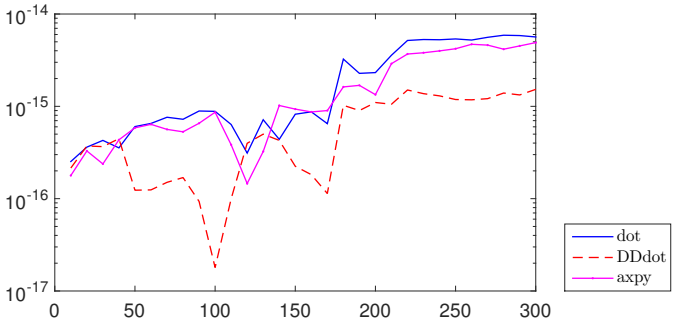


Figure 2.2. *Relative error norms when solving* $Lx = b$

Linear systems of equations had been solved by elimination methods a long, long time before matrices where introduced around the mid-1800s by A. Cayley, J.J. Sylvester and E. Laguerre, see [160]. Without going back to the Babylonians, linear systems were solved in almost the same way as today by Chinese mathematicians. The book in which we find these methods is the most important Chinese mathematical classic *Jiuzhang Suanshu*, which has been translated as *Nine Chapters on the Mathematical Art*. This text is believed to have been compiled some time between 100 BC and 100 AD, but it is likely that the content of Nine Chapters was much older than its date of compilation.

Algebra was introduced in medieval Europe through Latin translations of Arabic texts in the 12th century. After that, linear systems of order three or four were solved with integer coefficients by many people. We can cite, for instance, Nicolas Chuquet (c. 1445-1488), a French mathematician, who solved a system of order 3 in 1484, and Jean Borrel, also known as Johannes Buteo (1492-c. 1570), who solved a system of order 3 in 1559, which, in modern notation, is

$$\begin{aligned} 3x + y + z &= 42, \\ x + 4y + z &= 32, \\ x + y + 5z &= 40. \end{aligned}$$

Unlike Borrel, our readers can see that the corresponding matrix is strictly diagonally dominant and therefore nonsingular. All these early authors used some multiplications of the equations to eliminate unknowns. Their goal was to avoid divisions up to the end and also rational numbers as much as possible.

The use of elimination methods should have ended in the 18th century when the general solution of a nonsingular linear system of any order using ratios of determinants was given in 1750 by the Swiss mathematician Gabriel Cramer (1704-1754) in an appendix of his book [290] about algebraic curves published in 1750.

The solution is expressed by Cramer's formula which, in modern notation, is (see [494]),

$$x_i = \frac{1}{\det(A)} \begin{vmatrix} a_{1,1} & \cdots & a_{1,i-1} & b_1 & a_{1,i+1} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & a_{2,i-1} & b_2 & a_{2,i+1} & \cdots & a_{2,n} \\ \vdots & \ldots & \vdots & \vdots & \vdots & \ldots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & b_n & a_{n,i+1} & \cdots & a_{n,n} \end{vmatrix}, \quad i = 1, \ldots, n. \tag{2.2}$$

The computation of the solution x by (2.2) requires the evaluation of $n+1$ determinants of order n. This implies that this method will require more than $(n+1)!$ operations (multiplications and additions) to compute the solution. This is far too much even for small values of n. It gives already around $4\,10^7$ operations for $n = 10$. More efficient techniques to use determinants for solving a linear system were proposed by Felice Chiò (1813-1871), an Italian mathematician and physicist [251] in 1853, and Charles Lutwidge Dodgson (1832-1898) (who is better known as Lewis Caroll) [340] in 1866. Even with these modifications, using the determinant formulas to solve large linear systems is too expensive. Hence, elimination methods continued to be used, but Cramer's formulas show that there exists a perfectly parallel numerical method to solve a linear system, even though it may be too expensive.

The most widely used direct methods for general matrices belong to a class collectively known as Gaussian elimination. There are many variations of the same basic idea and we will describe some of them in the next sections.

2.2.1 ▪ Gaussian elimination without permutations

In this section we describe Gaussian elimination without permutations. The basis of the method is easily explained on a small example and then, it can be extended to any value of n. As the name of the technique implies, the main idea is to successively eliminate the unknowns, as it has been done for centuries. Consider the following small set of linear equations,

$$\begin{aligned} x_1 + x_2 &= 2 \\ 4x_1 + 5x_2 + 3x_3 &= 12 \\ 4x_1 + 6x_2 + 7x_3 &= 17 \end{aligned} \tag{2.3}$$

whose solution is $x_1 = x_2 = x_3 = 1$. Equations (2.3) can be written in matrix form as

$$Ax = \begin{pmatrix} 1 & 1 & 0 \\ 4 & 5 & 3 \\ 4 & 6 & 7 \end{pmatrix} x = \begin{pmatrix} 2 \\ 12 \\ 17 \end{pmatrix}. \tag{2.4}$$

The determinant of A is equal to 1. Therefore, the matrix is nonsingular and there is a unique solution to (2.3). Straightforwardly, the first equation is used to express x_1 as a function of x_2,

$$x_1 = 2 - x_2. \tag{2.5}$$

Then, using (2.5), x_1 is eliminated in the second and third equations giving a new (reduced) system of order 2, involving only x_2 and x_3,

$$\begin{aligned} x_2 + 3x_3 &= 4 \\ 2x_2 + 7x_3 &= 9. \end{aligned} \tag{2.6}$$

In the second step, x_2 is expressed as a function of x_3 using the first equation of (2.6),

$$x_2 = 4 - 3x_3. \tag{2.7}$$

Using (2.7) in the last equation of (2.6), we obtain a linear system involving only x_3 whose solution is immediately given, $x_3 = 1$. Knowing the value of x_3, we can compute x_2 with (2.7) which gives $x_2 = 1$. Finally, (2.5) gives $x_3 = 1$.

The previous elementary elimination method can be cast in a matrix framework. Eliminating x_1 from the second equation of (2.3) amounts to do a linear combination of the first two equations. This is obtained by left multiplying A with the matrix

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

Multiplying the system of (2.4) by $E_{2,1}$ replaces the second equation by the second equation minus four times the first one, leaving the two others invariant,

$$E_{2,1}A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 4 & 6 & 7 \end{pmatrix}.$$

Elimination of x_1 from the last equation of (2.3) is obtained by left multiplying with

$$E_{3,1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix},$$

and we obtain

$$E_{3,1}(E_{2,1}A) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 2 & 7 \end{pmatrix}.$$

Then, all the entries of the first column of A below the diagonal entry have been reduced to zero. Observe that the subsystem corresponding to the second and third rows and columns of this matrix is precisely the same as in (2.6).

Elimination of x_2 from the third equation is obtained by left multiplying with

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

and

$$E_{3,2}(E_{3,1}E_{2,1}A) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix}.$$

Therefore, when A is successively left multiplied by $E_{2,1}$, $E_{3,1}$ and $E_{3,2}$, it is reduced to an upper triangular form. Notice that $E_{3,1}$ and $E_{2,1}$ commute, since we have

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix}$$

We observe that the first column of the product is the "superposition" of the first columns of $E_{3,1}$ and $E_{2,1}$. Matrices $E_{2,1}$, $E_{3,1}$ and $E_{3,2}$ are nonsingular since their determinants are equal to 1. Therefore, the linear system of (2.4) can be transformed into the equivalent system,

$$E_{3,2}E_{3,1}E_{2,1}Ax = E_{3,2}E_{3,1}E_{2,1}b, \tag{2.8}$$

where

$$b = \begin{pmatrix} 2 \\ 12 \\ 17 \end{pmatrix}.$$

Let

$$L^{-1} = E_{3,2}E_{3,1}E_{2,1} = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 4 & -2 & 1 \end{pmatrix}.$$

With $E_{3,2}E_{3,1}E_{2,1}A$, equation (2.8) reduces to

$$L^{-1}Ax = Ux = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} x = L^{-1}b = \begin{pmatrix} 2 \\ 4 \\ 1 \end{pmatrix}. \tag{2.9}$$

Since the matrix U is upper triangular, this system, which is equivalent to (2.4), is easily solved starting with the last equation and moving backwards. This process is usually called *backward (or back) substitution*.

Besides solving the linear system, we have also seen that $L^{-1}A = U$. Therefore, we have the factorization,

$$A = LU.$$

L^{-1} being lower triangular, its inverse L is also lower triangular. The matrix A has been factored into the product of lower and upper triangular matrices. The matrix L is easily obtained from L^{-1}. First, we check that if

$$E_{3,1}E_{2,1} = \begin{pmatrix} 1 & 0 & 0 \\ \alpha & 1 & 0 \\ \beta & 0 & 1 \end{pmatrix},$$

then

$$E_{2,1}^{-1}E_{3,1}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & 0 & 1 \end{pmatrix}.$$

Moreover,

$$E_{3,2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \gamma & 1 \end{pmatrix} \Rightarrow E_{3,2}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\gamma & 1 \end{pmatrix},$$

and

$$L = E_{2,1}^{-1}E_{3,1}^{-1}E_{3,2}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\gamma & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -\alpha & 1 & 0 \\ -\beta & -\gamma & 1 \end{pmatrix}.$$

The matrix L is obtained straightforwardly from the elementary matrices $E_{i,j}$ by simply changing the signs of the nonzero off-diagonal entries. Of course, not all systems are as simple as this example, in which we were able to work with integers and without any division.

Let us now describe the method for a general linear system of order n. The first step of the algorithm is the elimination of the unknown x_1 in the equations 2 to n. This can be done through

$n-1$ steps. Assume that $a_{1,1} \neq 0$, $a_{1,1}$ is then called the first *pivot*. To eliminate x_1 from the second equation, we left multiply A by an *elementary matrix*,

$$E_{2,1} = \begin{pmatrix} 1 & & & & \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & & & \\ 0 & 0 & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ 0 & \dots & \dots & 0 & 1 \end{pmatrix}.$$

This corresponds to do a linear combination of the first two rows of A. More generally, to eliminate x_1 from the ith equation, we left multiply by

$$E_{i,1} = \begin{pmatrix} 1 & & & & & & & \\ 0 & 1 & & & & & & \\ \vdots & & \ddots & & & & & \\ 0 & & & \ddots & & & & \\ -\frac{a_{i,1}}{a_{1,1}} & 0 & \dots & 0 & 1 & & & \\ 0 & & & & & \ddots & & \\ \vdots & & & & & & \ddots & \\ 0 & \dots & & & & \dots & 0 & 1 \end{pmatrix},$$

the nonzero entries of the first column of $E_{i,1}$ being in positions $(1,1)$ and $(i,1)$. All these elementary matrices can be easily combined as it is shown in the following lemma.

Lemma 2.1. *Let $j > i$, then*

$$E_{i,1}E_{j,1} = \begin{pmatrix} 1 & & & & & & & & \\ 0 & 1 & & & & & & & \\ \vdots & & \ddots & & & & & & \\ 0 & & & \ddots & & & & & \\ -\frac{a_{i,1}}{a_{1,1}} & & & & 1 & & & & \\ 0 & & & & & \ddots & & & \\ \vdots & & & & & & \ddots & & \\ 0 & & & & & & & \ddots & \\ -\frac{a_{j,1}}{a_{1,1}} & & & & & & & 1 & \\ 0 & & & & & & & & \ddots \\ \vdots & & & & & & & & & \ddots \\ 0 & & & & & & & & & & 1 \end{pmatrix},$$

and

$$E_{j,1}E_{i,1} = E_{i,1}E_{j,1}.$$

Proof. The result is obtained by straightforward matrix multiplication. Note that we also have

$$E_{i,1}E_{j,1} = E_{i,1} + E_{j,1} - I.$$

□

Let us denote $L_1^{-1} = E_{n,1}E_{n-1,1}\cdots E_{2,1}$ and $A_2 = L_1^{-1}A$. Obviously, L_1^{-1} is lower triangular. The entries of A_2 are denoted by $a_{i,j}^{(2)}$. The matrix A_2 has the following nonzero structure,

$$A_2 = \begin{pmatrix} a_{1,1} & x & \dots & x \\ 0 & x & \dots & x \\ \vdots & \vdots & & \vdots \\ 0 & x & \dots & x \end{pmatrix},$$

the x's corresponding to (eventually) nonzero entries that are defined in the following lemma.

Lemma 2.2.

$$a_{i,j}^{(2)} = a_{i,j} - \frac{a_{i,1}a_{1,j}}{a_{1,1}}, \quad 2 \le i \le n, \quad 1 \le j \le n,$$
$$a_{1,j}^{(2)} = a_{1,j}, \quad 1 \le j \le n.$$

Proof. This is simply the result of the multiplication of A by L_1^{-1}. □

For the kth step of the algorithm, let A_k be the matrix that has been obtained by zeroing the entries below the diagonal in the $k-1$ first columns,

$$A_k = \begin{pmatrix} a_{1,1}^{(k)} & \dots & \dots & \dots & \dots & a_{1,n}^{(k)} \\ & \ddots & & & & \vdots \\ & & a_{k,k}^{(k)} & \dots & \dots & a_{k,n}^{(k)} \\ & & a_{k+1,k}^{(k)} & \dots & \dots & a_{k+1,n}^{(k)} \\ & & \vdots & \vdots & \vdots & \vdots \\ & & a_{n,k}^{(k)} & \dots & \dots & a_{n,n}^{(k)} \end{pmatrix}$$

and assume that $a_{k,k}^{(k)} \neq 0$. The entry $a_{k,k}^{(k)}$ is known as the kth pivot. For $i > k$, let

$$E_{i,k} = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & 0 & 1 & & & & \\ & & \vdots & & \ddots & & & \\ & & 0 & & & \ddots & & \\ & & -\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} & & & & 1 & \\ & & 0 & & & & & \ddots \\ & & \vdots & & & & & & \ddots \\ & & 0 & & & & & & & 1 \end{pmatrix},$$

where the entry $-\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}$ is in row i and column k. The off-diagonal entries that are not explicitly shown are zero. Let $L_k^{-1} = E_{n,k}E_{n-1,k}\cdots E_{k+1,k}$ and $A_{k+1} = L_k^{-1}A_k$. Then, as for the first column, we can characterize this product of matrices.

Lemma 2.3.

$$L_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & \\ & & \vdots & & \ddots \\ & & -\frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & 1 \end{pmatrix},$$

and A_{k+1} has the following structure

$$A_{k+1} = \begin{pmatrix} a_{1,1}^{(k+1)} & \cdots & \cdots & \cdots & \cdots & a_{1,n}^{(k+1)} \\ & \ddots & & & & \vdots \\ & & a_{k,k}^{(k+1)} & \cdots & \cdots & a_{k,n}^{(k+1)} \\ & & 0 & a_{k+1,k+1}^{(k+1)} & \cdots & a_{k+1,n}^{(k+1)} \\ & & \vdots & \vdots & & \vdots \\ & & 0 & a_{n,k+1}^{(k+1)} & \cdots & a_{n,n}^{(k+1)} \end{pmatrix}.$$

Proof. Straightforward. □

The entries of the jth column of A_{k+1} are given by the following expressions.

Lemma 2.4.

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)} a_{k,j}^{(k)}}{a_{k,k}^{(k)}}, \quad k+1 \le i \le n, \quad k \le j \le n,$$

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)}, \quad 1 \le i \le k, \quad 1 \le j \le n \quad \text{and } k+1 \le i \le n, \quad 1 \le j \le k-1.$$

Proof. This is the formula we obviously obtain when multiplying by L_k^{-1}. □

As we have seen in our small example above, it is useful to characterize the inverses of the matrices L_k^{-1}.

Lemma 2.5. *L_k^{-1} is nonsingular and*

$$L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & \\ & & \vdots & & \ddots \\ & & \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & 1 \end{pmatrix}$$

Proof. Let

$$\ell_k = \begin{pmatrix} 0 & \cdots & 0 & \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & \cdots & \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} \end{pmatrix}^T.$$

Obviously, $L_k^{-1} = I - \ell_k {e_k}^T$ where e_k is the kth column of the identity matrix. Then,

$$L_k = I + \ell_k {e_k}^T,$$

because,

$$\begin{aligned} L_k L_k^{-1} &= (I + \ell_k {e_k}^T)(I - \ell_k {e_k}^T), \\ &= I - \ell_k {e_k}^T \ell_k {e_k}^T, \end{aligned}$$

and ${e_k}^T \ell_k = 0$. □

The preceding results can be summarized in the following proposition.

Proposition 2.6. *If for all k, $1 \leq k \leq n-1$, $a_{k,k}^{(k)} \neq 0$, there exists a factorization*

$$A = LU,$$

where L is lower triangular with a unit diagonal and U is upper triangular.

Proof. The elimination process goes as follows,

$$\begin{aligned} A_1 &= A, \\ A_2 &= L_1^{-1} A_1, \\ &\vdots \\ A_n &= L_{n-1}^{-1} A_{n-1}. \end{aligned}$$

The last matrix A_n is upper triangular and is therefore denoted by U. Hence,

$$L_{n-1}^{-1} L_{n-2}^{-1} \cdots L_1^{-1} A = U.$$

The matrices $L_i^{-1}, 1 \leq i \leq n-1$ are nonsingular. Then,

$$A = (L_1 \cdots L_{n-1}) U.$$

The product of unit lower triangular matrices being unit lower triangular, we have

$$L = L_1 \cdots L_{n-1}$$

which is a lower triangular matrix with a unit diagonal. Moreover, let us show that

$$L = \begin{pmatrix} 1 & & & & & \\ \frac{a_{2,1}^{(1)}}{a_{1,1}^{(1)}} & \ddots & & & & \\ \vdots & \ddots & 1 & & & \\ \vdots & & \frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & & \\ \vdots & & \vdots & \ddots & \ddots & \\ \frac{a_{n,1}^{(1)}}{a_{1,1}^{(1)}} & \cdots & \frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & \cdots & \frac{a_{n,n-1}^{(n-1)}}{a_{n-1,n-1}^{(n-1)}} & 1 \end{pmatrix}.$$

Since $L_i L_{i+1} = I + \ell_i {e_i}^T + \ell_{i+1} {e_{i+1}}^T$, we obtain

$$L = L_1 \cdots L_{n-1} = I + \ell_1 {e_1}^T + \cdots \ell_{n-1} {e_{n-1}}^T.$$

□

Theorem 2.7. *If the factorization $A = LU$ exists, it is unique.*

Proof. The proof is by contradiction. Assume there exist two such factorizations, $A = L_1U_1 = L_2U_2$, where L_1 and L_2 (resp. U_1 and U_2) are two lower triangular matrices with a unit diagonal (resp. upper triangular matrices). Then,

$$L_2^{-1}L_1 = U_2U_1^{-1}.$$

The matrix on the left-hand side is lower triangular with a unit diagonal and the matrix on the right-hand side is upper triangular. Therefore, they are both diagonal and

$$L_2^{-1}L_1 = U_2U_1^{-1} = I$$

which shows that the decomposition is unique. □

Let us derive the conditions under which there exists an LU factorization. We have to identify conditions for all the pivots to be nonzero.

Theorem 2.8. *A nonsingular matrix A has a unique LU factorization if and only if all the principal minors of A are nonzero. That is,*

$$A\begin{pmatrix} 1 & 2 & \dots & k \\ 1 & 2 & \dots & k \end{pmatrix} \neq 0, \quad k = 1, \dots, n$$

where the determinant is defined as

$$A\begin{pmatrix} i_1 & i_2 & \dots & i_p \\ k_1 & k_2 & \dots & k_p \end{pmatrix} = \begin{vmatrix} a_{i_1,k_1} & a_{i_1,k_2} & \dots & a_{i_1,k_p} \\ a_{i_2,k_1} & a_{i_2,k_2} & \dots & a_{i_2,k_p} \\ \vdots & \vdots & & \vdots \\ a_{i_p,k_1} & a_{i_p,k_2} & \dots & a_{i_p,k_p} \end{vmatrix}.$$

Moreover,

$$a_{k,k}^{(k)} = \frac{A\begin{pmatrix} 1 & 2 & \dots & k \\ 1 & 2 & \dots & k \end{pmatrix}}{A\begin{pmatrix} 1 & 2 & \dots & k-1 \\ 1 & 2 & \dots & k-1 \end{pmatrix}},$$

and

$$a_{k,j}^{(k)} = \frac{A\begin{pmatrix} 1 & 2 & \dots & k-1 & k \\ 1 & 2 & \dots & k-1 & j \end{pmatrix}}{A\begin{pmatrix} 1 & 2 & \dots & k-1 \\ 1 & 2 & \dots & k-1 \end{pmatrix}}, \quad j > k.$$

Proof. Assume there exists an LU factorization. From the proof of Proposition 2.6,

$$A_{k+1} = L_k^{-1} \dots L_1^{-1} A.$$

Therefore,

$$A = L_1 \dots L_k A_{k+1},$$

and we also have $A = LU$. In block form, this is written as

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \quad L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}, \quad U = \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{pmatrix},$$

and from Proposition 2.6,

$$L_1 \dots L_k = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I \end{pmatrix}, \quad A_{k+1} = \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & W_{2,2} \end{pmatrix},$$

where all the matrices in block position $(1,1)$ are square of order k. By equating blocks, we have

$$\begin{aligned} A_{1,1} &= L_{1,1}U_{1,1} \\ A_{2,2} &= L_{2,1}U_{1,2} + L_{2,2}U_{2,2} \\ A_{2,2} &= L_{2,1}U_{1,2} + W_{2,2} \end{aligned}$$

Therefore, $L_{1,1}U_{1,1}$ is the LU factorization of the leading principal submatrix of order k of A and $L_{2,2}U_{2,2}$ is the factorization of the matrix $W_{2,2}$ in the bottom right-hand corner of A_{k+1} since we have $W_{2,2} = L_{2,2}U_{2,2}$.

Note that $\det(A) = \det(A_{k+1})$. We have $\det(L_{1,1}) = 1$ and $\det(A_{1,1}) = \det(U_{1,1})$. Since the matrix $U_{1,1}$ is upper triangular, its determinant is equal to the product of the diagonal entries. Therefore, for all k,

$$\det(A_{1,1}) = a_{1,1}^{(1)} \cdots a_{k,k}^{(k)}.$$

This shows that the principal minors are nonzero and the first formula. The converse of the proof is easily derived by induction.

Now, we proceed in the same way as J.H. Wilkinson in [1120]. We have

$$\begin{pmatrix} A_{1,1} \\ A_{2,2} \end{pmatrix} = \begin{pmatrix} L_{1,1} \\ L_{2,1} \end{pmatrix} U_{1,1}.$$

Let $A_{1,1}^i$ denotes the matrix constructed with the $k-1$ first rows and the i-th row of the first k columns of A with $i \geq k$ and let $L_{1,1}^i$ be defined in a similar way. Then, $A_{1,1}^i = L_{1,1}^i U_{1,1}$. The matrix $L_{1,1}^i$ is triangular and

$$\det(L_{1,1}^i) = \ell_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}.$$

Therefore, since $\det(A_{1,1}^i) = \ell_{i,k} \det(U_{1,1})$ and $\det(U_{1,1}) = \det(A_{1,1})$,

$$\ell_{i,k} = \frac{\det(A_{1,1}^i)}{\det(A_{1,1})} = \frac{A \begin{pmatrix} 1 & 2 & \cdots & k-1 & i \\ 1 & 2 & \cdots & k-1 & k \end{pmatrix}}{A \begin{pmatrix} 1 & 2 & \cdots & k \\ 1 & 2 & \cdots & k \end{pmatrix}}.$$

Similarly, we have

$$\begin{pmatrix} A_{1,1} & A_{1,2} \end{pmatrix} = L_{1,1} \begin{pmatrix} U_{1,1} & U_{1,2} \end{pmatrix}.$$

This leads to

$$u_{k,i} = \frac{A \begin{pmatrix} 1 & 2 & \cdots & k-1 & k \\ 1 & 2 & \cdots & k-1 & i \end{pmatrix}}{A \begin{pmatrix} 1 & 2 & \cdots & k-1 \\ 1 & 2 & \cdots & k-1 \end{pmatrix}},$$

for the entries of U. □

On modern computers with several level of memories, the LU factorization is often implemented in block form. Let

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & I \end{pmatrix} \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & S \end{pmatrix}, \tag{2.10}$$

where $A_{1,1}$ and $L_{1,1}$ are of order $p > 1$, and S is the Schur complement $S = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}$. First $A_{1,1}$ is factored as $A_{1,1} = L_{1,1}U_{1,1}$. Then, $U_{1,2}$ is computed by solving the triangular system with several right-hand sides $L_{1,1}U_{1,2} = A_{1,2}$. The matrix $L_{2,1}$ is computed by solving $L_{2,1}U_{1,1} = A_{2,1}$ which, when transposed, is also a triangular system. Then, $S = A_{2,2} - L_{2,1}U_{1,2}$ and the process is repeated on S. The optimal value of p depends on the computer architecture.

The same process repeated on $A_{1,1}$ and S, with a smaller value of p, is called a *recursive LU factorization*.

2.2.2 ▪ Gaussian elimination with permutations (partial pivoting)

In this section, we allow for the possibility of having zero pivots and we show that, nevertheless, a factorization can be computed by exchanging (or *swapping*) rows. This is known as *pivoting*. If the first pivot $a_{1,1}$ is zero, we permute the first row with a row p such that $a_{p,1} \neq 0$. Finding such a p is always possible, otherwise we must have $\det(A) = 0$. The row interchange is done by left multiplication of A by a *permutation matrix* P_1. P_1 is equal to the identity matrix except that rows 1 and p have been switched, that is,

$$P_1 = \begin{pmatrix} 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & \dots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & & \\ 1 & 0 & \dots & 0 & 0 & 0 & \dots & \\ 0 & \dots & & \dots & 0 & 1 & \ddots & \vdots \\ \vdots & & & & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & & \dots & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Note that $P_1^{-1} = P_1 = P_1^T$. For the permuted matrix, the algorithm is the same as without permutations. We construct L_1 such that $A_2 = L_1 P_1 A$. Let us describe the kth step. The main difference from what we have done before is that the possibility exists that the pivot is zero. If this is the case, it is possible to find a row p such that $a_{p,k}^{(k)} \neq 0$. The reason for that being that $\det(A_k) = \det(A) \neq 0$ and the determinant $\det(A_k)$ is equal to the product of the first $k-1$ (nonzero) pivots and the determinant of the matrix in the right-hand bottom corner. Therefore, this matrix is nonsingular. In fact, we choose a nonzero element $a_{p,k}^{(k)}, p > k$ which has the maximum modulus. This strategy of choosing the pivot in the kth column is called *partial pivoting*. Then, we multiply A_k by the corresponding permutation matrix P_k and we apply the elimination algorithm,

$$A_{k+1} = L_k^{-1} P_k A_k.$$

Finally, we obtain

$$U = L_{n-1}^{-1} P_{n-1} \cdots L_2^{-1} P_2 L_1^{-1} P_1 A.$$

It may seem that we have lost the good properties of the Gaussian algorithm since permutation matrices have appeared in between the lower triangular matrices, even if some of them are

equal to the identity matrix when there is no need for pivoting for the corresponding columns. Fortunately, we have the following result.

Lemma 2.9. *Let P_p be a permutation matrix representing the permutation between indices p and $q > p$. Then, $\forall k < p$*

$$L_k P_p = P_p L'_k.$$

where L'_k is deduced from L_k by the permutation of entries in rows p and q in column k.

Proof. Recall that $P_p^{-1} = P_p$ and $L_k = I + \ell_k {e_k}^T$,

$$L'_k = P_p L_k P_p = P_p(I + \ell_k {e_k}^T)P_p = I + P_p \ell_k {e_k}^T P_p.$$

Since $p > k$, $P_p e_k = e_k$. Therefore

$$L'_k = I + {\ell_k}' {e_k}^T$$

where ${\ell_k}' = P_p \ell_k$. The same kind of result holds for L_k^{-1} since $L_k^{-1} = I - \ell_k {e_k}^T$. □

Now, we can characterize the LU decomposition of a given matrix.

Theorem 2.10. *Let A be a nonsingular matrix, there exists a permutation matrix P such that*

$$PA = LU,$$

where L is lower triangular with a unit diagonal and U is upper triangular.

Proof. We have seen that

$$A = P_1 L_1 P_2 \cdots P_{n-1} L_{n-1} U.$$

From Lemma 2.9,

$$A = P_1 P_2 \cdots P_{n-1}(L''_1) \cdots (L''_{n-1})U,$$

where $L''_k = P_{n-1} \cdots P_{k+1} L_k P_{k+1} \cdots P_{n-1}$, corresponding to a permutation of the coefficients of column k. □

We observe that, by definition, in the factorization of Theorem 2.10, we have $|\ell_{i,j}| \leq 1$ since we have chosen the pivot as the element of maximum modulus.

Usually, the permutation matrix P is stored as a vector of indices since row permutations are not explicitly performed during the factorization, even though this is sometimes done to avoid indirect addressing. The linear system $Ax = b$ is transformed into

$$PAx = LUx = Pb,$$

and is solved in two steps by

$$Ly = Pb, \quad Ux = y.$$

These two triangular solves are known respectively as the *forward and backward sweeps*. For general systems and in finite precision arithmetic, pivoting is used even when $a_{k,k}^{(k)} \neq 0$ to improve stability. A permutation is systematically done with the row p giving $\max_{p>k} |a_{p,k}^{(k)}|$.

If we use the block factorization (2.10), the choices of the pivots only occur in $A_{1,1}$. A variant [1066] consists in factoring the $n \times p$ leftmost matrix (which is called a *panel*) using partial pivoting,

$$P_1 \begin{pmatrix} A_{1,1} \\ A_{1,2} \end{pmatrix} = \begin{pmatrix} L_{1,1} \\ L_{1,2} \end{pmatrix} U_{1,1}.$$

Then, permute the rows of the rightmost columns,

$$\begin{pmatrix} A'_{1,2} & A'_{2,2} \end{pmatrix} = P_1 \begin{pmatrix} A_{1,2} & A_{2,2} \end{pmatrix},$$

solve the triangular system $L_{1,1}U_{1,2} = A'_{1,2}$ for $U_{1,2}$ and compute $A''_{2,2} = A'_{2,2} - L_{2,1}U_{1,2}$. The next step is to recursively compute the factorization $P_2 A''_{2,2} = L_{2,2}U_{2,2}$.

When this is done, the new permutation is applied to the previous lower block $L'_{1,2} = P_2 L_{2,1}$. It yields the factorization

$$P \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} L_{1,1} & 0 \\ L'_{2,1} & L_{2,2} \end{pmatrix} \begin{pmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{pmatrix},$$

where the permutation P is constructed from P_1 and P_2. The width p of the panel can be different at each step of the recursion.

In fact, the entire trailing submatrix is not updated after a block of columns is factored. After the factorization of the first column ($p = 1$), we can update just the next column to the right, which enables the algorithm to proceed. When the second column is factored, we need to apply the updates from the first two columns before the algorithm can proceed. We update two more columns and proceed. When four columns are factored, they are used to update four more, and so on. This technique is called *look-ahead*.

2.2.3 ▪ Gaussian elimination with other pivoting strategies

Different pivoting strategies may also be used. For instance, we can search for the pivot not only in the lower part of the kth column but in all the remaining columns in the bottom-right submatrix. The chosen pivot is an element that realizes $\max_{i,j} |a^{(k)}_{i,j}|, \forall i, j \geq k$. This technique is known as *complete pivoting*. Then, we not only have to do row permutations but also column permutations to bring the chosen pivot into position (k, k). This is achieved by multiplying to the right by a permutation matrix. In the end, we obtain two permutation matrices P and Q such that

$$PAQ = LU.$$

The solution of the linear system is obtained through three steps,

$$Ly = Pb, \quad Uz = y, \quad x = Q^T z.$$

We will see that complete pivoting has some advantages regarding stability. However, the cost of finding the pivot is much larger than for partial pivoting.

Another strategy called *rook's pivoting* (or rook pivoting) has been introduced by G. Poole and L. Neal [922, 923]. At the k-th step, the algorithm is the following. Let

$$r_1 = \min\{r| \, |a^{(k)}_{r,k}| \geq |a^{(k)}_{i,k}|, \; k \leq i \leq n\}$$

and

$$c_1 = \min\{c| \, |a^{(k)}_{r_1,c}| \geq |a^{(k)}_{r_1,j}|, \; k \leq j \leq n\}.$$

If $c_1 = k$, then $a^{(k)}_{r_1,k}$ is the selected pivot. If $c_1 \neq k$, column c_1 is searched for the entry with maximum modulus. Let

$$r_2 = \min\{r| \, |a^{(k)}_{r,c_1}| \geq |a^{(k)}_{i,c_1}|, \; k \leq i \leq n\}$$

and

$$c_2 = \min\{c| \, |a^{(k)}_{r_2,c}| \geq |a^{(k)}_{r_2,j}|, \; k \leq j \leq n\}$$

Then, we proceed as with r_1 and c_1, and so on, until we have selected a pivot.

Therefore, rook's pivoting searches for coefficients of maximum modulus in rows, then columns and then, rows and columns until an entry $a_{r,c}^{(k)}$ satisfies $|a_{r,c}^{(k)}| \geq |a_{i,c}^{(k)}|$, $k \leq i \leq n$ and $|a_{r,c}^{(k)}| \geq |a_{r,j}^{(k)}|$, $k \leq j \leq n$. In other words this method selects a pivot which is the largest in magnitude in both its row and its column. It is an intermediate between partial and complete pivoting. Numerical experiments using this strategy are given in [923]. Rook's pivoting was also studied by X.-W. Chang [230].

Other pivoting strategies have been devised over the years; see, for instance, M. Olschowka and A. Neumaier [879], J.M. Peña [912], and V. Cortés and J.M. Peña [284]. More recently, different strategies have been defined for LU factorization on parallel computers, as *tournament pivoting* [575] that we will describe in Section 2.12, or threshold pivoting [749].

Another approach, that could be useful for parallel computations, is to try to completely avoid pivoting by multiplying the matrix by random recursive butterfly transformations (RBT) and then apply LU without pivoting, see [904, 902, 79, 939, 750]. Let n be even, a butterfly matrix is of the form

$$B^{(n)} = \frac{1}{\sqrt{2}} \begin{pmatrix} R_0 & R_1 \\ R_0 & -R_1 \end{pmatrix},$$

where R_0 and R_1 are nonsingular (random) diagonal matrices of order $n/2$. When n is a multiple of 2^{d-1}, an RBT of depth d is

$$U^{(n)} = \begin{pmatrix} B_1^{(n/2^{d-1})} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & B_{2^{d-1}}^{(n/2^{d-1})} \end{pmatrix} \cdots \begin{pmatrix} B_1^{(n/2)} & 0 \\ 0 & B_2^{(n/2)} \end{pmatrix} B^{(n)}.$$

These matrices have this name because their structure is similar to matrices in the discrete Fourier transform that bear the butterfly name due to the data transfer patterns they form. The matrix A is transformed to $A' = U^T A V$ where U and V are RBTs. D.S. Parker [904] proved that such a transformation for $d = \log_2 n$ makes the matrix factorizable without pivoting with probability 1. For an implementation, see [750].

Let us do some numerical experiments with pivoting strategies. We use the `gep` Matlab function from N.J. Higham's *Matrix Computation Toolbox*[2]. A is a random matrix of order 500 whose condition number is $6.5638\ 10^2$, with a norm equal to 44.366. The right-hand side b is also random. Each computation was repeated 10 times. The results are shown in Table 2.1. The computing times are normalized with the time of the algorithm without pivoting which is the fastest. We define the growth factor as

$$g_A = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\|A\|_\infty}.$$

We will see later that the smallest is g_A, the better we are. The growth factor is much larger when there is no pivoting. The smallest growth factor is given by complete pivoting, but partial pivoting and rook's pivoting already give a large reduction of g_A. Without pivoting, the equality $A = LU$ is not so well satisfied.

Let us now vary the order n from 10 to 300. We solve linear systems with the principal matrices of A and the corresponding parts of b. Figure 2.3 shows the growth factors as functions of n. Without pivoting, the growth factor is increasing fast and is much larger than for the other strategies. The best result is given by complete pivoting and rook's pivoting is in between partial and complete pivoting. Figure 2.4 displays the relative error norms. Not pivoting gives the largest error norms.

[2] https://nhigham.com/ or http://www.ma.man.ac.uk/∼higham/mctoolbox

Table 2.1. *Pivoting strategies, A random, $n = 500$*

	no piv.	partial piv.	complete piv.	rook's piv.
time	1	1.0362	1.4553	1.0399
g_A	$2.8315\ 10^4$	11.098	5.0790	7.5319
$\frac{\|PAQ-LU\|}{\|A\|}$	$1.9558\ 10^{-12}$	$4.4340\ 10^{-15}$	$2.6207\ 10^{-15}$	$3.1801\ 10^{-15}$

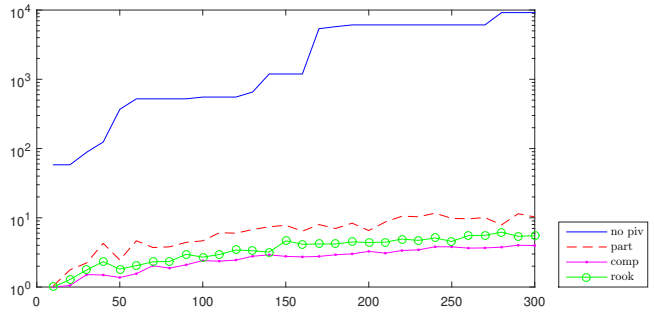


Figure 2.3. *Growth factors, A random*

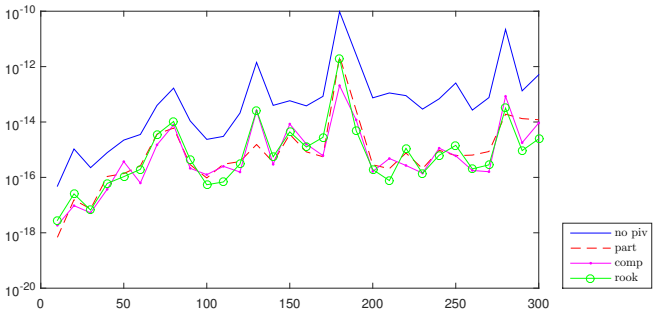


Figure 2.4. *Relative error norms, A random*

2.2.4 ▪ Operation counts

Despite what we said in Chapter 1, it is interesting to compute the number of floating-point operations that must be done to obtain the LU factorization of A, a dense matrix of order n.

For computing the kth column of L, we need one division by the pivot and $n-k$ multiplications. To compute the updated matrix A_{k+1}, we need (after having computed the multipliers $-a_{i,k}^{(k)}/a_{k,k}^{(k)}$ which are the entries of L) $(n-k)^2$ additions and the same number of multiplications. To obtain the total number of operations, we sum these numbers from 1 to $n-1$

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

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

$$= \frac{1}{3}n(n-1)(n-\frac{1}{2}).$$

Theorem 2.11. *To obtain the factorization $PA = LU$ of Theorem 2.10, we need $\frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6}$ floating-point operations (multiplications and additions) and $n-1$ divisions. The solutions of the triangular systems to obtain the solution x give $n(n-1)$ floating-point operations for L and $n(n-1)+n$ for U. Note that this is an order of magnitude less than what is needed for the factorization.*

□

2.3 ▪ Gaussian elimination for symmetric systems

The factorization of symmetric matrices is an important special case that we consider in more detail in this section. Let us specialize the previous algorithms to the symmetric case. We consider a factorization slightly different from what we have seen above,

$$A = LDL^T,$$

where L is lower triangular with a unit diagonal and D is diagonal. There are several possible ways to compute this factorization. We will study three different algorithms that will lead to six ways of coding the factorization. Note that these factorizations may fail if the matrix A is not positive definite.

2.3.1 ▪ The outer product algorithm

The first method to construct the LDL^T factorization is similar to what we used for nonsymmetric systems. It proceeds column by column. Assume $a_{1,1} \neq 0$ and

$$L_1 = \begin{pmatrix} 1 & 0 \\ \ell_1 & I \end{pmatrix}, \quad D_1 = \begin{pmatrix} a_{1,1} & 0 \\ 0 & A_2 \end{pmatrix},$$

$$A = \begin{pmatrix} a_{1,1} & a_1^T \\ a_1 & B_1 \end{pmatrix} = L_1 D_1 L_1^T.$$

By identification, we obtain expressions for ℓ_1 and A_2,

$$\ell_1 = \frac{a_1}{a_{1,1}},$$
$$A_2 = B_1 - \frac{1}{a_{1,1}} a_1 a_1^T = B_1 - a_{1,1}\ell_1\ell_1^T.$$

The matrix A_2 is obviously symmetric. If we assume that the $(1,1)$ element of A_2 is nonzero, we can use the same technique and write

$$A_2 = \begin{pmatrix} a_{2,2}^{(2)} & a_2^T \\ a_2 & B_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \ell_2 & I \end{pmatrix} \begin{pmatrix} a_{2,2}^{(2)} & 0 \\ 0 & A_3 \end{pmatrix} \begin{pmatrix} 1 & \ell_2^T \\ 0 & I \end{pmatrix}.$$

Similarly,

$$\ell_2 = \frac{a_2}{a_{2,2}^{(2)}},$$
$$A_3 = B_2 - \frac{1}{a_{2,2}^{(2)}} a_2 a_2^T = B_2 - a_{2,2}^{(2)}\ell_2\ell_2^T.$$

We observe that, if we denote,

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \ell_2 & I \end{pmatrix},$$

then,

$$D_1 = \begin{pmatrix} a_{1,1} & 0 \\ 0 & A_2 \end{pmatrix} = L_2 \begin{pmatrix} a_{1,1} & 0 & 0 \\ 0 & a_{2,2}^{(2)} & 0 \\ 0 & 0 & A_3 \end{pmatrix} L_2^T = L_2 D_2 L_2^T.$$

Therefore, after two steps, we have $A = L_1 L_2 D_2 L_2^T L_1^T$. We note that

$$L_1 L_2 = \begin{pmatrix} 1 & 0 \\ \ell_1 & \begin{pmatrix} 1 & 0 \\ \ell_2 & I \end{pmatrix} \end{pmatrix}.$$

The product of L_1 and L_2 is a lower triangular matrix. If all the pivots are nonzero, we can continue and at the last step, we obtain

$$A = A_1 = L_1 L_2 \cdots L_{n-1} D L_{n-1}^T \cdots L_1^T = L D L^T,$$

where L is unit lower triangular and D is diagonal. There exists a variant of this algorithm where a decomposition

$$A = \bar{L} \bar{D}^{-1} \bar{L}^T$$

is obtained with $\bar{L}$ lower triangular, $\bar{D}$ diagonal and $\mathrm{diag}(\bar{L}) = \mathrm{diag}(\bar{D})$. We obtain this variant from the previous algorithm by writing

$$A = LDL^T = (LD)D^{-1}(DL^T),$$

and $\bar{D} = D$, $\bar{L} = LD$.

The matrix L has been constructed column by column. This method is called the *outer product* algorithm since an outer product aa^T is involved at each step.

2.3.2 ▪ The bordering algorithm

The matrix A can be partitioned in a different way as

$$A = \begin{pmatrix} C_{n-1} & a_n \\ a_n^T & a_{n,n} \end{pmatrix},$$

Assume that C_{n-1} has already been factored as

$$C_{n-1} = L_{n-1} D_{n-1} L_{n-1}^T,$$

L_{n-1} being unit lower triangular and D_{n-1} diagonal. We can write,

$$A = \begin{pmatrix} L_{n-1} & 0 \\ \ell_n^T & 1 \end{pmatrix} \begin{pmatrix} D_{n-1} & 0 \\ 0 & d_{n,n} \end{pmatrix} \begin{pmatrix} L_{n-1}^T & \ell_n \\ 0 & 1 \end{pmatrix}.$$

Then, by identification,

$$\ell_n = D_{n-1}^{-1} L_{n-1}^{-1} a_n,$$
$$d_{n,n} = a_{n,n} - \ell_n^T D_{n-1} \ell_n.$$

By induction, we can start with the decomposition of the 1×1 matrix $a_{1,1}$, adding one row and column at a time, and obtaining at each step the factorization of an enlarged matrix. The only operation we have to perform at each step is solving a triangular system. To be able to proceed to the next step, we need the diagonal entries of D_n to be nonzero. For obvious reasons, this method is called the *bordering algorithm*.

2.3.3 ▪ The dot product algorithm

A third way to compute the factorization is simply to write the formulas for the matrix product,

$$A = LDL^T.$$

Assume $i \geq j$, we have

$$a_{i,j} = \sum_{k=1}^{j} \ell_{i,k}\ell_{j,k}d_{k,k}.$$

If we set $i = j$ in this formula then, since $\ell_{i,i} = 1$, we obtain

$$d_{j,j} = a_{j,j} - \sum_{k=1}^{j-1} (\ell_{j,k})^2 d_{k,k},$$

and for $i > j$,

$$\ell_{i,j} = \frac{1}{d_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k}\ell_{j,k}d_{k,k} \right).$$

Since we have to consider the product of the transpose of a vector times a vector, this method is called the *dot product algorithm* or sometimes the inner or scalar product algorithm. We observe that, if the diagonal entries of D are positive, we can write $A = L\sqrt{D}\sqrt{D}L^T$. This is the algorithm that was devised by A.-L. Cholesky around 1910, see [160].

The number of floating-point operations required for these three variants is about one half of the number of operations for the general algorithm, that is, approximately $\frac{n^3}{6}$ multiplications and the same number of additions.

2.3.4 ▪ Coding the three factorization algorithms

Let us now consider the different ways of coding the three algorithms we have just described for general (dense) symmetric matrices. The codes are written in Matlab-like language, although for clarity, we do not always use the most compact and efficient Matlab constructs.

In the outer product algorithm the matrix L is constructed column by column. At step k, column k is constructed by multiplying by the inverse of the pivot and then, the columns at the right of column k are modified using the values of the entries of column k. This is called a *right-looking algorithm* and it is summarized in Figure 2.5

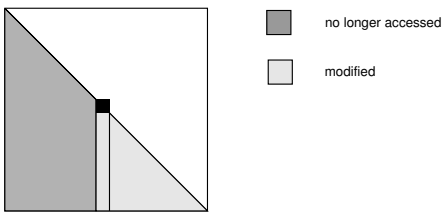


Figure 2.5. *The outer product algorithm data layout*

The modification of columns $k + 1$ to n can be done by rows or by columns and this leads to the two codes given below. We store the matrix D in a vector denoted by d and L in a separate matrix although in practice it can be stored in the lower triangular part of A (if A is not to be saved). The array `temp` is a temporary vector whose use can eventually be avoided. We use it for

clarity of presentation. Note that the coding is slightly different from that given in J.J. Dongarra, F.G. Gustavson, and A. Karp [352]. It is done such that in the main loop, `i` is a row index, `j` is a column index and `k` can eventually be both. The strictly lower triangular part of L is initialized to that of A using the function `tril`.

Outer product kij algorithm

```
function [L,d] = kij(A)
[m,n] = size(A);
d = zeros(n,1);
temp = zeros(n,1);
L = tril(A);
d(1) = A(1,1);
L(1,1) = 1;
for k=1:n-1
 dki = 1 / d(k);
 temp(k+1:n) = dki * L(k+1:n,k);
 for i=k+1:n
  for j=k+1:i
   L(i,j) = L(i,j) - temp(i) * L(j,k);
  end % for j
 end % for i
 L(k+1:n,k) = temp(k+1:n);
 d(k+1) = L(k+1,k+1);
 L(k+1,k+1) = 1;
end % for k
```

To reflect the way the three loops are nested, this algorithm is called the kij form. We can eliminate the temporary vector `temp` by using the upper part of the matrix `L`. However, we think the coding is clearer using `temp`. Modifying by rows (interchanging the loops on `i` and `j`) we obtain

Outer product kji algorithm

```
function [L,d] = kji(A)
[m,n] = size(a);
d=zeros(n,1);
temp=zeros(n,1);
L = tril(A);
d(1) = A(1,1);
L(1,1) = 1;
for k=1:n-1
 dki = 1 / d(k);
 temp(k+1:n) = dki * L(k+1:n,k);
 for j=k+1:n
  for i=j:n
   L(i,j) = L(i,j) - temp(i) * L(j,k);
  end % for i
```

```
 end % for i
 L(k+1:n,k) = temp(k+1:n);
 d(k+1) = L(k+1,k+1);
 L(k+1,k+1) = 1;
end % for k
```

Now we consider the bordering algorithm whose data accesses are summarized in Figure 2.6. It is a *left-looking algorithm.* For each row i, we have to solve a triangular system. As we have seen above, there are two algorithms to do this. One is column oriented, the other is row oriented.

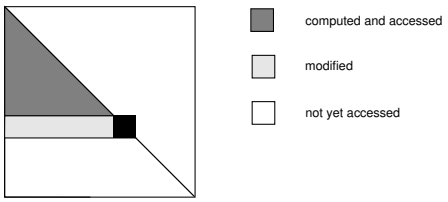


Figure 2.6. *The bordering algorithm data layout*

Bordering ijk algorithm

```
function [L,d] = ijk(A)
[m,n] = size(A);
d = zeros(n,1);
temp = zeros(1,n);
L = tril(A);
d(1) = A(1,1);
L(1,1)=1;
for i=2:n
 temp(1:i) = A(i,1:i);
 for j=1:i
  if j ~= i
   L(i,j) = temp(j) / d(j);
  end % if
  for k=j+1:i
   temp(k) = temp(k) - L(k,j) * temp(j);
  end % for k
 end % for j
 d(i) = temp(i);
 L(i,i)=1;
end % for i
```

There are too many divisions in the previous coding since they are in a `j` loop. This can be avoided by storing the inverses of d as they are computed.

Bordering ikj algorithm

```
function [L,d] = ikj(A)
[m,n] = size(A);
d = zeros(n,1);
temp = zeros(1,n);
L = tril(A);
d(1) = A(1,1);
L(1,1)=1;
for i=2:n
 temp(1:i) = A(i,1:i);
 for k=1:i
  for j=1:k-1
   temp(k) = temp(k) - temp(j) * L(k,j);
  end % for j
  if k ~= i
   L(i,k) = temp(k) / d(k);
  else
   d(i) = temp(i);
   L(i,i) = 1;
  end % if
 end % for k
end % for i
```

Finally, we consider the dot product algorithm. This algorithm is schematically depicted in Figure 2.7.

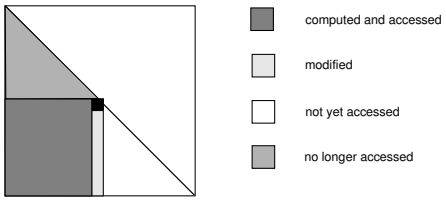


Figure 2.7. *The dot product algorithm data layout*

Dot product jik algorithm

```
function [L,d] = jik(A)
[m,n] = size(A);
L = tril(A);
for j=1:n
 for k=1:j-1
  L(j,k) = L(j,k) / d(k);
 end % for k
 d(j) = A(j,j);
 for k=1:j-1
  d(j) = d(j) - L(j,k)^2 * d(k);
 end % for k
 for i=j+1:n
```

```
  for k=1:j-1
   L(i,j) = L(i,j) - L(i,k) * L(j,k);
  end % for k
 end % for i
 L(j,j)=1;
end % for j
```

In the computation of $a_{i,j} - \sum_{k=1}^{j-1} \ell_{i,k}\ell_{j,k}d_{k,k}$, one can compute $a_{i,j} - \ell_{i,k}\ell_{j,k}d_{k,k}$ for a fixed value of k looping on i provided the division by $d_{j,j}$ is done afterwards. Then, we obtain the following algorithm.

Dot product jki algorithm

```
function [L,d] = jki(A)
[m,n] = size(A);
L = tril(A);
for j=1:n
 for k=1:j-1
  L(j,k) = L(j,k) / d(k);
 end % for k
 d(j) = A(j,j);
 for k=1:j-1
  d(j) = d(j) - L(j,k)^2 * d(k);
 end % for k
 for k=1:j-1
  for i=j+1:n
   L(i,j) = L(i,j) - L(i,k) * L(j,k);
  end % for i
 end % for k
 L(j,j) = 1;
end % for j
```

We have obtained six different ways of coding the LDL^T factorization of a symmetric matrix A. Similar things can be done for the LU factorization of a nonsymmetric matrix. Of course, users are interested in knowing what is the best implementation, that is, the one requiring the smallest computing time. Unfortunately, this is dependent on the computer architecture and also on the programming language. It also depends on the data structure chosen to store L since the performance depends on the way the data is accessed in the computer memory.

Suppose first that L is to be stored in a two-dimensional array or in the lower triangular part of A. In modern computing engines with data caches, it is advantageous to perform operations on data in consecutive memory locations. This increases the cache hit ratio since data is moved into the cache by blocks of consecutive addresses. The data access is by columns for algorithms kji and jki, by rows for ikj and jik and by rows and columns for kij and ijk. Form kji accesses the data by columns and the basic operation is known as an AXPY (for a times x plus y), that is,

$$y = y + \alpha x,$$

where x and y are vectors and α is a scalar. Note that the vector y is stored after it is computed. This particular form was used in the famous LINPACK package [348] in the 1970s.

Form jki also accesses that data by columns and the basic operation is also an AXPY. However, the same column (j) is successively accessed many times. This is known as a generalized AXPY or GAXPY. These algorithms were analyzed for a vector computer by J.J. Dongarra, F.G. Gustavson, and A. Karp [352]. Their notation was slightly different from ours. On vector architectures, the GAXPY jki form was generally the best one.

Let us do some numerical experiments with Matlab. Let B be a random matrix of order 500 and $A = B + B^T$. The matrix we use is $A + 70\,I$ to have an SPD matrix. Each computation was repeated 10 times. The average times t are shown in Table 2.2. They are normalized with the time of the fastest algorithm which is kji. For these implementations the fastest algorithms were kji and kij, but, as we said above, this can be different when using other languages or types of computers. The values of $\frac{\|LDL^T - A\|}{\|A\|}$ show that the six algorithms compute the factorization with the same accuracy which is around $14u$.

Table 2.2. *Normalized computing times and relative errors for the LDL^T factorization*

	kij	kji	ijk	ikj	jik	jki
t	1.0570	1	1.0924	1.3115	2.4925	2.2457
$\frac{\|LDL^T - A\|}{\|A\|} \times 10^{15}$	1.539	1.539	1.643	1.643	1.643	1.643

If L is not stored in the lower triangular part of A, it is better to store it in a one-dimensional array of dimension $n(n-1)/2$. Consecutive entries can be chosen by rows or columns. If consecutive entries are chosen by rows, it is better to use algorithms ikj and jik since the data accesses will be in consecutive addresses. Forms kji and jki are chosen if the data is stored by columns.

So far, we have assumed it was not necessary to do pivoting for a symmetric system. In the next section we describe particular cases where it can be shown that this is the case, at least to be able to run the algorithm to completion without breakdown.

2.3.5 ▪ Positive definite systems

In this part, we assume A is symmetric and positive definite (SPD). We are looking for an LDL^T factorization. In the outer product algorithm, we see that for the first step, $A_2 = B_1 - (1/a_{1,1})a_1 a_1^T$ is a Schur complement, the matrix A being partitioned as

$$A = \begin{pmatrix} a_{1,1} & a_1^T \\ a_1 & B_1 \end{pmatrix}.$$

Therefore, by using Lemma 1.61, A_2 is also positive definite and the next pivot is nonzero. This process can be continued until the last step. All the square matrices involved in the algorithm are SPD and the diagonal entries are strictly positive. All the pivots are nonzero and the algorithm can continue without any need for pivoting. This is summarized in the following result.

Theorem 2.12. *A matrix A has a factorization $A = LDL^T$, where L is a unit lower triangular matrix and D is a diagonal matrix with positive diagonal entries, if and only if A is symmetric and positive definite.*

Proof. Lemma 1.61 and the previous discussion show that if A is positive definite, it can be factored as LDL^T. Conversely, if $A = LDL^T$, then, of course, A is symmetric and if $x \neq 0$,

$$x^T A x = x^T L D L^T x = y^T D y,$$

where $y = L^T x \neq 0$. Note that

$$y^T D y = \sum_{i=1}^{n} d_{i,i} y_i^2 > 0,$$

since the diagonal entries of D are positive. □

We can introduce a diagonal matrix S such that $s_{i,i} = \sqrt{d_{i,i}}$, $i = 1, \ldots, n$. It yields, $S^2 = D$. Let $\bar{L} = LS$. Then,

$$A = LDL^T = LSSL^T = \bar{L}\bar{L}^T.$$

As we have seen above, this decomposition is usually called the *Cholesky factorization* of A. However, this form of factorization is not so often used today since the computation involves square roots. Factorizations like LDL^T are sometimes called the square root-free Cholesky. We will use the generic name "Cholesky" for any LDL^T factorization.

An interesting property of SPD matrices is that there is no growth of the entries of the reduced matrices during the factorization.

Theorem 2.13. *Let A be a symmetric positive definite matrix. Consider the matrices D_i, $i = 1, \ldots, n$ of the outer product algorithm. Then,*

$$\max_k (\max_{i,j} |(D_k)_{i,j}|) \leq \max_{i,j} |a_{i,j}| = \max_i (a_{i,i}).$$

Proof. By Lemma 1.61, the matrices D_k are positive definite. By Lemma 1.62, it is sufficient to consider the diagonal to find the maximum of the absolute values of the entries. We only consider the first step, since the proof is the same for the other steps. Since the diagonal entries are positive, we have

$$\mathrm{diag}(A_2) \leq \mathrm{diag}(B_1).$$

Therefore, either $\max_i(a_{i,i}) = a_{1,1}$ and then, $\max_i(D_1)_{i,i} = a_{1,1}$ or the maximum is on the diagonal of B_1 and then,

$$\max_i (D_1)_{i,i} = \max(a_{1,1}, \max_i [\mathrm{diag}(A_2)_{i,i}]),$$

with $\max_i[\mathrm{diag}(A_2)_{i,i}] \leq \max_i[\mathrm{diag}(B_1)_{i,i}]$. In both cases,

$$\max_i (D_1)_{i,i} \leq \max_{i,j} |a_{i,j}|.$$

□

2.3.6 ▪ Indefinite systems

When factorizing an indefinite matrix (that is, one that is neither positive or negative definite), there can be some problems as shown in the following example,

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 0 \\ 0 & -1/\epsilon \end{pmatrix} \begin{pmatrix} 1 & 1/\epsilon \\ 0 & 1 \end{pmatrix},$$

If ϵ is small, $1/\epsilon$ can be very large and the factorization can be unstable. Indeed, if $\epsilon = 0$, the decomposition does not exist. One can use pivoting to avoid this. However, if symmetry is to

be preserved, pivoting must be done on the diagonal but this does not always solve the problem. Moreover, zero pivots can sometimes be the only alternative. A method to solve these problems was introduced by J.R. Bunch and B.N. Parlett [184] and further developed by J.R. Bunch and L. Kaufman [183]. The remedy is to use diagonal pivoting with either 1×1 or 2×2 pivots. Assume

$$P_1 A P_1^T = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^T & A_{2,2} \end{pmatrix},$$

where $A_{1,1}$ is of order s with $s = 1$ or 2, $\det A_{1,1} \neq 0$ and P_1 is a permutation matrix. This matrix can be factored as

$$P_1 A P_1^T = \begin{pmatrix} I_s & 0 \\ A_{1,2}^T A_{1,1}^{-1} & I_{n-s} \end{pmatrix} \begin{pmatrix} A_{1,1} & 0 \\ 0 & A_{2,2} - A_{1,2}^T A_{1,1}^{-1} A_{1,2} \end{pmatrix} \begin{pmatrix} I_s & A_{1,1}^{-1} A_{1,2} \\ 0 & I_{n-s} \end{pmatrix}.$$

The algorithm can go through provided that $A_{1,1}$ is nonsingular. It can be proved that if A is nonsingular, it is always possible to find a nonzero pivot ($s = 1$) or a nonsingular 2×2 block ($s = 2$). A strategy was devised in [183] to find the block pivots; see also [548]. Another method that can be used for indefinite systems is due to J.O. Aasen [1].

2.4 ▪ Gaussian elimination for H-matrices

There are types of matrices (not necessarily symmetric) other than positive definite matrices for which it is not necessary to use pivoting (at least to obtain a factorization without permutations). Let us first consider matrices A which are diagonally dominant; see Chapter 1 for the definition.

Theorem 2.14. *If A is (row or column) diagonally dominant, then*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular.

Proof. Assume that A is (row) diagonally dominant. Then, $a_{1,1} \neq 0$,. Otherwise all the entries in the first row are 0 and A is singular. We first prove that A_2 is also (row) diagonally dominant, and then, the proof can proceed by induction. The case of the first row has already been handled. For a row i, $i > 1$, we have

$$a_{i,j}^{(2)} = a_{i,j} - \frac{a_{i,1} a_{1,j}}{a_{1,1}},\ 2 \leq i \leq n,\ 2 \leq j \leq n, \quad a_{i,1}^{(2)} = 0,\ 2 \leq i \leq n,$$

$$\sum_{j,j \neq i} |a_{i,j}^{(2)}| = \sum_{j,j \neq i, j \neq 1} |a_{i,j}^{(2)}| \leq \sum_{j,j \neq i, j \neq 1} |a_{i,j}| + \left| \frac{a_{i,1}}{a_{1,1}} \right| \sum_{j,j \neq i, j \neq 1} |a_{1,j}|.$$

But,

$$|a_{1,1}| \geq \sum_{j,j \neq i, j \neq 1} |a_{1,j}| + |a_{1,i}|.$$

Therefore,

$$\begin{aligned} \sum_{j,j \neq i} |a_{i,j}^{(2)}| &\leq \sum_{j,j \neq i, j \neq 1} |a_{i,j}| + \left| \frac{a_{i,1}}{a_{1,1}} \right| (|a_{1,1}| - |a_{1,i}|), \\ &\leq \sum_{j,j \neq i} |a_{i,j}| - \frac{|a_{i,1} a_{1,i}|}{|a_{1,1}|}, \end{aligned}$$

$$\leq |a_{i,i}| - \frac{|a_{i,1}a_{1,i}|}{|a_{1,1}|},$$
$$\leq \left| a_{i,i} - \frac{a_{i,1}a_{1,i}}{a_{1,1}} \right| = |a_{i,i}^{(2)}|.$$

The reduced matrix is also diagonally dominant. This shows that all the pivots are nonzero and the computation can continue. If A is column diagonally dominant, the same proof goes through with A^T. □

We now consider M-matrices. The following result was proved by M. Fiedler and V. Ptàk [452].

Theorem 2.15. *If A is an M-matrix, then*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular.

Proof. See Fiedler's book [451] or A. Berman and R.J. Plemmons [123]. The proof uses Lemma 1.45. □

Let us move to H-matrices. Let B be an M-matrix, we consider the equimodular set Ω_B defined in Chapter 1. It is the set of matrices A satisfying

$$|a_{i,i}| \geq b_{i,i},\ 1 \leq i \leq n,$$
$$|a_{i,j}| \leq |b_{i,j}|,\ i \neq j,\ 1 \leq i,j \leq n.$$

Note that the matrix A is at least as diagonally dominant as the matrix B.

Lemma 2.16. *Let B be an M-matrix. Each matrix $A \in \Omega_B$ is (row) generalized strictly diagonally dominant.*

Proof. There exists a diagonal matrix D (with $\text{diag}(D) > 0$) such that BD is strictly diagonally dominant. Let $A \in \Omega_B$. We have

$$BD \leq M(A)D = M(AD).$$

Therefore, if e is a vector of all ones,

$$0 < BDe \leq M(AD)e,$$

which implies that AD is (row) strictly diagonally dominant. □

Theorem 2.17. *Let B be an M-matrix. For each $A \in \Omega_B$,*

$$A = LU,$$

where L is unit lower triangular and U is upper triangular. In particular, for every H-matrix, there exists an LU factorization.

Proof. From the proof of Lemma 2.16, AD is (row) strictly diagonally dominant. Then, by Theorem 2.14, there exist $\bar{L}$ and $\bar{U}$, lower and upper triangular matrices such that

$$AD = \bar{L}\bar{U}.$$

We have,

$$A = \bar{L}\bar{U}D^{-1},$$

and the result follows. □

Let

$$\beta_D = \frac{\max_i(d_{i,i})}{\min_i(d_{i,i})}.$$

Then,

$$|\ell_{i,j}| \leq \beta_D,$$
$$|u_{i,j}| \leq 2\beta_D \max_i |a_{i,i}|.$$

We have defined the growth factor g_A above. It is also sometimes defined as

$$g_A = \frac{\max_{i,j,k} |a_{i,j}^{(k)}|}{\max_{i,j} |a_{i,j}|}.$$

For an H-matrix, we have $g_A \leq 2\beta_D$, and for an M-matrix, R.E. Funderlic, M. Neumann, and R.J. Plemmons [482] proved that $g_A \leq \beta_D$.

The proof that an H-matrix has an LU factorization can also be established by showing, as in A. Bermann and R.J. Plemmons [123], that all the leading principal minors are nonsingular. Similar results for the case where A is singular have been studied in R.S. Varga and D.-Y. Cai [1099], R.E. Funderlic and R.J. Plemmons [483], and R.E. Funderlic, M. Neumann, and R.J. Plemmons [482].

It has also been shown that any symmetric permutation of an H-matrix A has an LU factorization. However, even if g_A is bounded, it can be large. Consider, for instance, the following example from A.A. Ahac, J.J. Buoni, and D.D. Oleski [5],

$$A_x = \begin{pmatrix} 2 & 0 & -x \\ -x & x & -1 \\ 0 & -1 & x \end{pmatrix}, \; x > 0.$$

The matrix A_x is an M-matrix if $x > \sqrt{2}$, and the LU factorization is

$$A_x = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{x}{2} & 1 & 0 \\ 0 & -\frac{1}{x} & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & -x \\ 0 & x & -\frac{x^2}{2} - 1 \\ 0 & 0 & \frac{x}{2} - \frac{1}{x} \end{pmatrix}.$$

If x is large, the growth factor is large (in fact $O(x)$). This is bad for the stability of the algorithm. This can be avoided by using some form of symmetric pivoting. For M-matrices, A.A Ahac and D.D. Oleski [6] chose the column in the reduced matrix which has the largest column sum. In the example, we choose the second column. The permuted matrix is

$$A'_x = \begin{pmatrix} x & -1 & -x \\ -1 & x & 0 \\ -x & 0 & 2 \end{pmatrix},$$

and

$$A'_x = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{x} & 1 & 0 \\ -1 & \frac{x}{1-x^2} & 1 \end{pmatrix} \begin{pmatrix} x & -1 & -x \\ 0 & x - \frac{1}{x} & 1 \\ 0 & 0 & 2 - x + \frac{x}{x^2-1} \end{pmatrix}.$$

The growth factor of A'_x is bounded independently of x. In [6], it is proved that $g_A \leq n$ for M-matrices. This result was extended to H-matrices in [5].

2.5 ▪ Block methods

Block methods are obtained by partitioning the matrix A into blocks (submatrices). For instance, consider a 3×3 block partitioning. Then A is written as

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}.$$

The matrices $A_{i,i}$ are square of order n_i, $1 \leq n_i \leq n, \sum n_i = n$. To obtain a block LU factorization , we do exactly the analog of the point case, provided the blocks that arise on the diagonal are nonsingular. One way to proceed is the following,

$$A = \begin{pmatrix} I & & \\ L_{2,1} & I & \\ L_{3,1} & L_{3,2} & I \end{pmatrix} \begin{pmatrix} U_{1,1} & U_{1,2} & U_{1,3} \\ & U_{2,2} & U_{2,3} \\ & & U_{3,3} \end{pmatrix}.$$

Note that this is different from the block implementations of point LU factorization since the matrices U_i, are not triangular. For instance, $U_{1,1} = A_{1,1}$. The stability was investigated by J.W. Demmel, N.J. Higham and R.S. Schreiber [337]. Block LU factorization without pivoting is unstable in general, although it has been found to be stable for matrices which are block diagonally dominant by columns, that is, such that

$$\|A_{j,j}^{-1}\|^{-1} \geq \sum_{i \neq j} \|A_{i,j}\|.$$

2.6 ▪ Tridiagonal and block tridiagonal systems

Tridiagonal matrices arise quite often in numerical analysis. So, it is worth investigating their factorization. Let T be a symmetric tridiagonal matrix,

$$T = \begin{pmatrix} \alpha_1 & -\beta_2 & & & \\ -\beta_2 & \alpha_2 & -\beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -\beta_{n-1} & \alpha_{n-1} & -\beta_n \\ & & & -\beta_n & \alpha_n \end{pmatrix}.$$

We assume $\beta_i \neq 0, \forall i$. The minus sign in front of the β_i's is a technical convenience to avoid powers of -1 in some formulas. Assume that the Cholesky factorization $T = LD_L^{-1}L^T$ exists. It is easily obtained as,

$$L = \begin{pmatrix} \delta_1 & & & & \\ -\beta_2 & \delta_2 & & & \\ & \ddots & \ddots & & \\ & & -\beta_{n-1} & \delta_{n-1} & \\ & & & -\beta_n & \delta_n \end{pmatrix}, \quad D_L = \begin{pmatrix} \delta_1 & & & & \\ & \delta_2 & & & \\ & & \ddots & & \\ & & & \delta_{n-1} & \\ & & & & \delta_n \end{pmatrix}.$$

By identification, we have

$$\delta_1 = \alpha_1, \quad \delta_i = \alpha_i - \frac{\beta_i^2}{\delta_{i-1}},\ i = 2, \ldots, n.$$

The computation of the δ_i's requires only $n-1$ additions, multiplications and divisions. Extensions are easily obtained to nonsymmetric tridiagonal matrices as long as pivoting is not needed. A UL factorization $T = UD_U^{-1}U^T$ is also easily obtained, with

$$U = \begin{pmatrix} d_1 & -\beta_2 & & & \\ & d_2 & -\beta_3 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & -\beta_n \\ & & & & d_n \end{pmatrix}, \quad D_U = \begin{pmatrix} d_1 & & & & \\ & d_2 & & & \\ & & \ddots & & \\ & & & d_{n-1} & \\ & & & & d_n \end{pmatrix}.$$

By identification, we obtain

$$d_n = \alpha_n, \quad d_i = \alpha_i - \frac{\beta_{i+1}^2}{d_{i+1}}, \quad i = n-1, \ldots, 1.$$

The LU factorization starts from the top-left corner and the UL factorization from the bottom-right corner. We have seen in Chapter 1 what is the inverse of a tridiagonal matrix. The LU and UL factorizations of these matrices have been used by G.M. [809] to characterize the inverse as follows.

Theorem 2.18. *The entries of the inverse of T are*

$$(T^{-1})_{i,j} = \beta_{i+1} \cdots \beta_j \frac{d_{j+1} \cdots d_n}{\delta_i \cdots \delta_n}, \quad \forall i, \quad \forall j > i,$$

$$(T^{-1})_{i,i} = \frac{d_{i+1} \cdots d_n}{\delta_i \cdots \delta_n}, \quad \forall i.$$

In these products, terms with indices greater than n must be taken equal to 1.

Proof. From [97], [809] and Chapter 1, we know there exist two sequences $\{\nu_i\}, \{\sigma_i\}, i = 1, n$ such that

$$T^{-1} = \begin{pmatrix} \nu_1\sigma_1 & \nu_1\sigma_2 & \nu_1\sigma_3 & \ldots & \nu_1\sigma_n \\ \nu_1\sigma_2 & \nu_2\sigma_2 & \nu_2\sigma_3 & \ldots & \nu_2\sigma_n \\ \nu_1\sigma_3 & \nu_2\sigma_3 & \nu_3\sigma_3 & \ldots & \nu_3\sigma_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \nu_1\sigma_n & \nu_2\sigma_n & \nu_3\sigma_n & \ldots & \nu_n\sigma_n \end{pmatrix},$$

with $\nu_1 = 1$. Every nonsingular matrix of the same form as T^{-1} is the inverse of an irreducible tridiagonal matrix. The matrices of this class have been called *matrices factorisables* in J. Baranger and M. Duc-Jacquet, [97]. They are called *symmetric generator representable semiseparable matrices* in R. Vandebril, M. Van Barel, and N. Mastronardi [1093]. To find all the entries of T^{-1}, it is sufficient to compute its first and last columns. In fact, it is enough to know $2n-1$ quantities since ν_1 can be chosen equal to 1. Note that $2n-1$ is the number of nonzero entries determining T. The elements ν_i and σ_i can be computed in the following stable way. We first compute $\sigma = (\sigma_1, \ldots, \sigma_n)^T$. The first column of T^{-1} is σ, so $T\sigma = e_1$, where e_1 is the first column of the identity matrix. Because of the special structure of the right-hand side, it is natural to use a UL factorization $T = UD_U^{-1}U^T$ to solve the linear system for σ. It yields

$$\sigma_1 = \frac{1}{d_1}, \quad \sigma_i = \frac{\beta_2 \cdots \beta_i}{d_1 \cdots d_{i-1} d_i}, \quad i = 2, \ldots, n.$$

Let $\nu = (\nu_1, \ldots, \nu_n)^T$. The last column of T^{-1} is $\sigma_n \nu$, and therefore $\sigma_n T \nu = e_n$, where e_n is the last column of the identity matrix. To solve this system, with the special structure of the right-hand side, it is easier to use the LU factorization $T = LD_L^{-1}L^T$,

$$\nu_n = \frac{1}{\delta_n \sigma_n}, \quad \nu_{n-i} = \frac{\beta_{n-i+1} \cdots \beta_n}{\delta_{n-i} \cdots \delta_n \sigma_n}, \quad i = 1, \ldots, n-1.$$

Note that

$$\nu_1 = \frac{\beta_2 \cdots \beta_n}{\delta_1 \cdots \delta_n \sigma_n} = \frac{d_1 \cdots d_n}{\delta_1 \cdots \delta_n},$$

but $d_1 \cdots d_n = \delta_1 \cdots \delta_n = \det T$, so $\nu_1 = 1$, since the σ_i's were computed with this scaling. □

This result for inverses of tridiagonal matrices has been extended to nonsymmetric matrices by R. Nabben [847]. It gives a computationally stable and simple algorithm for computing entries of the inverse of T since it involves only Cholesky factorizations that are stable when the matrix T possesses enough properties such as, for instance, being diagonally dominant.

We are also interested in characterizing the decrease of the elements of T^{-1} along a row or column starting from the diagonal element. In P. Concus, G.H. Golub, and G.M. [273], it is proved that if T is strictly diagonally dominant, the sequence $\{\nu_i\}$ is strictly increasing and the sequence $\{\sigma_i\}$ is strictly decreasing. From Theorem 2.18, we have

$$\frac{(T^{-1})_{i,j}}{(T^{-1})_{i,j+1}} = \frac{d_{j+1}}{\beta_{j+1}},$$

and, therefore

$$(T^{-1})_{i,j} = \frac{d_{j+1} \cdots d_{j+\ell}}{\beta_{j+1} \cdots \beta_{j+\ell}} (T^{-1})_{i,j+\ell}, \quad \ell \geq 1.$$

Theorem 2.19. *If T is strictly diagonally dominant, the sequence d_i is such that $d_i > \beta_i$. Hence, the sequence $T_{i,j}^{-1}$ is a strictly decreasing function of j, for $j > i$. Similarly, we have $\delta_i > \beta_{i+1}$.*

Proof. We prove the result by induction. We have $d_n = \alpha_n > \beta_n$. Assume $d_{i+1} > \beta_{i+1}$. Then

$$d_i = \alpha_i - \frac{\beta_{i+1}^2}{d_{i+1}} > \alpha_i - \beta_{i+1} > \beta_i,$$

and this proves the theorem. □

For any nonsingular tridiagonal matrix, we have

$$|[T_k^{-1} e_1]_i| < |[T_{k+1}^{-1} e_1]_i|, \quad i = 1, \ldots, k.$$

At this point, it is interesting to introduce another factorization, which can be obtained from the LU and UL factorizations. It is called the *twisted factorization* or also the BABE (burn at both ends) algorithm. To compute that factorization, we must first select an integer $j, 1 \leq j \leq n$. Let

$$T = (\Phi + \mathcal{L})\Phi^{-1}(\Phi + \mathcal{L}^T),$$

where Φ is a diagonal matrix with diagonal entries ϕ_i, and $\mathcal{L}$ is twisted

$$\mathcal{L} = \begin{pmatrix} 0 & & & & & & & \\ -\beta_2 & 0 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & -\beta_j & 0 & -\beta_{j+1} & & & \\ & & & & \ddots & \ddots & & \\ & & & & & 0 & -\beta_n \\ & & & & & & 0 \end{pmatrix},$$

where the two nonzero entries are in the jth row. By identification, we have

$$\phi_i = \delta_i \quad i = 1, \ldots, j-1, \quad \phi_i = d_i \quad i = j+1, \ldots, n,$$

$$\phi_j = \alpha_j - \frac{\beta_j^2}{\delta_{j-1}} - \frac{\beta_{j+1}^2}{d_{j+1}}.$$

The first $j-1$ entries come from the LU factorization and the last $n-j$ come from the UL factorization. Therefore, whatever is j, every twisted factorization can be computed with the knowledge of the LU and UL factorizations. Only the jth entry requires a special treatment. This twisted factorization leads naturally to a parallel method for a two-processor computer. It can be used to prove the following result.

Theorem 2.20. *The diagonal entries of the inverse of T are*

$$(T^{-1})_{j,j} = \phi_j^{-1}.$$

Proof. This comes from solving a linear system with e_j as the right-hand side and using the twisted factorization. We have to use a different twisted factorization for each value of j. □

Let us specialize the previous results to the case of tridiagonal Toeplitz matrices. The interesting thing is that we are then able to solve analytically the recurrences arising in the Cholesky factorization. Let

$$T_\alpha = \begin{pmatrix} \alpha & -1 & & & \\ -1 & \alpha & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & \alpha & -1 \\ & & & -1 & \alpha \end{pmatrix}.$$

Lemma 2.21. *Let*

$$\delta_1 = \alpha, \quad \delta_i = \alpha - \frac{1}{\delta_{i-1}}, \quad i = 2, \ldots, n.$$

Then, if $\alpha \neq 2$,

$$\delta_i = \frac{r_+^{i+1} - r_-^{i+1}}{r_+^i - r_-^i},$$

where $r_\pm = \frac{\alpha \pm \sqrt{\alpha^2 - 4}}{2}$ are the two solutions of the quadratic equation $r^2 - \alpha r + 1 = 0$. If $\alpha = 2$, then $\delta_i = \frac{i+1}{i}$.

Proof. We set

$$\omega_i = \frac{\gamma_i}{\gamma_{i-1}}.$$

We now have a recurrence on ω_i,

$$\omega_i - \alpha\omega_{i-1} + \omega_{i-2} = 0, \quad \omega_0 = 1, \quad \omega_1 = \alpha.$$

The solution of this linear second order difference equation is well known,

$$\omega_i = c_0 r_+^{i+1} + c_1 r_-^{i+1}.$$

From the initial conditions we have $c_0 + c_1 = 0$. Hence, the solution can be written as

$$\omega_i = c_0(r_+^{i+1} - r_-^{i+1}).$$

When $\alpha = 2$ one can see that $\omega_i = i + 1$ and the result follows. □

From Lemma 2.21, the solutions of the recurrences involved in the Cholesky factorization of T_α can be deduced. When $\alpha \neq 2$ we have,

$$d_{n-i+1} = \frac{r_+^{i+1} - r_-^{i+1}}{r_+^i - r_-^i}.$$

Solving for σ the following result is obtained.

Proposition 2.22. *For the sequence σ_i in T_α^{-1},*

$$\sigma_i = \frac{r_+^{n-i+1} - r_-^{n-i+1}}{r_+^{n+1} - r_-^{n+1}}, \quad i = 1, \ldots, n.$$

□

In particular,

$$\sigma_n = \frac{r_+ - r_-}{r_+^{n+1} - r_-^{n+1}}.$$

It is obvious that for the Toeplitz case, we have the relation $\delta_i = d_{n-i+1}$. Solving for ν, the following result is obtained.

Proposition 2.23. *For the sequence ν_i in T_α^{-1},*

$$\nu_i = \frac{r_+^i - r_-^i}{r_+ - r_-}, \quad i = 1, \ldots, n.$$

□

Now we are able to obtain expressions for the entries of the inverse.

Theorem 2.24. *For $j \geq i$ and when $\alpha \neq 2$,*

$$({T_\alpha}^{-1})_{i,j} = \nu_i \sigma_j = \frac{(r_+^i - r_-^i)(r_+^{n-j+1} - r_-^{n-j+1})}{(r_+ - r_-)(r_+^{n+1} - r_-^{n+1})},$$

where $r_\pm$ are the two solutions of the quadratic equation $r^2 - \alpha r + 1 = 0$.

For $\alpha = 2$, we have

$$(T_\alpha)_{i,j}^{-1} = i\,\frac{n-j+1}{n+1}.$$

□

Regarding the decay of the elements of T_α^{-1}, in this simple case we can obtain useful bounds. Assume that $\alpha > 2$. Then,

$$\frac{\nu_i \sigma_j}{\nu_i \sigma_{j+1}} = \frac{r_+^{n-j+1} - r_-^{n-j+1}}{r_+^{n-j} - r_-^{n-j}} = \frac{r_+^{n-j+1}}{r_+^{n-j}} \left(\frac{1 - r^{n-j+1}}{1 - r^{n-j}} \right) > r_+ > 1,$$

and

$$\nu_i \sigma_j < r_+^{i-j-1} \frac{(1 - r^i)(1 - r^{n-j+1})}{(1 - r)(1 - r^{n+1})}, \quad j \geq i + 1,$$

where $r = \frac{r_-}{r_+} < 1$ and the following result follows.

Theorem 2.25. *If $\alpha > 2$, we have the bound*

$$(T_\alpha{}^{-1})_{i,j} < (r_-)^{j-i} (T_\alpha{}^{-1})_{i,i}, \quad \forall\, i, \quad \forall j \geq i,$$

$$(T_\alpha{}^{-1})_{i,j} < \frac{r_-^{j-i+1}}{1 - r}, \quad \forall\, i, \quad \forall j \geq i + 1.$$

□

Let $\epsilon_1 > 0$ and $\epsilon_2 > 0$ be given. The following estimate holds,

$$\frac{(T_\alpha{}^{-1})_{i,j}}{(T_\alpha{}^{-1})_{i,i}} \leq \epsilon_1 \quad \text{if} \quad j - i \geq \frac{\log \epsilon_1^{-1}}{\log r_+},$$

and

$$(T_\alpha{}^{-1})_{i,j} \leq \epsilon_2 \quad \text{if} \quad j - i + 1 \geq \frac{\log \left[\epsilon_2 (1 - r)\right]^{-1}}{\log r_+}.$$

As an example, Figure 2.8 shows the inverse of T_4, a Toeplitz matrix of order 30 with $\alpha = 4$. The off-diagonal entries decrease quite fast since T_4 is strongly diagonally dominant.

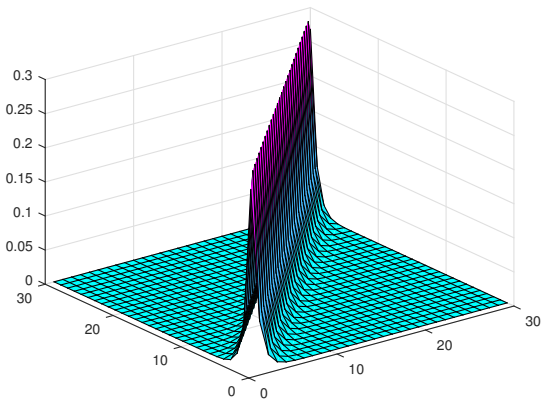


Figure 2.8. *The inverse of T_4*

The previous factorizations are easily extended to block tridiagonal symmetric matrices. Let

$$A = \begin{pmatrix} D_1 & -A_2^T & & & \\ -A_2 & D_2 & -A_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & -A_{m-1} & D_{m-1} & -A_m^T \\ & & & -A_m & D_m \end{pmatrix},$$

each block being of order m. Let L be the block lower triangular part of A. Then, if such a factorization exists, we have

$$A = (\Delta + L)\Delta^{-1}(\Delta + L^T),$$

where Δ is a block diagonal matrix whose diagonal blocks are denoted by Δ_i. By identification, we have

$$\Delta_1 = D_1, \quad \Delta_i = D_i - A_i(\Delta_{i-1})^{-1}A_i^T,\ i = 2, \ldots, m$$

Obtaining this block factorization only involves solving linear systems with matrices Δ_i and several right-hand sides. Whatever the structure of the matrices D_i, matrices Δ_i, $i = 2, \ldots, m$ are dense matrices.

This block factorization (as well as the block twisted factorization) can also be used to characterize the inverse of block tridiagonal matrices, see G.M. [809]. Let us write the block LU and UL factorizations as

$$A = (\Delta + L)\ \Delta^{-1}\ (\Delta + L^T) = (\Sigma + L^T)\ \Sigma^{-1}\ (\Sigma + L),$$

where Δ and Σ are block diagonal matrices whose diagonal blocks are denoted by Δ_i and Σ_i. They are given by block recurrences

$$\begin{cases} \Delta_1 = D_1, \\ \Delta_i = D_i - A_i\ (\Delta_{i-1})^{-1}\ (A_i)^T \end{cases} \quad \begin{cases} \Sigma_m = D_m, \\ \Sigma_i = D_i - (A_{i+1})^T\ (\Sigma_{i+1})^{-1}\ A_{i+1}. \end{cases}$$

A block twisted factorization can be defined for each $j = 2, \ldots, m-1$ as

$$A = (\Phi + \mathcal{L})\Phi^{-1}(\Phi + \mathcal{L}^T)$$

where Φ is a block diagonal matrix and $\mathcal{L}$ has the following twisted block structure

$$\mathcal{L} = \begin{pmatrix} 0 & & & & & & \\ -A_2 & 0 & & & & & \\ & \ddots & \ddots & & & & \\ & & -A_j & 0 & -A_{j+1}^T & & \\ & & & & \ddots & \ddots & \\ & & & & & 0 & -A_m^T \\ & & & & & & 0 \end{pmatrix},$$

where the row with two nonzero blocks is the jth block row. By identification, we have

$$\Phi_i = \Delta_i \quad i = 1, \ldots, j-1, \qquad \Phi_i = \Sigma_i \quad i = j+1, \ldots, m,$$

$$\Phi_j = D_j - A_j\Delta_{j-1}^{-1}A_j^T - A_{j+1}^T\Sigma_{j+1}^{-1}A_{j+1}.$$

With the block twisted factorization, the block jth column X of the inverse can be computed in a straightforward way.

Theorem 2.26. *The jth block column X of A^{-1} is given by*

$$X_j = \Phi_j^{-1},$$
$$X_{j-l} = \Delta_{j-l}^{-1} A_{j-l+1}^T \Delta_{j-l+1}^{-1} \cdots \Delta_{j-1}^{-1} A_j^T \Phi_j^{-1},\ l = 1, \ldots, j-1$$
$$X_{j+l} = \Sigma_{j+l}^{-1} A_{j+l} \Sigma_{j+l-1}^{-1} \cdots \Sigma_{j+1}^{-1} A_{j+1} \Phi_j^{-1},\ l = 1, \ldots, m-j.$$

□

These expressions are valid for any symmetric block tridiagonal matrix. When the matrices A_i are nonsingular, A is said to be *proper*; in this case, the formulas can be simplified. Using the uniqueness of the inverse, we can prove the following result.

Proposition 2.27. *If A is proper, then*

$$\begin{aligned}\Phi_j^{-1} &= A_{j+1}^{-1} \Sigma_{j+1} \cdots A_n^{-1} \Sigma_m \Delta_m^{-1} A_m \cdots \Delta_{j+1}^{-1} A_{j+1} \Delta_j^{-1} \\ &= A_j^{-T} \Delta_{j-1} \cdots A_2^{-T} \Delta_1 \Sigma_1^{-1} A_2^T \cdots \Sigma_{j-1}^{-1} A_j^T \Sigma_j^{-1}.\end{aligned}$$

□

From these relations, we deduce alternate formulas for the other elements of the inverse.

Theorem 2.28. *If A is proper,*

$$X_{j-l} = (A_{j-l}^{-T} \Delta_{j-l-1} \cdots A_2^{-T} \Delta_1)(\Sigma_1^{-1} A_2^T \cdots A_j^T \Sigma_j^{-1}), \quad l = 1, \ldots, j-1$$
$$X_{j+l} = (A_{j+l+1}^{-1} \Sigma_{j+l+1} \cdots A_m^{-1} \Sigma_m)(\Delta_m^{-1} A_m \cdots \Delta_{j+1}^{-1} A_{j+1} \Delta_j^{-1}),\ l = 1, \ldots, m-j.$$

□

As before, the elements of the inverse can be computed in a stable way using the block Cholesky factorization when the matrix is diagonally dominant or positive definite. These formulas give a characterization of the inverse of a proper block tridiagonal matrix.

Theorem 2.29. *If A is proper, there exist two (non-unique) sequences of matrices $\{U_i\}, \{V_i\}$ such that for $j \geq i$*

$$(A^{-1})_{i,j} = U_i V_j^T,$$

with $U_i = A_i^{-T} \Delta_{i-1} \cdots A_2^{-T} \Delta_1$ and $V_j^T = \Sigma_1^{-1} A_2^T \cdots A_j^T \Sigma_j^{-1}$.

□

In other words, A^{-1} can be written as

$$A^{-1} = \begin{pmatrix} U_1 V_1^T & U_1 V_2^T & U_1 V_3^T & \ldots & U_1 V_m^T \\ V_2 U_1{}^T & U_2 V_2^T & U_2 V_3^T & \ldots & U_2 V_m^T \\ V_3 U_1{}^T & V_3 U_2{}^T & U_3 V_3^T & \ldots & U_3 V_m^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ V_m U_1{}^T & V_m U_2{}^T & V_m U_3{}^T & \ldots & U_m V_m^T \end{pmatrix}.$$

The inverse of the matrix of the Poisson model problem constructed with $m = 5$ is shown on Figure 2.9.

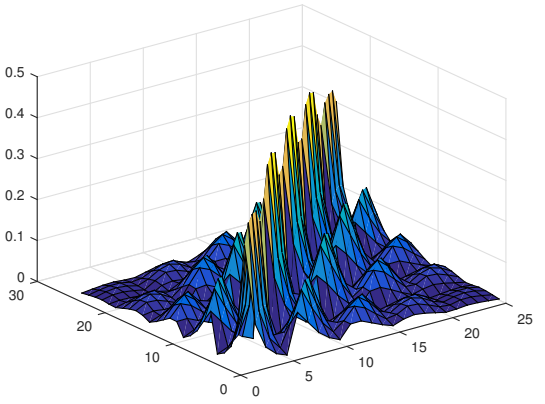


Figure 2.9. *The inverse of the matrix of the Poisson problem,* $m = 5$

2.7 ▪ Rounding error analysis

Let us consider a small example to show some of the difficulties that can happen when solving a linear system using Gaussian elimination. We would like to solve the linear system

$$\begin{aligned} \alpha x_1 + \beta x_2 &= \alpha + \beta, \\ \gamma x_1 + \omega x_2 &= \gamma + \omega. \end{aligned}$$

Obviously, the solution is $x_1 = x_2 = 1$. We choose $\alpha = \gamma = 10^8$, $\beta = 1$, and $\omega = 1 + 10^{-4}$. The exact determinant is 10^4 and the condition number is $2\ 10^{12}$. In double precision arithmetic (fp64), the computed solution and the relative error are respectively,

$$\text{sol} = \begin{pmatrix} 0.9999999999998307 \\ 1.000016927719226 \end{pmatrix}, \quad \text{error} = \begin{pmatrix} -1.693090112553364\ 10^{-13} \\ 1.692771922634506\ 10^{-5} \end{pmatrix}.$$

We have a large relative error on the second component of the solution. It does not come from the factorization since the computed factors are

$$L = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 10^8 & 1 \\ 0 & 9.999999999998899\ 10^{-5} \end{pmatrix}.$$

The error arises from the solve $Ly = b$. The first component x_1 is correctly computed as $10^8 + 1$. The second component x_2, which must be equal to 10^{-4}, is computed as $(10^8 + 1 + 10^{-4}) - (10^8 + 1) \approx 1.000016927719116\ 10^{-4}$. The error is $1.692771911616302\ 10^{-9}$. Note that if we compute the relative error in the infinity norm,

$$\frac{\|x_{ex} - x\|_\infty}{\|x_{ex}\|_\infty} = \frac{1.692771911616302\ 10^{-9}}{1 + 10^8} = 1.692771894688583\ 10^{-17}.$$

The relative error is small, but this is because the norm of the exact solution is large.

If we do a rounding error analysis as in Chapter 1, we see that the computed x_2 is

$$1 + \delta \left(1 - \frac{2\alpha\gamma}{\alpha\omega - \beta\gamma}\right) + O(u^2), \quad |\delta| \leq u.$$

The second term in the parenthesis is roughly $2\ 10^{12}$, and multiplied by δ, it is of the order 10^{-4} and gives a large error which is of the order of $u\|A\|$. Even if the determinant $\alpha\omega - \beta\gamma$ is not small, the error can be large if $\alpha\gamma$ is large, which is what happens in our example.

Wether or not the computed solution can be considered as satisfactory is a matter of debate.

2.7.1 ▪ Triangular systems

Let L be a lower triangular matrix of order n and c be the right-hand side. We would like to solve the linear system $Lx = c$. Straightforwardly, $x_1 = c_1/\ell_{1,1}$, and

$$x_k = \frac{1}{\ell_{k,k}}(c_k - \sum_{i=1}^{k-1} \ell_{k,i} x_k), \quad k = 2, \ldots, n.$$

Apart from the division by the diagonal entry, this is very similar to what we have seen in Chapter 1 for sums and dot products. An analysis is given in N.J. Higham's book [633], Chapter 8, where he considered the computation of

$$(c - \sum_{i=1}^{k-1} a_i b_i) \,/\, b_k.$$

The result of his Lemma 8.4 is that the computed solution y satisfies

$$b_k\, y(1 + \theta_k^{(0)}) = c - \sum_{i=1}^{k-1} a_i b_i (1 + \theta_k^{(i)}), \quad |\theta_k^{(i)}| \leq \gamma_k, \quad \gamma_k = \frac{ku}{1 - ku}.$$

The upper bound is reduced to γ_{k-1} if $b_k = 1$. This yields the backward error bound

$$(L + \Delta L)x = c, \quad |\Delta L| \leq \gamma_n\, |L|.$$

If x_{ex} is the exact solution, the forward error is bounded as

$$\frac{\|x_{ex} - x\|_\infty}{\|x_{ex}\|_\infty} \leq \frac{\text{cond}(L, x_{ex})\gamma_n}{1 - \text{cond}(L)\gamma_n},$$

where

$$\text{cond}(L, x_{ex}) = \frac{\|\, |L^{-1}|\, |L|\, |x_{ex}|\, \|_\infty}{\|x_{ex}\|_\infty}, \quad \text{cond}(L) = \|\, |L^{-1}|\, |L|\, \|_\infty.$$

In absence of underflows and overflows, and if the rounding mode is "round to nearest", these results were improved by S.M. Rump and C.-P. Jeannerod [967]. The bound $|\theta_k^{(i)}| \leq \gamma_k$ can be replaced by $|\theta_k^{(i)}| \leq ku$, and $(k-1)u$ if $b_k = 1$. It yields the backward error bound

$$|\Delta L| \leq D_L\, |L| \leq nu\, |L|,$$

where D_L is a diagonal matrix whose kth diagonal entry is ku. It shows that γ_n can be replaced by nu. Theorem 7.4 in [633] implies that

$$\frac{\|x_{ex} - x\|_\infty}{\|x_{ex}\|_\infty} \leq \frac{\text{cond}(L, x_{ex})nu}{1 - \text{cond}(L)nu}.$$

2.7.2 ▪ General linear systems

As remarked in [633, p. 163], all the variants of LU factorizations have different rounding errors, but they all have the same error bounds and permutations due to pivoting are not a concern. Hence, it is sufficient to analyze the simple following algorithm,

for $k = 1 : n$
  for $j = k : n$
    $u_{k,j} = a_{k,j} - \sum_{i=1}^{k-1} \ell_{k,i} u_{i,j}$
  end
  for $i = k + 1 : n$
    $\ell_{i,k} = (a_{i,k} - \sum_{j=1}^{k-1} \ell_{i,j} u_{j,k})/u_{k,k}$
  end
end

The operations that are involved to compute $u_{k,j}$ and $\ell_{i,k}$ are similar to what we have seen for triangular solves. Therefore, we obtain the same kind of bounds. Higham's results are

$$|a_{k,j} - \sum_{i=1}^{k-1} \ell_{k,i} u_{i,j} - u_{k,j}| \leq \gamma_k \sum_{i=1}^{k} |\ell_{k,i}|\,|u_{i,j}|, \quad j \geq k,$$
$$|a_{i,k} - \sum_{j=1}^{k} \ell_{i,j} u_{j,k}| \leq \gamma_k \sum_{j=1}^{k} |\ell_{i,j}|\,|u_{j,k}|, \quad i > k.$$

It yields the backward error bound,

$$LU = A + \Delta A, \quad |\Delta A| \leq \gamma_n |L|\,|U|.$$

Using the results of S.M. Rump and C.-P. Jeannerod [967], we can replace γ_n by nu. In fact, it can even be replaced by $(n-1)u$, but this does not make such a big difference.

The computed solution of $Ax_{ex} = b$ satisfies $(A + \Delta A)x = b$. Higham's bound is

$$|\Delta A| \leq (3\gamma_n + \gamma_n^2)|L|\,|U|.$$

There is a term proportional to u^2. Moreover, in the bound we have $|L|\,|U|$, and not $|A|$ as we would have expected. Rump and Jeannerod's upper bound is

$$((3n-2)u + (n^2-n)u^2)|L|\,|U|.$$

A more traditional analysis, similar to what was done by J.H. Wilkinson [1120], allows to relate the error bounds to the growth factor.

Theorem 2.30. *At step k of Gaussian elimination, we have*

$$L_k^{-1} A_k = A_{k+1} + E_k,$$

where

$$|(E_k)_{i,j}| \leq Cu \max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|) + O(u^2).$$

Proof. The multipliers (that is, the entries of L^{-1}) that we denote by $m_{i,k}$ are

$$m_{i,k} = fl\left(\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}\right), \; i \geq k+1,$$

and

$$a_{i,j}^{(k+1)} = \begin{cases} 0 & i \geq k+1,\ j = k, \\ fl(a_{i,j}^{(k)} - m_{i,k}\, a_{k,j}^{(k)}) & i \geq k+1,\ j \geq k+1, \\ a_{i,j}^{(k)} & \text{otherwise} \end{cases}$$

Let us first consider $i \geq k+1$,

$$m_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}(1+\delta_{i,k}), \quad |\delta_{i,k}| \leq u.$$

This translates into

$$a_{i,k}^{(k)} - m_{i,k}\, a_{k,k}^{(k)} + a_{i,k}^{(k)}\delta_{i,k} = 0.$$

Let $e_{i,j}^{(k)}$ be the entries of E_k. Then,

$$e_{i,k}^{(k)} = a_{i,k}^{(k)}\delta_{i,k},\ i \geq k+1.$$

For $i \geq k+1$ and $j \geq k+1$, we have

$$a_{i,j}^{(k)} = fl(a_{i,j}^{(k)} - m_{i,k}a_{k,j}^{(k)}).$$

From what we have seen above,

$$a_{i,j}^{(k+1)}(1+\theta^{(0)}) = a_{i,j}^{(k)} - m_{i,k}a_{k,j}^{(k)}(1+\theta^{(1)}), \quad |\theta^{(i)}| \leq 2u.$$

It yields

$$e_{i,j}^{(k)} = a_{i,j}^{(k+1)}\theta^{(0)} + m_{i,k}a_{k,j}^{(k)}\theta^{(1)} = a_{i,j}^{(k+1)}\theta^{(0)} + \frac{a_{i,k}^{(k)}a_{k,j}^{(k)}}{a_{k,k}^{(k)}}(1+\delta_{i,k})\theta^{(1)}.$$

$$|e_{i,j}^{(k)}| \leq \left(2u\left(1 + \max_i \frac{|a_{i,k}^{(k)}|}{|a_{k,k}^{(k)}|}\right)\right)\max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|) + O(u^2).$$

In general, the absolute values of the multipliers are bounded. For partial pivoting, there are smaller than 1. For $e_{i,k}^{(k)}$,

$$|e_{i,k}^{(k)}| \leq u\,|a_{i,k}^{(k)}.$$

Therefore, we have

$$|e_{i,j}^{(k)}| \leq Cu\,\max(|a_{i,j}^{(k+1)}|, |a_{i,j}^{(k)}|) + O(u^2), \quad i \geq k+1,\ j \geq k,$$

where C is a small constant and the other entries of E_k are zero. □

We have the following useful technical lemma.

Lemma 2.31. *If B_k is a matrix whose first k rows are zero, then*

$$L_i B_k = B_k,\ i \leq k.$$

Similarly, $(L_i)^{-1}B_k = B_k$.

Proof. Remember that $L_k = I + \ell_k {e_k}^T$. Then

$$L_i B_k = (I + \ell_i {e_i}^T)B_k = B_k - \ell_i {e_i}^T B_k = B_k,$$

since ${e_i}^T B_k = 0$. □

With this result we can prove the following theorem about LU factorization.

Theorem 2.32. *Let F be defined as*

$$F = F_1 + \cdots + F_{n-1},$$

where

$$(F_k)_{i,j} = \begin{cases} 1 & i \geq k+1,\ j \geq k, \\ 0 & \textit{otherwise} \end{cases}$$

Then

$$A = LU + E,$$

with

$$|E| \leq Cu \max_{k,i,j} |a_{i,j}^{(k)}|\ F + O(u^2),$$

where C is a small constant.

Proof. We have

$$L_k^{-1} A_k = A_{k+1} + E_k \Longrightarrow A_k = L_k A_{k+1} + E_k.$$

By Lemma 2.31,

$$(L_1) \ldots (L_{k-1}) A_k = (L_1) \ldots (L_k) A_{k+1} + E_k.$$

We sum these equalities for $k = 1$ to $n-1$. Most of the terms cancel and we obtain,

$$A = (L_1) \ldots (L_{k-1}) A_n + E_1 + \ldots + E_{n-1}.$$

Therefore, $A = LU + E$ with $E = E_1 + \ldots + E_{n-1}$. Finally, it is easy to see that

$$|E| \leq Cu \max_{k,i,j} |a_{i,j}^{(k)}|\ F + O(u^2).$$

□

The entries of F are bounded by $n-1$. Therefore, $|E|$ being small depends on $\max_{i,j,k} |a_{i,j}^{(k)}|$. Using the growth factor g_A for the computed quantities, the bound can be written as

$$|E| \leq Cu\, g_A \|A\|_\infty F + O(u^2).$$

Taking norms, we have

$$\|E\|_\infty \leq Cu\, g_A\, n^2 \|A\|_\infty + O(u^2).$$

Pivoting techniques are not only used to avoid zero or small pivots, but also to reduce the growth factor. J.H. Wilkinson [1120] gave contrived examples where an exponential growth is observed. Let us consider matrices of order n which have the following form (for $n = 5$),

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

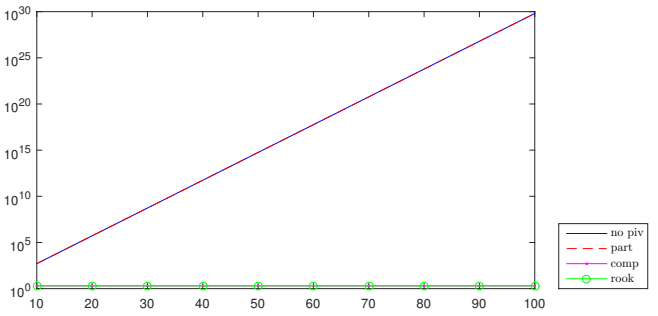


Figure 2.10. *Growth factors, Wilkinson's example*

Figure 2.10 shows the growth factors for several pivoting strategies for increasing n. No pivoting and partial pivoting give a linear increase in this logarithmic scale whence complete and rook's pivoting strategies give bounded growth factors.

It was thought for quite some time that this type of growth arises only in contrived examples. However, in the 1990s, linear systems, arising from practical problems, for which an exponential growth with partial pivoting is obtained appeared in the literature, see S.J. Wright [1123] and L.V. Foster [463]. The average case stability of Gaussian elimination with partial pivoting was studied by L.N. Trefethen and R.S. Schreiber [1073]. They looked at random matrices of order ≤ 1024. The average growth factor was approximately $n^{3/2}$ for partial pivoting and $n^{1/2}$ for complete pivoting. Although Gaussian elimination with partial pivoting gives accurate results in most cases, these examples show that, when using this algorithm, we must carefully analyze the results.

For complete pivoting, Wilkinson [1120] showed that the growth factor is bounded (in the absence of rounding) by

$$|a_{i,j}^{(k)}| \leq k^{\frac{1}{2}}(2 \cdot 3^{\frac{1}{2}} \cdots k^{\frac{1}{k-1}})^{\frac{1}{2}} \max_{i,j} |a_{i,j}|.$$

It was conjectured that, in this case, $g_A \leq n$. C.W. Cryer [292] proved that this is true for $n \leq 4$. However, the conjecture was shown to be false for $n > 4$ if rounding errors are allowed by N. Gould [552]. A. Edelman and M. Ohlroch [392] modified Gould's counterexample to show that the conjecture is also false in exact arithmetic. N.J Higham and D.J. Higham [635] exhibited some matrices of practical interest which have a growth factor of at least $n/2$ for complete pivoting.

2.7.3 ▪ Probabilistic bounds

Since the bounds obtained from rounding error analyses are often pessimistic, some researchers tried to use probability theory. Probably the first ones to do this were J. von Neumann and H.H. Goldstine [1105, 538]. This was also considered by P. Henrici [623], as well as some other authors starting in the 1960s up to the end of the century. These authors considered the perturbation terms due to rounding errors as independent random variables with mean zero and assumed them to be from a uniform or a normal distributions. This approach contributed to a rule of thumb saying that, in rounding error upper bounds, the order of the matrix n can be replaced by $\sqrt{n}$. However, we have seen in Chapter 1 that rounding errors are not random. Moreover, in many cases they are not independent. For probabilistic bounds, see also L.N. Trefethen and R.S. Schreiber [1073], M.C. Yeung and T.F. Chan [1141], and M.C. Yeung [1139].

A recent revival of probabilistic rounding error analysis was proposed by N.J. Higham and T. Mary [636]. They assumed that rounding errors associated with every pair of operands are independent random variables of mean zero without any assumption on the probability distribution. They used Hoeffding's inequality [638] to bound expressions like $\prod_{i=1}^{n}(1+\delta_i)^{\rho_i}$ where $|\delta_i| \leq u$ and $\rho_i = \pm 1$. For any $\lambda > 0$, this is equal to $1+\theta_n$ with $|\theta_n| \leq \tilde{\gamma}_n(\lambda)$ with probability

$$P(\lambda) = 1 - 2\exp\left(-\frac{\lambda^2(1-u)^2}{2}\right),$$

where

$$\tilde{\gamma}_n(\lambda) = \exp\left(\lambda\sqrt{n}u + \frac{nu^2}{1-u}\right) - 1.$$

This result is used to obtain probabilistic bounds for

$$(c - \sum_{i=1}^{k-1} a_i b_i) \,/\, b_k.$$

The computed solution y satisfies

$$b_k\, y(1+\mu_0) = c - \sum_{i=1}^{k-1} a_i b_i (1+\mu_i), \quad |\mu_i| \leq \tilde{\gamma}_k(\lambda),$$

with probability $Q(\lambda, k) = 1 - k(1 - P(\lambda))$.

For triangular systems, it yields $|\Delta L| \leq \tilde{\gamma}_n(\lambda)\,|L|$ with probability $Q(\lambda, n(n+1)/2)$. For the LU factorization, the bound $|\Delta A| \leq \tilde{\gamma}_n(\lambda)\,|L|\,|U|$ with probability $Q(\lambda, n^3/3+n^2/2+n/6)$ is obtained. When,

$$\lambda\sqrt{n}u + \frac{nu^2}{1-u} < 1,$$

the constant $\tilde{\gamma}_n(\lambda)$ is bounded from above by $\lambda\sqrt{n}u + O(u^2)$. The authors of [636] were aware that their model is not very realistic and they provided many numerical experiments to show that, nevertheless, it can give upper bounds that are better than the deterministic upper bounds in many cases.

2.7.4 ▪ Symmetric linear systems

For symmetric positive definite matrices, what has been mainly considered in the literature is the Cholesky factorization $A = LL^T$, for which error bounds can be derived in the same way as those for the LU factorization. The standard result from [633] is,

$$A + \Delta A = LL^T, \quad |\Delta A| \leq \gamma_{n+1}\,|L|\,|L^T|.$$

There is a γ_{n+1}, and not a γ_n, in the bound because of the square root in Cholesky's algorithm. Note that

$$\||L|\,|L^T|\,\| \leq n\|A\|.$$

The previous bound was improved in [967] as

$$|\Delta A| \leq (n+1)\,|L|\,|L^T|.$$

We observe, as in [633], that ΔA is not necessarily symmetric. Moreover, in these results, we have to add the hypothesis that the algorithm runs to completion, since this is not obvious in floating-point arithmetic; on this issue, see J.W. Demmel [332].

2.8 ▪ Perturbation analysis

We have been concerned with the consequences of running the Gaussian elimination algorithm in floating-point arithmetic, but we should also look at the sensitivity of the solution to perturbations of the data. When the matrix A or the right-hand side b are constructed (from scratch or from some other computations) some errors could also have been introduced. For works on perturbation theory, see, for instance, G.W. Stewart and J.-G. Sun [1047], J.-G. Sun [1056, 1055], G.W. Stewart [1043, 1044], Z. Drmač, M. Omladič, and K. Veselič [360], X.-W. Chang, C.C. Paige, and G.W. Stewart [232], X.-W. Chang and C.C. Paige [231], F.M. Dopico and J.M. Molera [355], and X.-W. Chang and D. Stehlé [233].

There are different ways to study the effect of perturbations that differ mainly in the way they are measured; see N.J. Higham [631, 633]). The oldest way is known as *normwise error analysis*. Let x and y be such that

$$\begin{aligned} Ax &= b, \\ (A + \Delta A)y &= b + \Delta b. \end{aligned}$$

ΔA and Δb are chosen such that

$$\|\Delta A\| \leq \alpha\omega, \quad \|\Delta b\| \leq \beta\omega,$$

where ω is given, α (resp. β) is 0 or $\|A\|$ (resp. $\|b\|$), depending on whether A, or b, or both are perturbed. The number ω defines the normwise relative perturbation. Then, we can bound the solution of the perturbed system.

Lemma 2.33. *If $\xi = \alpha\omega\|A^{-1}\| < 1$, then*

$$\frac{\|y\|}{\|x\|} \leq \frac{1}{1-\xi}\left(1 + \frac{\xi\beta}{\alpha\|x\|}\right).$$

Proof. First, we must show that, with our hypothesis, $A + \Delta A$ is nonsingular. We observe that

$$A + \Delta A = A(I + A^{-1}\Delta A),$$

and, with the hypothesis,

$$\|A^{-1}\Delta A\| \leq \alpha\omega\|A^{-1}\| = \xi < 1.$$

Then, $A + \Delta A$ is nonsingular by Lemma 2.3.3 of [547]. We have

$$(I + A^{-1}\Delta A)y = A^{-1}(b + \Delta b) = x + A^{-1}\Delta b.$$

Taking norms,

$$\|y\| \leq \frac{1}{1 - \alpha\omega\|A^{-1}\|}(\|x\| + \beta\omega\|A^{-1}\|).$$

But,

$$\omega \leq \frac{1}{\alpha\|A^{-1}\|} \Longrightarrow \|y\| \leq \frac{1}{1 - \alpha\omega\|A^{-1}\|}\left(\|x\| + \xi\frac{\beta}{\alpha}\right).$$

Therefore,

$$\frac{\|y\|}{\|x\|} \leq \frac{1}{1-\xi}\left(1 + \xi\frac{\beta}{\alpha\|x\|}\right).$$

Note that if $\alpha = \|A\|$ and $\beta = \|b\|$, then

$$\frac{\|y\|}{\|x\|} \leq \frac{1+\xi}{1-\xi}.$$

□

The relative difference with the exact solution x can also be bounded.

Theorem 2.34. *Under the hypothesis of Lemma 2.33,*

$$\frac{\|x-y\|}{\|x\|} \leq \omega \left(\|A^{-1}\| \frac{\alpha\|x\| + \beta}{\|x\|} \right) \left(\frac{1}{1-\xi} \right).$$

Proof. We have,

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

Taking norms, the result follows easily. □

If $\alpha = \|A\|$ and $\beta = \|b\|$,

$$\frac{\|x-y\|}{\|x\|} \leq 2\omega \|A\| \, \|A^{-1}\| \left(\frac{1}{1-\xi} \right).$$

The *normwise condition number* of the problem is defined as

$$K_T(A,b) = \|A^{-1}\| \frac{\alpha\|x\| + \beta}{\|x\|}.$$

Note that, if $\beta = 0$, this reduces to $K_T(A) = \|A^{-1}\| \, \|A\|$, which is the condition number related to inversion. This condition number is frequently denoted $\kappa(A)$. The subscript T refers to A. Turing [1078] who first introduced this condition number. It measures the sensitivity of the solution to perturbations.

The bounds, like those in Theorem 2.34, are, quite often, overly conservative. T.F. Chan and D.E. Foulser [221] introduced the concept of "effective well-conditioning". They considered the SVD of the matrix A, $A = U\Sigma V^T$ and perturbations of the right-hand side. Let σ_i, $i = 1, \ldots, n$ be the singular values and $P_k = U_k U_k^T$, where the columns of U_k are the k last columns of U for $1 \leq k \leq n$. If $Ax = b$ and $Ay = b + \Delta b$, then

$$\frac{\|x-y\|}{\|x\|} \leq \frac{\sigma_{n+1-k}}{\sigma_n} \left(\frac{\|P_k b\|}{\|b\|} \right)^{-1} \frac{\|\Delta b\|}{\|b\|}, \quad 1 \leq k \leq n.$$

If $k = n$, this is the standard bound involving $\kappa(A)$. However, if, for some $k < n$, a large fraction of b lies in span(U_k) and the ratio of singular values is not too large, then the solution x is relatively insensitive to perturbations in b.

Other results can be obtained for some special classes of matrices. For instance, bounds for row diagonally dominant matrices were described by M. Dailey, F.M. Dopico, and Q. Ye [306].

The *normwise backward error* measures the minimal distance to a perturbed problem which is solved exactly by the computed solution y. Let

$$\eta_T = \inf\{\omega \,|\, \omega \geq 0, \ \|\Delta A\| \leq \omega\alpha, \ \|\Delta b\| \leq \omega\beta, \ (A + \Delta A)y = b + \Delta b\}.$$

The normwise backward error has been characterized by J.-L. Rigal and J. Gaches [945] in the following result.

Theorem 2.35. *Let $r = b - Ay$ be the residual of the computed solution. Then*

$$\eta_T = \frac{\|r\|}{\alpha\|y\| + \beta}.$$

□

We remark that the bound on the forward error $\frac{\|x-y\|}{\|x\|}$ is approximately the product of the condition number and the backward error, as it was already stated in Chapter 1.

Another type of analysis considers componentwise perturbations. It was introduced by F.L. Bauer and R. Skeel (see [1016]]. It allows to study perturbations on individual entries of A and b. This is particularly useful for sparse matrices. We consider perturbations ΔA and Δb such that

$$|\Delta A| \leq \omega E, \quad |\Delta b| \leq \omega f.$$

We have the same kind of results as for the normwise perturbations.

Theorem 2.36. *If $\omega \| \, |A^{-1}|E\|_\infty < 1$,*

$$\frac{\|x - y\|_\infty}{\|x\|_\infty} \leq \omega \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty} \frac{1}{1 - \omega \| \, |A^{-1}|E\|_\infty}.$$

□

S. Chandrasekaran and I.C.F. Ipsen gave componentwise bounds for the error [228, 229].

The *componentwise condition number* is defined as

$$K_{BS}(A, b) = \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty}.$$

The subscript BS refers to Bauer and Skeel. As above, we introduce the backward error,

$$\eta_{BS} = \inf\{\omega \, |, \omega \geq 0, \ |\Delta A| \leq \omega E, \ |\Delta b| \leq \omega f, \ (A + \Delta A)y = b + \Delta b\}.$$

W. Oettli and W. Prager [868] proved the following characterization of η_{BS}.

Theorem 2.37.

$$\eta_{BS} = \max_i \frac{|(b - Ay)_i|}{(E|y| + f)_i}.$$

□

An algorithm is said to be *backward stable* when the backward error is of the order of the unit roundoff u. Unfortunately, Gaussian elimination with partial pivoting is both normwise and componentwise backward unstable. There are examples where η_T or η_{BS} are large compared to machine precision. Despite this fact, Gaussian elimination can be used safely on most practical examples. Moreover, there are some remedies to this (potential) backward instability; see Section 2.10.

2.9 ▪ Scaling

Scaling has been used for, at least, two goals: improve the condition number of the matrix and improve the behavior of Gaussian elimination. On the first goal, see G.E. Forsythe and E.G. Straus [462], F.L. Bauer [102, 103], A. van der Sluis [1083], J.R. Bunch [182], C.A. McCarthy and G. Strang [792], and G.H. Golub and J.M. Varah [549]. Scaling is also used, for instance, as a preprocessing step in eigenvalue computations.

Let D_1 and D_2 be two nonsingular diagonal matrices. The system $Ax = b$ is transformed into

$$A'y = (D_1 A D_2)y = D_1 b,$$

and the solution x is recovered by $x = D_2 y$. The left multiplication by D_1 is a row scaling and right multiplication by D_2 is a column scaling. The entry $a_{i,j}$ is changed into $d_i^1 d_j^2 a_{i,j}$ where $d_i^\ell,\ \ell = 1, 2$ are the diagonal entries of D_ℓ.

Note that if one uses Gaussian elimination with partial pivoting, row scaling influences the choice of the pivots. On this issue, see G.E. Forsythe and C.B. Moler [461], A. van der Sluis [1084], A.R. Curtis and J.K. Reid [299] in 1972 and G.W. Stewart [1042]. Classical strategies can be found in A.R. Curtis and J.K. Reid [299]. Other proposals were described by W.W. Hager [607]. A common strategy for row scaling is to divide the entries of a row by the maximum norm of the row. R.D. Skeel [1016] showed that a good scaling matrix is choosing the diagonal entries of D_1 as $d_i = (|A|\,|y|)_i$ where y is the computed solution. Of course, this is impractical since the solution y depends on the scaling. However, if an approximation c of the solution is known, then A could be scaled by $(|A|\,|c|)_i$.

More recent discussions of scaling are found in O. Livne and G.H. Golub [764], I.S. Duff and S. Pralet [378], P.R. Amestoy, I.S. Duff, D. Ruiz, and B. Uçar [32], P.A. Knight and D. Ruiz [707], P.A. Knight, D. Ruiz, and B. Uçar [708], and Z. Allen-Zhu, Y. Li, R. Oliveira, and A. Wigderson [13].

Another related possibility is to permute the rows and columns to bring large entries on the diagonal; see, for instance, I.S. Duff and J. Koster [375, 376].

So far, no scaling strategy has been shown to give consistently better results than not using any scaling, although many different strategies have been proposed over the years. As we said above, it has been argued that the real role of scaling is to alter the pivoting sequence. This can result in a better or worse solution. The rule of thumb given by G. Poole and L. Neal [923] is that scaling can be useful if it leads to a system which is more diagonally dominant, but otherwise it should be avoided. But, as for any rule of thumb, there are exceptions.

2.10 ▪ Iterative refinement

From the rounding error analysis, we have shown that the computed solution y satisfies

$$(A + H)y = b,$$

with

$$\|H\|_\infty \leq u\, C \|A\|_\infty,$$

if the growth factor is bounded. Let $r = b - Ay$ be the residual. Then, obviously

$$\|r\|_\infty \leq \|H\|_\infty \|y\|_\infty \leq uC\|A\|_\infty \|y\|_\infty.$$

If the constant C and the norms are not too large, Gaussian elimination usually produces a small residual. But, unfortunately, small residuals do not always imply high accuracy in the solution.

Let $e = x - y$ be the error vector, then $Ae = b - Ay = r$. Therefore, a natural idea to improve the computed solution is to solve $Ae = r$. This will produce a computed solution $\tilde{e}$, satisfying

$$(A + \tilde{H})\tilde{e} = r$$

and we can set $\tilde{x} = y + \tilde{e}$ as a new approximation to the solution. Eventually, we can iterate this process. This algorithm is known as *iterative refinement*. Of course, iterative refinement is just one example of a very simple iterative method and one can use any suitable iterative method to solve $Ae = r$.

Myrick Hascall Doolittle (1830-1913) was probably the first to use iterative refinement [354] in 1881. He was solving linear systems arising from least squares problems by hand with the help of multiplication tables. He computed the solutions with only a few decimal digits and used iterative refinement to improve the solution. He also realized that the residual has to be computed in higher precision than the one used for the computation; see [160] for details.

Iterative refinement started to be used in the 1960s, see R.S. Martin, G. Peters, and J.H. Wilkinson [787] and C.B. Moler [833] who studied iterative refinement in floating-point arithmetic.

The main question is to decide to which precision the residual r has to be computed since we are not able to compute the exact answer. If we have $\tilde{r} = fl(b - Ay)$ and $(A + \tilde{H})\tilde{e} = \tilde{r}$, R.D. Skeel [1016] showed that computing the residual vector to the same precision as the original computation is enough to make Gaussian elimination with partial pivoting backward stable. This is shown in the following theorem.

Theorem 2.38. *If $u|A|\,|A^{-1}|$ is sufficiently small, one step of iterative refinement with single precision residual computation is componentwise backward stable.*

□

On this issue, see also M. Jankowski and H. Wozniakowski [668].

However, one step of iterative refinement in working precision only gives a small backward error. It does not guarantee a better accuracy. If this is desired, the residual must be computed in higher precision and then rounded to the lower precision. Iterative refinement was studied by A. Kiełbasiński [701], and N.J. Higham [632]. Numerical experiments with sparse matrices were discussed in Z. Zlatev [1166]. A Chebyshev acceleration of iterative refinement was proposed by M. Arioli and J. Scott [42]. Error bounds from extra-precise iterative refinement were given by J.W. Demmel, Y. Hida, W. Kahan, X.S. Li, S. Mukherjee, and E.J. Riedy [336]. Forward and backward stability results were given by S. Oishi, T. Ogita, and S.M. Rump [870] for an algorithm using an approximate inverse.

Recently, there has been a renewal of interest into iterative refinement because of the availability of computing engines capable of doing half, single and double precision computations. One can compute an approximate solution in low precision and then improve the result using an higher precision.

Results in single and double precision were presented by A. Buttari, J. Langou, J. Langou, P. Luszczek, and J. Kurzak [188]. Three precisions were used by E.C. Carson and N.J. Higham [205]. N.J. Higham described a GMRES-based iterative refinement in two and three precisions [634]. Mixed-precision iterative refinement on GPUs was studied by A. Haidar, H. Bayraktar, S. Tomov, J.J. Dongarra, and N.J. Higham [609] using a multiprecision LU factorization and GMRES. When using half precision, some scaling of the data has to be used. A five-precision GMRES-based iterative refinement algorithm was presented by P.R. Amestoy, A. Buttari, N.J. Higham, J.Y. L'Excellent, T. Mary, and B. Vieublé [24]. For mixed precision iterative refinement, see also E. Oktay and E.C. Carson [871, 872].

2.11 ▪ Software

Since their arrival during and after World War II, computer architectures have undergone tremendous changes as well as a spectacular increase of their computing power. On the first digital computers, coding was done directly using the machine instruction set. This was obviously machine-dependent. High-level languages and compilers started to appear at the end of the 1950s. FORTRAN (FORmula TRANslator) was first released by IBM in 1957. It was widely and quickly adopted by programmers. In 1958, FORTRAN II allowed user-written subroutines as well as COMMON statements. FORTRAN IV was developed from 1961 to 1966. It added some data types and the logical IF statement. In the 1960s, other computer manufacturers started proposing FORTRAN compilers to their customers. A new enriched standard, FORTRAN 77, was approved in 1978 by the American National Standards Institute (ANSI). It introduced the ELSE statement, DO loops extensions, and the CHARACTER data type. FORTRAN 77 remained the standard for a long time. The next standard, Fortran 90 (note the move to lower case letters), was adopted only in 1991-92 with many new features like free-form source input, recursive procedures, modules, array features, dynamic memory allocation, and pointers.

Algol was an interesting language developed in the 1950s and 1960s. It was used in many papers and books to describe algorithms. Unfortunately, Algol did not have much success, particularly in the USA because it was not so popular in the industry, and IBM, the most influential manufacturer at that time, was pushing FORTRAN.

The C language was originally developed at Bell Labs in the beginning of the 1970s and C++, with object-oriented features, was started in 1979.

With the sequential computers of the 1960s and 1970s, and with FORTRAN or C, the LU factorization can be coded in an almost straightforward way as a translation of the mathematical formulas. However, the programmers had already to be careful. A dense matrix A is stored as a two-dimensional array. Such an array is stored by columns in FORTRAN, that is, the entries of a column are in consecutive memory locations. Accessing the array in a row-wise fashion within the matrix could involve successive memory references to locations separated from each other by a large increment. In computers using memory pages to control memory usage this can result in a large number of page swaps. In C, a two-dimensional array is stored by rows and the indices start at 0, instead of 1 in FORTRAN. Hence, it is likely that the implementations must be different.

LINPACK is a package of mathematical software for solving problems in linear algebra, mainly dense and banded linear systems and factorizations that was developed at the end of the 1970s [348]. It was written in FORTRAN 66.

In the second half of the 1970s the computer landscape changed with the introduction of vector computers. The most notable evolution was the use of vector registers and of pipelined functional units. If one wants to do (say) the addition of two vectors, those are segmented into chunks by the compiler, loaded in the vector registers, and then the addition pipelined functional unit can be used at full speed provided there is a large enough bandwidth between the memory and the vector registers. With this kind of computers one can deliver one result each clock period and more if there are several independent functional units. However, to obtain the best performances, algorithms have to be expressed in vector form. Therefore, in the late 1970s there was an effort to standardize vector operations for use in scientific computations. A de facto standardization of low-level vector operations was done, leading to the Basic Linear Algebra Subroutines (BLAS), now known as Level 1 BLAS or BLAS1 [615, 726]. These routines can be implemented efficiently in assembly language by the manufacturer. They brought modularity, portability, and efficiency. LINPACK was using BLAS1 for its column-oriented algorithms. A Matlab-like translation of the code `dgefa` doing the factorization in LINPACK is following.

```
function [A,ipvt,info,L,U,P] = dgefa(A);
[n,m] = size(A);
ipvt = zeros(n,1);
pp = 1:n;
info = 0;
nm1 = n - 1;
if nm1 < 1
 ipvt(n) = n;
 if A(n,n) == 0
  info = n;
 end % if
 L = 1;
 U = A(n,n);
 return
end % if
L = eye(n,n);
for k = 1:nm1
 kp1 = k + 1;
 [~,I] = max(abs(A(k:n,k)));
 l = I(1) + k - 1; % pivot index, BLAS1 idamax
 ipvt(k) = l;
 pp( [k, l] ) = pp( [l, k] ); % this is not in LINPACK dgefa
 if A(l,k) == 0 % zero pivot implies this column is already
     triangularized
  info = k;
  continue
 end % if
 if l ~= k % interchange if necessary
  t = A(l,k);
  A(l,k) = A(k,k);
  A(k,k) = t;
 end % if
 t = -1 / A(k,k); % compute multipliers
 A(k+1:n,k) = t * A(k+1:n,k); % BLAS1 dscal
 L(k+1:n,k) = -A(k+1:n,k); % this is not in LINPACK dgefa
 % row elimination with column indexing
 for j = kp1:n
  A(l,j);
  if l ~= k % interchange if necessary
   A(l,j) = A(k,j);
   A(k,j) = t;
  end % if
  A(k+1:n,j) = t * A(k+1:n,k) +  A(k+1:n,j); % BLAS1 daxpy
 end % for j
 % apply the permutation to the previous columns (this is not
     in LINPACK dgefa)
 % this is to obtain L
 L([l,k],1:k-1) = L([k,l],1:k-1);
end % for k
ipvt(n) = n;
```

```
if A(n,n) == 0
 info = n;
end
U = triu(A);
P = eye(n);
P = P(pp,:);
```

P is a permutation matrix such that $PA = LU$. It must be noted that the original code does not contain the last arguments `L, U, P` because `dgefa` computes only the U factor and the multipliers needed to solve a linear system with the routine `dgesl`. Nevertheless, this shows how LU factorization was coded at that time. The previous code is quite slow when executed in Matlab. A faster one is the following.

```
function [L,U,P,rho] = gauss_pp(A);
[n,m] = size(A);
pp = 1:n;
maxA = norm(A(:),inf);
rho = maxA;
for k = 1:n-1
 [~,I] = max(abs(A(k:n,k))); % find largest element in column
 row = I(1) + k - 1;
 % Permute largest element into pivot position
 A([k,row],:) = A([row,k],:);
 pp([k,row]) = pp([row,k]);
 i = k+1:n;
 if A(k,k) ~= 0
  A(i,k) = A(i,k) / A(k,k);  % multipliers
 end % if
 if k+1 <= n
  j = k+1:n;
  A(i,j) = A(i,j) - A(i,k) * A(k,j);
  rho = max(rho,max(max(abs(A(i,j)))));
 end % if
end % for k
L = tril(A,-1) + eye(n,n);
U = triu(A);
P = eye(n,n);
P = P(pp,:);
rho = rho / maxA;
```

In `gauss_pp`, `rho` is the growth factor. For coding the LU factorization, kij, kji,... coding can be used similarly to what we have seen for the symmetric case. LINPACK used a kji algorithm coded with BLAS1 calls. The six variants were compared on vector computers in [882].

The rise of microprocessors in the 1980s and 1990s was the demise of vector processors, which were proprietary and dedicated designs of their manufacturers. Many of the machines based on these microprocessors had a deep memory hierarchy with multiple levels of cache memory. LINPACK was inefficient on these computers. To regain efficiency it was necessary to develop successively Level 2 BLAS with matrix-vector operations and Level 3 BLAS with matrix-matrix operations. Matrix-matrix operations offer the proper level of modularity for performance and portability across a wide range of computer architectures.

LAPACK was started to be developed at the end of the 1980s. It was necessary to restructure the algorithms to a block form in terms of matrix-vector and matrix-matrix products to be able to use BLAS2 and BLAS3. Originally in Fortran 77, LAPACK is now written in Fortran 90. Unfortunately, the codes became much more involved and difficult to understand than those in LINPACK, see [769]. Over the years there were at least four different versions of the routine DGETRF for the LU factorization in double precision: a Crout BLAS3 version, an implementation of the recursive algorithm in [1066], a left-looking algorithm, and a blocked right-looking BLAS3 version. To show how more complicated are these implementations, let us look at a Maltlab-like translation of the right-looking version.

```
function [A,ipiv,info,L,U,P] = dgetrf(A,nb);
% Test the input parameters
[m,n] = size(A);
info = 0;
if m < 0
 info = -1;
elseif n < 0
 info = -2;
end % if
if info ~= 0
 error('dgetrf, info ~= 0')
end % if
ipiv = zeros(m,1);
% Quick return if possible
if m == 0 || n == 0
 ipiv = 0; L = 0; U = 0; P = 0;
 return
end % if
if nb >= n || nb <= 1
 [A,ipiv,info,L,U,P] = dgetf2(A);
 return
end % if
% Use blocked code
for j=1:nb:min(m,n)
 jb = min(min(m, n) - j + 1,nb);
 % Factor diagonal and subdiagonal blocks and test for exact
    singularity
 [A2,ipiv2,iinfo] = dgetf2(A(j:m,j:j+jb-1));
 A(j:m,j:j+jb-1) = A2;
 % Adjust info and the pivot indices
 jj = min(m,j+jb-1);
 ipiv(j:jj) = j - 1 + ipiv2(1:jj-j+1);
 if info == 0 && iinfo > 0
  info = iinfo + j - 1;
 end % if
 % Apply interchanges to column 1:j-1
 for i=j:j+jb-1
  A([i,ipiv(i)],1:j-1) = A([ipiv(i),i],1:j-1); % dlaswp
 end % for i
 if j+jb <= n
```

```
  % Apply interchanges to column j+jb:n
  for i=j:j+jb-1
   A([i,ipiv(i)],j+jb:n) = A([ipiv(i),i],j+jb:n); % dlaswp
  end % for i
  % Compute block row of U
  LA = tril(A(j:j+jb-1,j:j+jb-1),-1) + eye(jb,jb);
  X = LA \ A(j:j+jb-1,j+jb:n); % dtrsm
  A(j:j+jb-1,j+jb:n) = X;
  if j+jb <= m
   % Update trailing submatrix
   A(j+jb:m,j+jb:n) = A(j+jb:m,j+jb:n)...
    - A(j+jb:m,j:j+jb-1) * A(j:j+jb-1,j+jb:n); % dgemm
  end % if j+jb <= m
 end % if j+jb <= n
end % for j
% this is not in dgetrf
% Assume m >= n
L = zeros(m,n);
L(1:m,1:n) = tril(A(1:m,1:n),-1) + eye(m,n);
U = triu(A(1:n,1:n));
P = eye(m,m);
for i=1:n
 P([i,ipiv(i)],:) = P([ipiv(i),i],:);
end % for i
```

In that code `nb` is the panel width and `dgetrf2` does the LU factorization of a panel. The corresponding names of the BLAS3 functions as shown as comments. The recursive code derived from [1066] is even more involved.

```
function [A,ipiv,info,L,U,P] = dgetrf_r(A);
% Test the input parameters
[m,n] = size(A);
info = 0;
if m < 0
 info = -1;
elseif n < 0
 info = -2;
end % if
if info ~= 0
 error('dgetrf\_r, info ~= 0')
end % if
ipiv = zeros(m,1);
% Quick return if possible
if m == 0 || n == 0
 ipiv = 0; L = 0; U = 0; P = 0;
 return
end % if
% Compute machine safe minimum
sfmin = realmin;
small = 1 / realmax;
if small >= sfmin
```

```
 sfmin = small * (1 + eps );
end % if
nstep = min(m,n);
for j=1:nstep
 kahead = bitand(int64(j),int64(-j));
 kstart = j + 1 - kahead;
 kcols = min(kahead,m-j);
 % Find pivot
 [~,I] = max(abs(A(j:m,j)));
 jp = j - 1 + I(1);
 ipiv(j) = jp;
 % Permute just this column
 if j ~= jp
  A([j,jp],j) = A([jp,j],j);
 end % if
 % Apply pending permutations to L
 ntopiv = 1;
 ipivstart = j;
 jpivstart = j - ntopiv;
 while ntopiv < kahead
  for i=ipivstart:j
   A([i,ipiv(i)],jpivstart:jpivstart+ntopiv-1)...
    = A([ipiv(i),i],jpivstart:jpivstart+ntopiv-1); % dlaswp
  end % for i
  ipivstart = ipivstart - ntopiv;
  ntopiv = ntopiv * 2;
  jpivstart = jpivstart - ntopiv;
 end % while
 %  Permute U block to match L
 for i=kstart:j
  A([i,ipiv(i)],j+1:j+kcols) = A([ipiv(i),i],j+1:j+kcols); %
      dlaswp
 end % for i
 % Factor the current column
 if A(j,j) ~= 0 && isnan(A(j,j)) == 0
  if abs(A(j,j)) >= sfmin
   t = 1 / A(j,j);
   A(j+1:m,j) = t * A(j+1:m,j); % dscal
  else
   for i=1:m-j
    A(j+i,j) = A(j+i,j) / A(j,j);
   end % for i
  end % if abs
 elseif A(j,j) == 0 && info == 0
  info = j;
 end % if A(j,j)
 % Solve for U block
 LA = tril(A(kstart:kstart+kahead-1,kstart:kstart+kahead-1),-1)
     ...
  + eye(kahead,kahead);
```

```
 X = LA $\backslash$ A(kstart:kstart+kahead-1,j+1:j+kcols); %
     dtrsm
 A(kstart:kstart+kahead-1,j+1:j+kcols) = X;
 % Schur complement
 jk = min(j+kahead,n);
 A(j+1:m,j+1:jk) = A(j+1:m,j+1:jk)...
  - A(j+1:m,kstart:kstart+kahead-1) * A(kstart:kstart+kahead-1,
      j+1:jk); % dgemm
end % for j
% Handle pivot permutations on the way out of the recursion
npived = bitand(int64(nstep),int64(-nstep));
j = nstep - npived;
while j > 0
 ntopiv = bitand(int64(j),int64(-j));
 for i=j+1:nstep
  A([i,ipiv(i)],j-ntopiv+1:j) = A([ipiv(i),i],j-ntopiv+1:j); %
      dlaswp
 end % for i
 j = j - ntopiv;
end % while
% this is not in dgetrf
% Assume m >= n
L = zeros(m,n);
L(1:m,1:n) = tril(A(1:m,1:n),-1) + eye(m,n);
U = triu(A(1:n,1:n));
P = eye(m,m);
for i=1:m
 P([i,ipiv(i)],:) = P([ipiv(i),i],:);
end % for i
```

Let us consider a random matrix of order 300. Figure 2.11 shows the computing times as functions of the block (panel) size nb for different Matlab versions of the LU factorization with partial pivoting: the unblocked algorithm dgetf2, the blocked Crout version dgetfr_c, the blocked left-looking version dgetfr_lk, the recursive algorithm dgetfr_r and a version similar to the LAPACK standard dgetrf. Note that the computing times are subject to variations and that the respective merits of the algorithms may not be the same when coded in Fortran or C when using BLAS3. Moreover, Matlab's lu, which is a built-in function, is much faster than these implementations using m-files, in fact about 10 times faster on the old PC used for these experiments. Figure 2.12 shows the norms $\|PA - LU\|$ as functions of the block size. There is not much difference in the norms.

For the experiments in figures 2.13-2.14 we use a random matrix of order 1000 that we denote by A_{1000} and we compute the LU factorizations of the principal submatrices $A_k = A(1:k, 1:k)$. Figure 2.13 shows that the blocked versions (with a block size nb equal 40) and the recursive one are asymptotically faster than the unblocked algorithm. The norms in Figure 2.14 are more or less the same even though the Crout version is asymptotically the best.

Other packages for dense matrices were developed in the 1990s and later, but they were not as successful as LAPACK. Over the years, other data distributions were proposed. For instance, since a given data distribution may not be efficient for all the phases of an LU factorization. G. Ballard, J.W. Demmel, B. Lipshitz, O. Schwartz, and S. Toledo [92] introduced a shape morphing procedure that dynamically matches the layout to the computation throughout the algo-

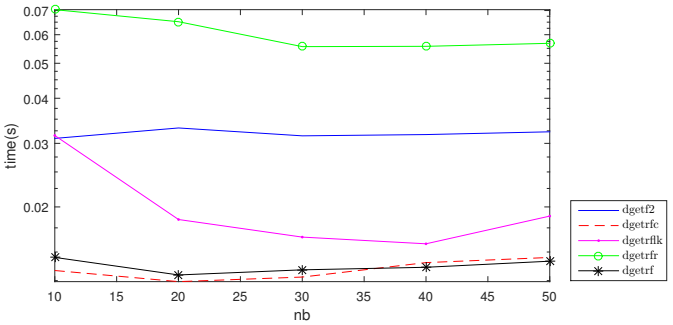


Figure 2.11. *Computing times for the LU factorization as a function of* nb, $n = 300$

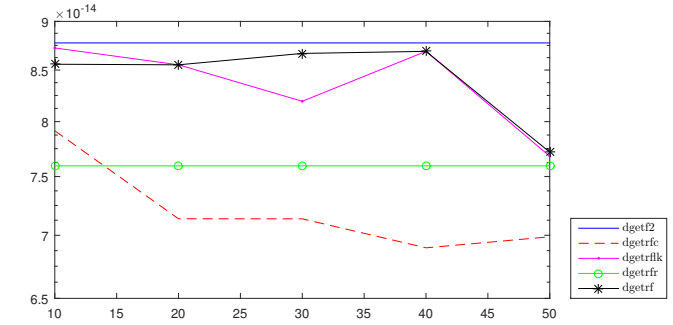


Figure 2.12. $\|PA - LU\|$ *as a function of* nb, $n = 300$

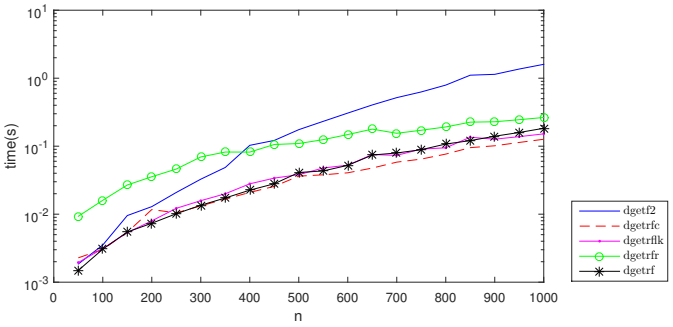


Figure 2.13. *Computing times for the LU factorization as a function of* n, nb=*40,* A_{1000}

rithm, switching the data layout of parts of the matrix back and forth between column-major layout and recursive block-contiguous layout.

2.12 ▪ Parallel solution of general linear systems

Many algorithms have been developed for solving linear systems on parallel computers since they became commercially available in the 1970s. Devising parallel versions of Gaussian elimination is not so easy since, at first sight, elimination is a sequential process. The main problems are

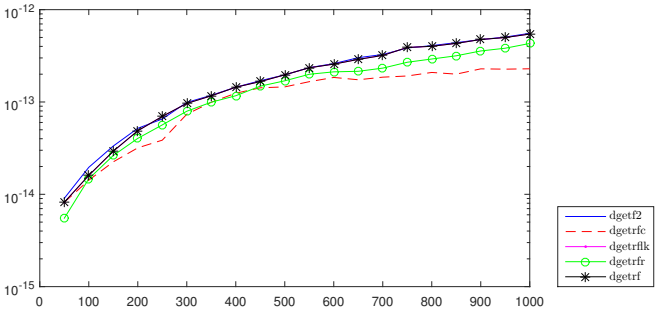


Figure 2.14. $\|PA - LU\|$ *as a function of* n, nb=*40,* A_{1000}

the pivoting strategy and how to distribute the matrix for distributed memory computers. In the following, we review some approaches to these problems for dense matrices and show the progresses that had been made over the years. We start by considering triangular systems on distributed memory architectures.

2.12.1 ▪ Triangular systems

Although this accounts only for a small fraction of the overall computing time when solving general linear systems on sequential computers, solving triangular systems might be a bottleneck on parallel computers if not implemented efficiently. It is an interesting challenge for parallel computing. It might seem a very sequential task since the unknowns must be computed in sequence, one after the other. Nevertheless, many parallel algorithms have been devised for this problem, see the survey by D. Heller [622] in 1978 and, for instance, the early papers by S.C. Chen and D.J. Kuck [245], A.H. Sameh and R.P. Brent [987], S.C. Chen, D.J. Kuck, and A.H. Sameh [246]. Several of the oldest algorithms assumed that $O(n^3)$ processors were available for a system of order n and they were not of practical interest at that time since the number of processors was quite limited, see D. Heller [621], or some of these algorithms were unstable, see L. Csanky [293].

We observe that the method described in [987] is quite different from the others. It uses the fact that the inverse of a lower triangular matrix L can be written as $L^{-1} = L_n^{-1} L_{n-1}^{-1} \cdots L_1^{-1}$, where

$$L_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & \frac{1}{\ell_{k,k}} & & \\ & & -\frac{\ell_{k+1,k}}{\ell_{k,k}} & 1 & \\ & & \vdots & & \ddots \\ & & -\frac{\ell_{n,k}}{\ell_{k,k}} & & 1 \end{pmatrix}.$$

Assuming n is even, the product for L^{-1} is started as $n/2$ products that can be computed in parallel,

$$L^{-1} = (L_n^{-1} L_{n-1}^{-1})\,(L_{n-2}^{-1} L_{n-3}^{-1}) \cdots (L_2^{-1} L_1^{-1}).$$

This is repeated in the next stages until there is only one term which multiplied by the right-hand side gives the solution. Of course, the degree of parallelism is decreasing with the number of

steps. There was no numerical experiments on a parallel computer in [987]. This algorithm is also described in the more recent book [486] by E. Gallopoulos, B. Philippe, and A.H. Sameh.

Many papers were published in the 1980s about the implementation of LU factorization and triangular solves on hypercube parallel computers that were quite popular at that time. Let us cite R.M. Chamberlain [216, 217], G.A. Geist and M.T. Heath [500], C.B. Moler [834], C.H. Romine and J.M. Ortega 1986 [950], G. Li and T.F. Coleman [736, 737]. These implementations distributed the rows or the columns of the triangular matrix in the local memories of the processors. Some of them assumed a particular (virtual) topology of the communication network. Let us briefly explain what was proposed by M.T. Heath and C.H. Romine [619] and S.C. Eisenstat, M.T. Heath, C.S. Henkel, and C.H. Romine [405] at the end of the 1980s.

To distribute the rows or columns, we define a mapping `map` giving the processor number to which a row or column is mapped. The most commonly used one at these times is known as the *wrap mapping*. Assuming that n is a multiple of the number of processors, it is defined as,

$$\begin{pmatrix} \texttt{j}: & 1 & 2 & 3 & \dots & p & p+1 & \dots & 2p & 2p+1 & \dots & n \\ \texttt{map(j)}: & 1 & 2 & 3 & \dots & p & 1 & \dots & p & 1 & \dots & p \end{pmatrix}$$

where p is the number of processors. This mapping can be generalized by considering blocks of consecutive rows or columns instead of individual rows or columns, as it was suggested by V.Y. Pan [901]. Doing this decreases the communication time but may increase load imbalance. Methods using these block mappings are called panel methods in E. Rothberg's Ph.D. thesis [957].

People used the two variants of the triangular solve. Let us first consider the axpy algorithm for a lower triangular solve on a distributed memory computer. It is obvious that the modifications of the components of the right-hand side b due to one column can be computed in parallel if the matrix is distributed by rows. Assume that each processor has a set `{myrows}` containing the indices of the rows (as well as right-hand side and solution components) the memory of the processor is storing. As soon as the x_j component of the solution is computed by processor number j, it must be broadcasted to all the other processors. This is called a *fan-out operation*. The notation `Fan-out(x,proc)` means that processor `proc` sends `x` (located in its local memory) to all the other processors. Of course, sending just one word of data is not really efficient because of the latency of the send operation. Sending a message of ℓ words costs $t = t_0 + \tau\ell$, t_0 is the start-up time (or latency). The efficiency depends on the value of t_0 relatively to τ and ℓ. For sending only one word, the cost is essentially the start-up time. The algorithm (the code running on one processor) looks like the following (see [619]),

```
for j = 1:n
  if j ∈ { myrows }
    x(j) = b(j) / L(j,j);
    fan-out(x(j),map(j))
  else
    wait for x(j)
  endif
  for i ∈ { myrows }
    b(i) = b(i) - L(i,j) * x(j);
  end
end
```

The implementation of the fan-out operation depends on the computer architecture, particularly the topology of the communication network. It provides the necessary synchronization since one processor sends data and all the others wait to receive it. This algorithm can be efficient only on computers with a small latency. Otherwise, much of the time is spent in communications.

To obtain some parallelism in the inner loop for the dot product algorithm, the data has to be distributed by columns or block of columns. Then, each processor can compute the `L(i,j)*x(j)` term for `j` in its column index set { `mycolumns` }. These partial contributions must be added to those of the other processors to obtain the dot product. This is done in a *fan-in operation*: `Fan-in(x,proc)` means that processor `proc` receives the sum of all the `x`'s over all processors. Other algorithms were devised that look for parallelism in the outer loop, such as wavefront algorithms or cyclic algorithms; see M.T. Heath and C.H. Romine [619], G. Li and T.F. Coleman [736, 737]) who assumed an embedding on a ring topology with a wrap mapping of the matrix columns. They solved systems of order up to $n = 2000$.

Some people tried other data distributions. L.D.J.C. Loyens and R.H. Bisseling [766], and R.H. Bisseling and J.G.G. van de Vorst [127], used a square grid cyclic distribution in which $\ell_{i,j}$ is stored in processor (i mod q, j mod q) when $p = q \times q$. Their algorithm is dot product-oriented with neighbour to neighbour communication. It was programmed in Occam 2 on a network of transputers with at most 36 processors, for solving systems up to $n = 1200$ in single precision.

In the 1970s and 1980s, many people used models of the computation to try to estimate the volume of communications and the speed up one can expect from a parallel algorithm. E.E. Santos [988] used a so-called LogP model to obtain lower bounds for computation times and data layout in parallel trinagular solvers for several data distributions. This study shows that block data layout and block-cyclic layouts can incur higher running times than those of many other common data layouts, such as row-column wrapped. Unfortunately, no comparisons of these results with real computations were done.

Computer architectures and performances changed over the years as we have seen in Chapter 1. In the beginning of the 1990s, a de facto standard for message passing, the *Message Passing Interface* (MPI), was defined and adopted by many computer companies. It facilitated the portability of the parallel programs and an abstraction from the network topology.

The package ScaLAPACK was intended to be a parallel version of a subset of the LAPACK library. The development started at the beginning of the 1990s and it was first released in 1995 [253]. It was mainly written in Fortran at the University of Tennessee and used a block cyclic data distribution for dense matrices as well as block-partitioned algorithms. This data distribution means that $map(L_{i,j})$=$((i-1) \bmod p, (j-1) \bmod q)$ on a $p \times q$ processor grid. A ScaLAPACK users' guide [132] was published in 1997. Extensions were developed later [317].

The benefits of blocked and recursive algorithms, as well as appropriate data distributions, were discussed by E. Elmroth, F.G. Gustavson, I. Jonsson, and B. Kågström [421]. P. Stpiczyński [1049] described a new data distribution for block triangular matrices and made numerical experiments on a parallel computer up to $n = 16,000$.

In 1965, G.E. Moore, who was the co-founder of Intel, stated that the number of transistors in integrated circuits doubles about every year. This was later known as *Moore's law*, even though it was not a law of physics, but just based on observations. In the mid-1970s, Moore revised his "law" to doubling every two years. Unfortunately, after some time, in the beginning of the 2000s the rate of increase started to slow down. It became difficult to increase again the frequency and the microprocessor companies started to put several processing units on one chip. This induced a change of terminology; these processing units that would have been called processors before are now known as *cores*. A processor has several or many cores, becoming a parallel computer with, eventually, a shared memory. There are multicore chips with tens of cores that are used in supercomputers.

To further increase the computational speed, computer manufacturers relied on special purpose computing engines, known as GPUs, that were initially designed for computations related to graphic devices. GPUs have their own memory and have a larger degree of parallelism than multicore chips. Today's supercomputers mainly owe their computing power from GPUs. Therefore,

there were many attempts to devise efficient implementations of dense numerical linear algebra on these devices. We can cite the MAGMA and PLASMA (Parallel Linear Algebra Software for Multicore Architectures) libraries developed at the University of Tennessee.

Other works specifically related to triangular systems are [1116] by T. Wicky, E. Solomonik, and T. Hoefler, considering systems with multiple right-hand sides and using inversion of diagonal blocks, [236, 235] by A. Charara, D.E. Keyes, and H. Ltaief with the implementation on GPUs of a recursive block method for multiple right-hand sides.

More recently, another library, named SLATE (Software for Linear Algebra Targeting Exascale), was in development at the University of Tennessee [496]. It adopts a new data distribution since a matrix is a "loose" collection of *tiles* (that is, submatrices) without any constraints on the memory location of any tile with respect to the others. The distribution of tiles can be arbitrary, but the default is a 2D cyclic distribution. It means that, for a triangular matrix, it is not necessary to store the whole matrix. The goal of SLATE is to replace ScaLAPACK with a better performance potential and maximum scalability on modern, many-node high-performance computers with large numbers of cores and multiple hardware accelerators per node.

2.12.2 ▪ LU factorization

A main issue when implementing LU factorization on distributed memory computers is the distribution of the matrix in the local memories of the processing units. As we have seen above, many implementations in the 1980s and 1980s used distributions in which one column or one row is assigned to one processor with a wrap mapping. Another important issue is pivoting. Most of the time, partial pivoting was used. Finding the pivot at one step is easy if the matrix is distributed by columns. But, then, the pivot information has to be broadcast to the other processors. A straightforward implementation looks like the following.

```
for k = 1:n-1
  if k ∈ { mycolumns }
    find pivot index r
    for i=k+1:n
      m(i,k) = a(i,k) / a(k,k)
    end
    broadcast m and pivot index
  else
    receive m and pivot index
  end
  for j> k & j ∈ { mycolumns }
    for i = k+1:n
      a(i,j)=a(i,j) - m(i,k) * a(k,j)
    end
  end
end
```

Many of the implementation in the 1980s and early 1990s were devised for hypercube computers without too many processors. G.A. Geist [499] used a scheduling of tasks such that the pivot search is completely masked. Other implementations on hypercubes were [216, 217] by R.M. Chamberlain, [261] by E. Chu and A. George who used a simple load-balancing scheme to maintain computational balance in the presence of pivoting with an explicit exchange of rows, [501] by G.A. Geist and C.H. Romine where pipelining is used to mask the cost of pivoting, [339] by F. Desprez, B. Tourancheau, and J.J. Dongarra who used a column distribution and pipelining, [37] by M. Angelaccio and M. Clajanni with a subcube matrix decomposition.

More efficient algorithms can be obtained if the data is partitioned by blocks, see J.J. Dongarra and D.W. Walker [353]. Independent data distributions are used for rows and columns. An object m (a piece of row or column) is mapped to a couple (p, i), p being the processor number and i the location in the local memory of that processor. By using wrapping, we have

$$m \longrightarrow (m \bmod p, \lfloor m/p \rfloor).$$

Blocking consists of assigning contiguous entries to processors by blocks,

$$m \longrightarrow (\lfloor m/L \rfloor, m \bmod L), \quad L = \lceil m/p \rceil.$$

The *block cyclic distribution* is a combination of both. Blocks of consecutive data are distributed by wrapping,

$$m \longrightarrow (q, b, i),$$

where q is the processor number, b the block number in processor q and i the index in block b. If there are r data objects in a block, then

$$m \longrightarrow \left(\left\lfloor \frac{m \bmod T}{r} \right\rfloor, \left\lfloor \frac{m}{T} \right\rfloor, m \bmod r \right), \quad T = rp.$$

To distribute the matrix, independent block cyclic distributions are applied for the rows and columns. The processors are supposed to be (logically) arranged in a two dimensional mesh and referred by couples (q_1, q_2). Communications are required for the pivot search and the computation of the multipliers. The communications to be done are a broadcast to all processors and a broadcast to all processors in the same row (or column) in the 2D mesh of processors. For details, see [353], and also E. Rothberg [957]. A taxonomy of distributed dense LU factorization methods was proposed by C.C. Ashcraft [48, 49].

At the end of the 1980s, experiments were done on vector multiprocessors; see, for instance, T.C. Oppe and D.R. Kincaid [882] who tried several variants of LU factorization on a CRAY X-MP/4 and M.J. Daydé and I.S.Duff [316] who studied the impact of Level 3 BLAS in LU factorization on the Cray 2, the ETA-10P, and the IBM 3090/VF.

As we said above, ScaLAPACK is a parallel version of a subset of the LAPACK library, developed in the mid-1990s; see J. Choi, J.J. Dongarra, S. Ostrouchov, A. Petitet D. Walker, and R.C. Whaley [252] for the design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. The DGETRF routine in LAPACK implements a blocked LU factorization. The algorithm iterates over block columns (panels). At each step, the LU factorization with partial pivoting of the current panel is computed, a block row of U is determined, and the trailing matrix is updated. The PDGETRF routine in ScaLAPACK distributes the input matrix over processors using a block cyclic layout. With this partition, every column is distributed over several processors. Finding the maximum element in a column of L for partial pivoting incurs one reduction operation among processors.

In the 2000s, supercomputers were evolving towards machines with multi-core processors and GPU accelerators. More and more, communication was the bottleneck of LU factorization and other implementations were devised to adapt the algorithms to the new architectures. New libraries were made available. MAGMA (Matrix Algebra on GPU and Multi-core Architectures) LU factorization does a CPU factorization of the panel, with look-ahead, update of another panel on GPU which is sent back to CPU for factorization while the GPU updates the rest of the panels, see S. Tomov, J.J. Dongarra, and M. Baboulin [1067].

Reducing the communication time became the key factor to obtain good performances. At the beginning of the 2000s, D. Irony and S. Toledo [667] used a 3-dimensional data distribution

in which every processor is responsible for a 3D subcube. Their approach led to less communications but more temporary storage.

In 2008-2010, J.W. Demmel, L. Grigori, and H. Xiang [574, 335, 575] introduced CALU, a *communication avoiding* Gaussian elimination. Communication avoiding refers to algorithms that reduce the communication to a minimum, while being numerically as stable as classic algorithms, and without a significant increase of the number of floating-point operations. The main novelty of their approach was the way the pivots are found when factorizing a panel. Their method is called *tournament pivoting*. Let us explain this strategy for a first panel of dimension $n \times n_b$. It has the property that the communication for computing the panel factorization does not depend on the number of columns, but only on the number of processors in the parallel case. A preprocessing step aims at finding, at low communication cost, n_b rows that can be used as pivots to factor the entire panel. Let us assume that we have $p = 4$ processing units and that n is a multiple of 4. The $n \times n_b$ panel is partitioned into four blocks of size $n/4 \times n_b$ which are factored independently using a standard Gaussian elimination with partial pivoting (GEPP) as

$$A_{:,1:n_b} = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{pmatrix} = \begin{pmatrix} P_{1,1}L_{1,1}U_{1,1} \\ P_{2,1}L_{2,1}U_{2,1} \\ P_{3,1}L_{3,1}U_{3,1} \\ P_{4,1}L_{4,1}U_{4,1} \end{pmatrix} = P_1 L_1 U_1,$$

where P_1 and L_1 are block diagonal matrices. Then, we have four sets of n_b potential pivot rows. The global pivot rows are obtained from these four sets by performing a binary tree of GEPP factorizations of matrices of size $2n_b \times n_b$, grouping the blocks two by two,

$$\begin{pmatrix} (P_1^T A)_{1:n_b,1:n_b} \\ (P_1^T A)_{n_4+1:n_4+n_b,1:n_b} \\ (P_1^T A)_{2n_4+1:2n_4+n_b,1:n_b} \\ (P_1^T A)_{3n_4+1:3n_4+n_b,1:n_b} \end{pmatrix} = \begin{pmatrix} P_{1,2}L_{1,2}U_{1,2} \\ P_{2,2}L_{2,2}U_{2,2} \end{pmatrix} = P_2 L_2 U_2,$$

where $n_4 = n/4$. We extend P_2 by identity matrices to the dimensions of P_1 and we proceed to the last step,

$$\begin{pmatrix} (P_2^T P_1^T A)_{1:n_b,1:n_b} \\ (P_2^T P_1^T A)_{2n_4+1:2n_4+n_b,1:n_b} \end{pmatrix} = P_{1,3}L_{1,3}U_{1,3} = P_3 L_3 U_3.$$

This provides us with the n_b global pivots. Then, these n_b rows are permuted into the first positions of the panel and the LU factorization without pivoting of the entire panel is performed. In general, there are around $\log_2 p$ stages. Other reduction trees than a binary tree can be used. The found pivots may be different from those of GEPP. Tournament pivoting reduces to partial pivoting when $n_b = 1$. A stability analysis was done in [575], but the upper bound of the growth factor is worse than with GEPP. Note also that the degree of parallelism is decreasing at each stage of the reduction tree. Communication lower bounds for QR and LU factorization are given in [334].

For the experiments in Figure 2.15 we used the same random matrix A_{1000} of order 1000 as above and we computed the LU factorizations of the principal submatrices $A_k = A(1:k, 1:k)$. It shows $\|PA - LU\|$ for several values of p for tournament pivoting, as well as the norms for unblocked and blocked GEPP. We see that the norms are larger with tournament pivoting. The computing times can be smaller on a parallel computer, but there is no reason to use tournament pivoting on a sequential computer since most of the time is spent in the factorization of parts of the panels, just to find the pivots.

E. Solomonik and J.W. Demmel [1029] introduced what they called 2.5D matrix multiplication and LU factorization algorithms with the use of tournament pivoting. With c copies of

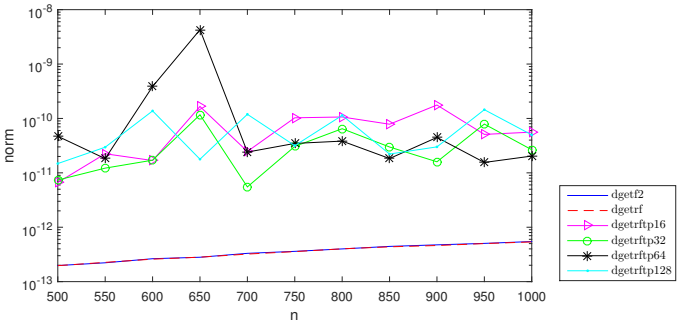


Figure 2.15. $\|PA - LU\|$ *as a function of* n, nb=*40, tournament pivoting and GEPP*

the input data, this algorithm reduces the number of words moved by a factor of $\sqrt{c}$, but it send $\sqrt{c}$ more messages. They showed that using extra memory cannot reduce the latency cost of LU below the 2D standard algorithm.

Starting from 2010, many papers appeared describing implementations of LU factorization on computers using GPU accelerators. In [1110] F. Wang, C.Q. Yang, Y.F. Du, J. Chen, H.Z. Yi, and W.X. Xu reported on the implementation of the High Performance Linpack benchmark on the TianHe-1 Chinese supercomputer which obtained a 0.563 Pflops performance in 2009.

E.F. D'Azevedo and J.C. Hill [321] proposed a left-looking algorithm for complex matrices based on ScaLAPACK whose aim was to minimize the data transfer between the CPU host and the GPU memory.

M. Baboulin, S. Donfack, J.J. Dongarra, L. Grigori, A. Rémy, and S. Tomov [78] studied two approaches to reduce the communication overhead due to pivoting on hybrid CPU/GPU architectures. The first one is based on tournament pivoting using two levels of blocking. On the CPU, each panel of width n_b is subdivided into several panels of smaller widths which are tuned to optimize the utilization of the multicore cache. The matrix-matrix update of the trailing submatrix is done on the GPUs. A look-ahead technique is used to overlap the computations of the CPU and GPU. As soon as a new panel has been computed in the GPU, it is sent to the CPU. Experiments are reported in [78] with a 48-core CPU and one GPU. The second algorithm used recursive butterfly transformations that we have described in Section 2.2.3 with only two levels of recursion (PBRT). The matrix $U^T AV$, where U and V are two random PBRTs, is factorized without pivoting, followed with iterative refinement in the working precision.

For the experiments in Figure 2.16 we use a random matrix of order $1024 = 2^{10}$ and we compute the LU factorizations of the principal submatrices $A_k = A(1:k, 1:k)$ for $k = 2^j, j = 4:10$. We use two levels of recursion in the construction of the RBTs and we plot the ℓ_2 norm of the error where the exact solution of each linear system is a given random vector. One can see that the error is much larger with RBT, but that two steps of iterative refinement give an error of the same order or better than with GEPP.

Y. Jia, P. Luszczek, and J.J. Dongarra [681] described a multi-core, multi-GPU implementation of recursive LU factorization [588]. A number of CPU cores are dedicated to the panel factorizations. The rest of the CPU cores are used for the update of the trailing submatrix. All the GPUs are used only for the submatrix update. The authors solved problems of order n up to $32,000$ with 4 GPUs and reached 46% of the theoretical computational peak. A parallel fine-grained recursive formulation of the panel factorizations was exploited by J.J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek [349]. It is based on conflict-free partitioning of the data and lockless synchronization mechanisms.

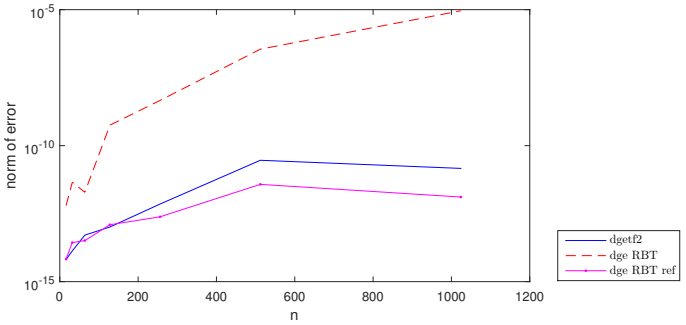


Figure 2.16. *Norm of error as a function of n, RBT, $d = 2$ with and without iterative refinement and GEPP*

In [700], A. Khabou, J.W. Demmel, L. Grigori, and M. Gu revisited the CALU algorithm. In the new algorithm, a panel is factored by computing the strong rank revealing QR (RRQR) factorization of its transpose. The permutation which is obtained is applied to the rows of the matrix A, and the block L factor of the panel is computed based on the R factor of the strong RRQR factorization. Doing this, they obtained better bounds of the growth factor than with CALU.

An interesting review of the evolution of computers and software for solving dense linear systems was written by P. Luszczek, J. Kurzak, and J.J. Dongarra [769] in 2014. It shows than in order to adapt to the new computer architectures, the codes are getting more and more intricate. That same year, J.J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek described a recursive tile LU factorization with partial pivoting [350]. Their idea was to arrange the original dense matrix into small square regions of data (tiles) that are contiguous in memory to fit into the core's caches in multicore architectures. M. Faverge, J. Herrmann, J. Langou, B. Lowery, Y. Robert, and J.J. Dongarra [444] mixed LU and QR factorization algorithms, dynamically alternating LU with local pivoting and QR elimination steps, based upon some robustness criterion. The hybrid algorithm executes a QR step when a robustness criterion detects some risk for instability, and they execute an LU step otherwise. Porting the PLASMA library to the OpenMP standard for shared memory architectures was described in [1138].

More recently, the SLATE library stores the matrix as a collection of individual tiles. This is a flexible data distribution since it can be used for holding many different matrix types and tiles can easily be moved or copied among different memory spaces. Tile indexing is global, and each computer node stores only its local subset of tiles. By default, mapping of tiles is a 2D block cyclic mapping. For details, see J. Kurzak, M. Gates, A. Charara, A. YarKhan, I. Yamazaki, and J.J. Dongarra [716]. Tournament pivoting in SLATE is described in R. Alomairy, M. Gates, S. Cayrols, D. Sukkari, K. Akbudak, A. YarKhan, P. Bagwell, and J.J. Dongarra [14]. They showed computational speeds for matrices of order up to $n = 307,200$.

N. Lindquist, M. Gates, P. Luszczek, and J.J. Dongarra [749] used threshold pivoting. It means that the pivot is selected such that $|a_{i,i}| \geq \tau |a_{j,i}|$, $j = i, \ldots, n$ with $0 \leq \tau \leq 1$. In step k, this is used for all processors except the one holding $a_{k,k}$. This method reduces the inter-process exchanges and data movement.

As we have seen in Chapter 1, Section 1.21, a new trend in numerical linear algebra is the use of mixed-precision computations. GPU's computations are much faster in single, or even half-precision. Therefore, LU factorization did not escape from this trend.

The mixed-precision iterative refinement algorithm described by A. Haidar, H. Bayraktar,

S. Tomov, J.J. Dongarra, and N.J. Higham [609] computes an LU factorization in low precision, uses the LU factors to compute an initial approximation x_0, and then computes an iterative refinement process in double precision (fp64) arithmetic. This work is based on the three-precision algorithm of E.C. Carson and N.J. Higham [205], and computes using tensor cores on GPUs. Tensor cores are a special type of unit that computes $D = C + AB$, where all matrices are 4×4, A and B must be stored in fp16, and C and D can be in fp16 or fp32. The LU factorization itself is done using two precisions, the panel factorizations and the triangular solves are done in fp32 and the trailing matrix updates are done in fp16 on the tensor cores but with an output in fp32. The GMRES iterative algorithm (see Chapter 6) preconditioned by the low-precision LU factors is used for the iterative refinement or as an iterative method by itself to solve $Ax = b$. On this type of algorithms, see also [610].

In [765], F. Lopez and T. Mary stored the matrix in fp16. In their left-looking factorization without pivoting, using fp16 LU factors with fp32 buffers, the fp32 buffers are used to accumulate the output of fp16 multiplications in fp32. They also studied variants with panel factorization entirely in fp32 arithmetic or with a doubly partitioned factorization algorithm that exploits GPU tensor cores in the panel factorization.

The fastest LU factorization that has been registered so far (end of 2022) was obtained with the HP Linpack benchmark on the Oak Ridge National Laboratory Frontier supercomputer, see [351]. This machine has 9408 nodes, each node having one AMD EPYC 64-core CPU and four AMD Instinct MI250X GPU accelerators with 220 cores each. The benchmark was run on May 27, 2022 with a dense matrix of order $n = 24,440,832$. The measured speed was 1.102 Eflops which is 66% of the peak. It took $8,828.74$ seconds (2.45 hours) to complete the solve.

2.13 ▪ The Gauss-Jordan algorithm

This method was introduced independently by W. Jordan [687] and B.I. Clasen [268] in 1888. For a (brief) history of Gauss-Jordan elimination, see S.C. Althoen and R. McLaughlin [17] or [160]. In Gaussian elimination, at a given step, we eliminate the entries below the diagonal in a column of A to transform the matrix into an upper triangular form. Gauss-Jordan also eliminates the entries above the diagonal to reduce the matrix to a diagonal form. Let us illustrate that with a small example from [242],

$$A = \begin{pmatrix} 5 & 1 & 2 & 1 \\ 2 & 10 & 3 & 1 \\ 1 & 4 & 8 & 2 \\ 6 & 2 & 4 & 20 \end{pmatrix}, \quad b = \begin{pmatrix} 17 \\ 35 \\ 41 \\ 102 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}. \tag{2.11}$$

Note that the matrix A is strictly diagonally dominant and therefore nonsingular. Moreover, we do not need pivoting. In the first step, which is similar to what is done in Gaussian elimination, we multiply A by the elementary matrix J_1^{-1},

$$J_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -0.4 & 1 & 0 & 0 \\ -0.2 & 0 & 1 & 0 \\ -1.2 & 0 & 0 & 1 \end{pmatrix} \rightarrow J_1^{-1}A = \begin{pmatrix} 5 & 1 & 2 & 1 \\ 0 & 9.6 & 2.2 & 0.6 \\ 0 & 3.8 & 7.6 & 1.8 \\ 0 & 0.8 & 1.6 & 18.8 \end{pmatrix}.$$

In the second step we eliminate in the second column,

$$J_2^{-1} = \begin{pmatrix} 1 & -0.1042 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -0.3958 & 1 & 0 \\ 0 & -0.08333 & 0 & 1 \end{pmatrix} \rightarrow J_2^{-1}J_1^{-1}A = \begin{pmatrix} 5 & 0 & 1.771 & 0.9375 \\ 0 & 9.6 & 2.2 & 0.6 \\ 0 & 0 & 6.729 & 1.563 \\ 0 & 0 & 1.417 & 18.75 \end{pmatrix}.$$

Eliminating in the third column we obtain,

$$J_3^{-1} = \begin{pmatrix} 1 & 0 & -0.2632 & 0 \\ 0 & 1 & -0.3269 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -0.2105 & 1 \end{pmatrix} \rightarrow J_3^{-1}J_2^{-1}J_1^{-1}A = \begin{pmatrix} 5 & 0 & 0 & 0.5263 \\ 0 & 9.6 & 0 & 0.08916 \\ 0 & 0 & 6.729 & 1.563 \\ 0 & 0 & 0 & 18.42 \end{pmatrix}.$$

Finally, the last step is

$$J_4^{-1} = \rightarrow \begin{pmatrix} 1 & 0 & 0 & -0.02857 \\ 0 & 1 & 0 & -0.00484 \\ 0 & 0 & 1 & -0.08482 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow J_4^{-1}J_3^{-1}J_2^{-1}J_1^{-1}A = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 9.6 & 0 & 0 \\ 0 & 0 & 6.729 & 0 \\ 0 & 0 & 0 & 18.42 \end{pmatrix}.$$

The last matrix is diagonal. Multiplying the right-hand side by $J_4^{-1}J_3^{-1}J_2^{-1}J_1^{-1}$, which is a dense matrix, and dividing by the diagonal entries of the last matrix, we obtain the solution of the linear system. The inverse of A is also easily obtained without any triangular solve.

Let us now consider the general case. At step k we multiply the current matrix A_k by

$$J_k^{-1} = \begin{pmatrix} 1 & & & -\frac{a_{1,k}^{(k)}}{a_{k,k}^{(k)}} & & \\ & \ddots & & \vdots & & \\ & & 1 & -\frac{a_{k-1,k}^{(k)}}{a_{k,k}^{(k)}} & & \\ & & & 1 & & \\ & & & -\frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & \\ & & & \vdots & & \ddots \\ & & & -\frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & 1 \end{pmatrix}.$$

The inverse J_k is obtained by changing $-$ to $+$ in the kth column. To relate this to what is done in Gaussian elimination, we observe that

$$J_k^{-1} = \begin{pmatrix} 1 & & & -\frac{a_{1,k}^{(k)}}{a_{k,k}^{(k)}} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & -\frac{a_{k-1,k}^{(k)}}{a_{k,k}^{(k)}} & & & \\ & & & 1 & & & \\ & & & 0 & 1 & & \\ & & & \vdots & & \ddots & \\ & & & 0 & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & 0 & & & \\ & \ddots & & \vdots & & & \\ & & 1 & 0 & & & \\ & & & 1 & & & \\ & & & -\frac{a_{k+1,k}^{(k)}}{a_{k,k}^{(k)}} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & -\frac{a_{n,k}^{(k)}}{a_{k,k}^{(k)}} & & & 1 \end{pmatrix}.$$

The matrix on the right is the matrix L_k^{-1} in the standard Gaussian elimination. Let U_k^{-1} be the other matrix in the product such that $J_k^{-1} = U_k^{-1}L_k^{-1}$. These two matrices commute. Moreover, as observed in [242], $U_j^{-1}L_i^{-1} = L_i^{-1}U_j^{-1}$. Since

$$U_{n-1}^{-1}L_{n-1}^{-1}\cdots U_1^{-1}L_1^{-1}A = D,$$

with a diagonal D, we obtain

$$U_{n-1}^{-1}\cdots U_1^{-1}L_{n-1}^{-1}\cdots L_1^{-1}A = D.$$

Hence, $A = LU$ with

$$L = L_1 \cdots L_{n-1}, \quad U = U_1 \cdots U_{n-1} D.$$

Of course, we can also introduce pivoting in the Gauss-Jordan algorithm. The following code solves $Ax = b$ using the Gauss-Jordan algorithm with row pivoting (pivot search in column and exchange of rows). Of course, we do not compute the matrices J_k^{-1}, but just the multipliers which are needed to update the right part of the matrix and the right-hand side.

```
function [x, info] = dgjsv(A,b);
[m,n] = size(A);
info = 0;
for j=1:min(m,n)
 % Find pivot and test for singularity
 [~,I] = max(abs(A(j:m,j)));
 jp = j - 1 + I(1); % global index of the pivot
 if A(jp,j) ~= 0
  % Apply the interchange to columns j:n
  if jp ~= j
   A([j,jp],j:n) = A([jp,j],j:n);
   b([j,jp]) = b([jp,j]);
  end % if
  % Compute elements 1:m of j-th column
  rowind = [1:j-1, j+1:m];
  mult = A(rowind,j) / A(j,j);
 elseif info == 0
  info = j;
 end % if jp
 % Update trailing submatrix
 jm = rowind;
 jn = j:n;
 u = -mult;
 v = A(j,j:n);
 A(jm,jn) = prank1(A(jm,jn),u,v); % rank-one update of A(jm,jn)
 b(jm) = b(jm) - mult * b(j);
end % for j
x = b ./ diag(A);
```

The stability of the Gauss-Jordan algorithm with partial pivoting was studied by G. Peters and J.H. Wilkinson [915] in the 1970s. They stated that this algorithm "is commonly regarded as suspect. It is shown that in many respects suspicions are unfounded, and in general the absolute error in the solution is strictly comparable with that corresponding to Gaussian elimination with partial pivoting plus back substitution. However, when A is ill-conditioned, the residual corresponding to the Gauss-Jordan solution will often be much greater than that corresponding to the Gaussian elimination solution". People also observe that a drawback of Gauss-Jordan is that the operation count for solving $Ax = b$ is $n^3 + 2n - 3$, that is, 50% more than for Gaussian elimination.

A more recent rounding error analysis was done by N.J. Higham [633]. He considered Gauss-Jordan elimination as a two-phase algorithm, one to compute an upper triangular matrix U and a second one to reduce U to a diagonal matrix. The final result of the two phases is that

$$|x_{ex} - x| \leq (n-1)\gamma_3(1+\gamma_3)^{n-2}|X|\,(|U|\,|x| + |y|), \quad X = U^{-1} + O(u),$$

where x is the Gauss-Jordan solution, y is the result of the first phase, x_{ex} is the exact solution, and u is the unit roundoff. Finally, using an error analysis for the first phase, it holds

$$|b - Ax| \leq 9nu|L|\,|U|\,|U^{-1}|\,|U|\,|x| + O(u^2),$$
$$|x_{ex} - x| \leq 2nu(|A^{-1}|\,|L|\,|U| + 3|U^{-1}|\,|U|)\,|x| + O(u^2).$$

One can see that U^{-1} is involved in these bounds. The backward error can be guaranteed to be small only if U is well-conditioned. Gauss-Jordan with partial pivoting is normwise forward stable and, even without pivoting, it is forward stable for symmetric positive definite matrices.

Up to the 1980s, despite what was written by G. Peters and J.H. Wilkinson, Gauss-Jordan had a bad reputation. It was "rehabilitated" by T.J. Dekker and W. Hoffmann [329] in 1989. They noticed that Gauss-Jordan with pivoting by searching the pivot in rows and exchanging columns has good numerical properties and that it is well suited for use on vector computers that were in favor at that time. They also gave a normwise error analysis; see also W. Hoffmann's thesis [641] and [640].

A Gauss-Jordan code with pivot search in rows and column exchanges is the following. Note that we have to do a permutation of the components of the solution at the end because of the column exchanges.

```
function [x,info] = dgjsvr(A,b);
[m,n] = size(A);
col = 1:m;
info = 0;
for j=1:min(m,n)
 % Find pivot in row j and test for singularity
 [~,I] = max(abs(A(j,j:n)));
 jp = j - 1 + I(1);
 if A(j,jp) ~= 0
  % Apply the column interchange
  if jp ~= j
   A(:,[j,jp]) = A(:,[jp,j]);
   col([j,jp]) = col([jp,j]);
  end % if jp
  % Compute elements 1:m of j-th column
  rowind = [1:j-1, j+1:m];
  t = 1 / A(j,j);
  mult = t * A(rowind,j);
 elseif info == 0
  info = j;
 end % if A
 % Update trailing submatrix and rhs
 jm = rowind;
 jn = j:n;
 u = -mult;
 v = A(j,j:n);
 A(jm,jn) = prank1(A(jm,jn),u,v); % rank-one update of A(jm,jn)
 b(jm) = b(jm) - mult * b(j);
end % for j
x = b ./ diag(A);
ip = invperm(col); % inverse permutation
x = x(ip,:);
```

As for what can be done for Gaussian elimination, Y. Li [743] related the entries of the matrix A_k obtained at step k of the Gauss-Jordan algorithm to ratios of determinants. A more direct and simpler proof by induction was given by N. Van Tran, J. Justino, and I. van den Berg [1092].

The Gauss-Jordan algorithm is better suited to parallel computing than Gaussian elimination. This was already noticed in the 1970s and 1980s, see T.D. Kimura [703], and R.G. Melhem [797]. T.J. Dekker, W. Hoffmann, and K. Potma [330] described parallel algorithms for solving large linear systems, including Gauss-Jordan. In the 21st century, we can cite N. Melab, E-G. Talbi, and S. Petiton [796] who proposed a parallel adaptive block Gauss-Jordan algorithm to compute the inverse, but without pivoting, P.D. Michailidis and K.G. Margaritis [831] who described an OpenMP pipelined implementation on a multicore platform, and, more recently, J.-P. David [307] with a low latency and division free Gauss-Jordan solver, as well as H. Anzt, J.J. Dongarra, G. Flegar, and E.S. Quintana-Ortí [39] with a GPU implementation of Gauss-Jordan to compute block Jacobi preconditioners.

A parallel block Gauss-Jordan was studied by O. Tingleff [1061]. It is relatively easy to develop a blocked Gauss-Jordan algorithm where `nb` columns (a panel) are handled together. A prototype code is the following.

```
function [x,info] = dgbjsvc(A,b,nb);
[m,n] = size(A);
nbl = ceil(m / nb); % number of blocks
for k=1:nbl
 % block k
 js = (k - 1) * nb + 1; % start of the block
 je = min(js + nb - 1,m); % end of the block
 % permute the rows, Gauss factorization of the panel
 [~,~,info,L,U,row] = dgetf2(A(js:m,js:je));
 row = row + js - 1; % global index
 A(js:m,:) = A(row,:);
 b(js:m) = b(row);
 % compute the inverse of the permuted block
 nblo = je - js + 1;
 Y = L(1:nblo,1:nblo) $\backslash$ eye(nblo,nblo);
 IA11 = U \ Y;
 A(1:js-1,js:je) = A(1:js-1,js:je) * IA11;
 A(je+1:m,js:je) = A(je+1:m,js:je) * IA11;
 A(1:js-1,je+1:n) = A(1:js-1,je+1:n) - A(1:js-1,js:je) * A(js:
    je,je+1:n);
 A(je+1:m,je+1:n) = A(je+1:m,je+1:n) - A(je+1:m,js:je) * A(js:
    je,je+1:n);
 b(1:js-1,:) = b(1:js-1,:) - A(1:js-1,js:je) * b(js:je,:);
 b(je+1:m,:) = b(je+1:m,:) - A(je+1:m,js:je) * b(js:je,:);
 A(js:je,js:je) = IA11;
end % for k
x = zeros(m,1);
for k=1:nbl
 % block k
 js = (k - 1) * nb + 1; % start of the block
 je = min(js + nb - 1,m); % end of the block
 IA11 = A(js:je,js:je);
 x(js:je) = IA11 * b(js:je);
```

```
end % for k
```

2.14 ▪ The Gauss-Huard algorithm

A less well-known elimination method was introduced by P. Huard [655] in 1979. This method is often called the Gauss-Huard method, even though it was not known to Gauss. The elimination is done alternatively in rows and columns. We eliminate successively entries in a row to the left of the diagonal and in column above the diagonal. Let us illustrate that without pivoting on the small example defined in (2.11). We first scale the first row to have $a_{1,1} = 1$, and eliminate the entry $(2, 1)$ in the second row,

$$\begin{pmatrix} 5 & 1 & 2 & 1 \\ 2 & 10 & 3 & 1 \\ 1 & 4 & 8 & 2 \\ 6 & 2 & 4 & 20 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0.2 & 0.4 & 0.2 \\ 2 & 10 & 3 & 1 \\ 1 & 4 & 8 & 2 \\ 6 & 2 & 4 & 20 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0.2 & 0.4 & 0.2 \\ 0 & 9.6 & 2.2 & 0.6 \\ 1 & 4 & 8 & 2 \\ 6 & 2 & 4 & 20 \end{pmatrix}.$$

We observe that the bottom part of the matrix (that is, the last two rows) is not modified. Then, we scale the second row and eliminate the entry $(1, 2)$, modifying the 2×2 top-right block,

$$\rightarrow \begin{pmatrix} 1 & 0 & 0.3542 & 0.1875 \\ 0 & 1 & 0.2292 & 0.0625 \\ 1 & 4 & 8 & 2 \\ 6 & 2 & 4 & 20 \end{pmatrix},$$

and we continue in the same way, with the third row and third column,

$$\rightarrow \begin{pmatrix} 1 & 0 & 0.3542 & 0.1875 \\ 0 & 1 & 0.2292 & 0.0625 \\ 0 & 0 & 6.729 & 1.563 \\ 6 & 2 & 4 & 20 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0.1053 \\ 0 & 1 & 0 & 0.009288 \\ 0 & 0 & 1 & 0.2322 \\ 6 & 2 & 4 & 20 \end{pmatrix},$$

and we finally obtain an identity matrix after eliminating entries in the last row and last column,

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

Applying the same transformations to the right-hand side, we solve $Ax = b$. The elementary matrices involved in that process are the following. At the first step, E_1 scales the first row, and H_1 eliminates the entry $(2, 1)$,

$$H_1 E_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.2 & 0 & 0 & 0 \\ -0.4 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

At the second step, we have,

$$H_2 E_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -0.02083 & 0 & 0 \\ 0 & 0.1042 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -0.02083 & 0 & 0 \\ 0 & 0.1042 & 0 & 0 \\ -1 & -0.3958 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

and the third step is

$$H_3E_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -6 & -2 & -4 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -0.05263 & 0 \\ 0 & 1 & -0.03406 & 0 \\ 0 & 0 & 0.1486 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -0.05263 & 0 \\ 0 & 1 & -0.03406 & 0 \\ 0 & 0 & 0.1486 & 0 \\ -6 & -2 & -0.2105 & 1 \end{pmatrix}.$$

The last step, to eliminate entries in the last column, is

$$E_4 = \begin{pmatrix} 1 & 0 & 0 & -0.005714 \\ 0 & 1 & 0 & -0.0005042 \\ 0 & 0 & 1 & -0.01261 \\ 0 & 0 & 0 & 0.05429 \end{pmatrix},$$

and $H_4 = I$. The product of all these transformations $H_4E_4H_3E_3H_2E_2H_1E_1$ is a dense matrix (which is, in exact arithmetic, the inverse of A) and we observe that the elementary matrices do not commute.

A Gauss-Huard code with pivot search in rows and column exchanges is the following.

```
function [x,info] = dghsvr(A,b);
[m,n] = size(A);
col = 1:n;
info = 0;
% find first pivot in row 1, permute the columns and divide
[~,I] = max(abs(A(1,1:n)));
jp = I(1);
if jp ~= 1
 A(:,[1,jp]) = A(:,[jp,1]);
 col(1) = col(jp);
 col(jp) = 1;
end % if
t = 1 / A(1,1);
A(1,:) = t * A(1,:);
B(1,:) = t * B(1,:);
for j=2:min(m,n)
 % modify row j and the rhs
 A(j,j:n) = A(j,j:n) - A(j,1:j-1) * A(1:j-1,j:n);
 b(j) = b(j) - A(j,1:j-1) * b(1:j-1,);
 % Find pivot in row j and test for singularity
 [~,I] = max(abs(A(j,j:n)));
 jp = j - 1 + I(1);
 if A(j,jp) ~= 0
  % Apply the interchange
  if jp ~= j
   A(:,[j,jp]) = A(:,[jp,j]);
   col([j,jp]) = col([jp,j]);
  end % if jp
 elseif info == 0
  info = j;
 end % if A
 % scale row j and the rhs
 t = 1 / A(j,j);
```

```
 A(j,j:n) = t * A(j,j:n);
 b(j) = t * b(j);
 % rank-one update of the top right matrix
 m = 1:j-1;
 jn = j+1:n;
 u = -A(1:j-1,j);
 v = A(j,j+1:n);
 A(jm,jn) = prank1(A(jm,jn),u,v);
 b(jm) = b(jm) -  A(jm,j) * b(j);
end % for j
ip = invperm(col); % inverse permutation
x = b(ip);
```

The Gauss-Huard algorithm is described in K. Chen's book [242]. One interest of Huard's method, when compared to Gauss-Jordan, is that it needs the same number of floating-point operations than the standard Gaussian elimination. They both transform the matrix to a diagonal form, but Gauss-Jordan needs 50% more operations.

Gauss-Jordan and Huard's methods were studied by M. Cosnard, Y. Robert, and D. Trystram [286] in 1986. They introduced parallel versions for shared-memory computers; see also [287] in English. Pivoting with column swaps was considered in W. Hoffman's thesis [641] in 1989. Results for a parallel version on the Alliant FX/4 were given. A block Gauss-Huard method for linear solves on parallel computers was constructed by W. Hoffmann, K. Potma, and G. Pronk [643] in 1994.

The stability of the Gauss-Huard algorithm with column swaps was studied by T.J. Dekker, W. Hoffmann, and K. Potma [331], see also [330]. They also gave numerical comparisons of Gauss, Gauss-Jordan, and Gauss-Huard methods on different types of matrices in single-precision arithmetic. Their conclusion was that Gauss-Huard with column swaps appeared to be as stable as Gaussian and Gauss-Jordan eliminations with column swaps.

In [723], P. Laurent-Gengoux and D. Trystram showed that, for symmetric matrices, the Gauss-Huard algorithm without pivoting is equivalent to a conjugate direction method that we will consider in the next section. W. Hoffmann [642] showed that, without pivoting, the Gauss-Huard algorithm computes in each step the next row of a unit diagonal matrix U that satisfies $A = LU$ where L is lower triangular, like it is done in the ikj form of LU factorization. However, in the Gauss-Huard algorithm, the row of U that has been computed is not kept.

An implementation of a block Gauss-Huard algorithm was proposed by P. Benner, P. Ezzatti, E.S. Quintana-Ortí, and A. Remón [109] on computers with multicore processors and GPUs. The impact of panel factorization on the Gauss-Huard algorithm, and the impossibility of overlapping the panel factorization stage with the remaining matrix updates was discussed by S. Catalán, P. Ezzatti, E.S. Quintana-Ortí, and A. Remón [210]. M.A. Hassanein [618] described implementations of different variants of the block Gauss-Huard algorithm with column pivoting on a hybrid CPU-GPU architecture. Numerical results show that these algorithms yield solutions as good as those obtained by Gaussian elimination with partial pivoting and that they are more suitable for parallel architectures

For the numerical experiments of figures 2.17-2.18, we use the random matrix A_{1000} and a random exact solution. We display results for Gaussian elimination (`dgetf2` and `dgetrf`), Gauss-Jordan with row (resp. column) swaps (`dgj`, resp. `dgjcol`), blocked Gauss-Jordan with row swaps (`dgbjrow`), Gauss-Huard with column swaps (`dghcol`), and blocked Gauss-Huard with column swaps (`dgbhcol`). Figure 2.17 shows the residual norms. Those of the block Gauss-Jordan are larger than the other ones. Figure 2.18 displays the error norms. There are of the same magnitude for all methods.

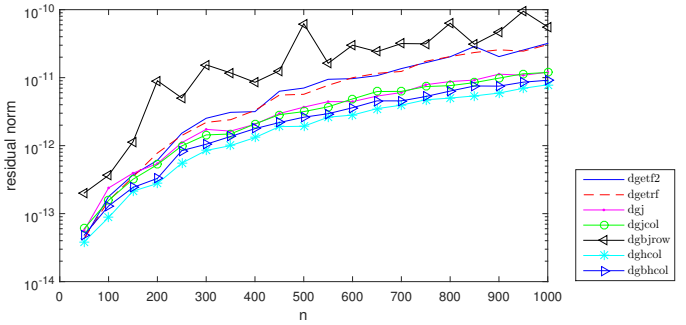


Figure 2.17. *$\|b - Ax\|$ as a function of n, with Gauss, Gauss-Jordan and Gauss-Huard*

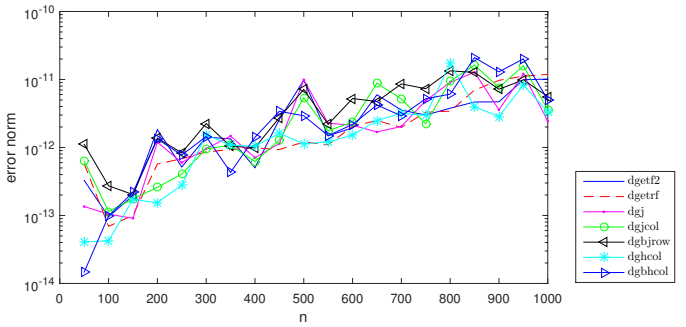


Figure 2.18. *Error norms as a function of n, with Gauss, Gauss-Jordan and Gauss-Huard*

2.15 ▪ The Purcell algorithm

This method was introduced by E.W. Purcell [933] in 1953. His purpose was to solve linear systems on an IBM card programmable calculator whose memory was too small to accommodate the dense matrix entries. The linear system $Ax = b$ with a nonsingular A is written as

$$\hat{A}y = 0 \;\text{ with }\; \hat{A} = (\, A, \quad -b\,), \;\; y = \begin{pmatrix} x \\ 1 \end{pmatrix}.$$

Hence, one looks for a vector y in the null space of the $n \times (n+1)$ matrix $\hat{A}$. One way to compute such a vector is the following. Let $\hat{a}_i$ denotes the ith row of $\hat{A}$ and $V^{(n+1)}$ be a given matrix of order $n+1$ with linearly independent columns $v_i^{(n+1)}$. We do n steps i from n to 1. At step i, we compute new vectors $v_k^{(i)}$, $k = 1, \ldots, i$ as linear combinations of the vectors at step $i+1$,

$$v_k^{(i)} = \alpha_k v_p^{(i+1)} + v_{m_k}^{(i+1)}, \quad k = 1, \ldots, i,$$

where p is a chosen index in $\{1, 2, \ldots, i+1\}$ and m_k is the kth index in that set when p has been removed. The coefficient α_k is chosen such that $\hat{a}_{n+1-i} v_k^{(i)} = 0$, that is,

$$\alpha_k = -\frac{\hat{a}_{n+1-i} v_{m_k}^{(i+1)}}{\hat{a}_{n+1-i} v_p^{(i+1)}}.$$

The integer p can be seen as a pivot index. It is chosen as the index giving the maximum of $|\hat{a}_{n+1-i}v_j^{(i+1)}|$ for $j = 1, \ldots, i+1$. In the original method, division by zero or tiny values was possible. This was considered later by J.P. Roth and D.S Scott [956] in 1956 and C.-H. Lai [720] in 1997.

With this process, at the first step ($i = n$), we have $\hat{a}_1 v_k^{(n)} = 0$, $k = 1, \ldots, n$. For the second step ($i = n-1$), $\hat{a}_2 v_k^{(n-1)} = 0$, $k = 1, \ldots, n-1$, but, multiplying the relation for $v_k^{(n-1)}$ by $\hat{a}_1$, we see that $v_k^{(n-1)}$ is also orthogonal to $\hat{a}_1$. By induction, in the end, we obtain that $v_1^{(1)}$ is orthogonal to all the rows of $\hat{A}$. The solution x is obtained from the first n components of that vector divided by its last component.

Usually, $V^{(n+1)}$ is chosen as I_{n+1}, the identity matrix of order $n+1$, but it can be any nonsingulat matrix. With the choice of the identity, initially there are many zero entries in the vectors $v_j^{(i)}$. Let us illustrate that on the small example defined in (2.11). For the first step ($i = 4$), we have $p = 5$ and $m = [1, 2, 3, 4]$,

$$\alpha = [0.29412, 0.058824, 0.11765, 0.058824],$$

$$V^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0.29412 & 0.058824 & 0.11765 & 0.058824 \end{pmatrix}.$$

In the second step ($i = 3$), $p = 1$ and $m = [2, 3, 4]$,

$$\alpha = [0.95745, -0.13475, -0.12766], \quad V^{(3)} = \begin{pmatrix} 0.95745 & -0.13475 & -0.12766 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0.34043 & 0.078014 & 0.021277 \end{pmatrix}.$$

For the third step ($i = 2$), we obtain $p = 1$, $m = [2, 3]$,

$$\alpha = [0.51852, 0.11111], \quad V^{(2)} = \begin{pmatrix} 0.36170 & -0.021277 \\ 0.51852 & 0.11111 \\ 1 & 0 \\ 0 & 1 \\ 0.25453 & 0.059102 \end{pmatrix}.$$

In the last step ($i = 1$), $p = 1$, $m = [2]$,

$$\alpha = [0.75], \quad V^{(1)} = \begin{pmatrix} 0.25000 \\ 0.5.0000 \\ 0.75000 \\ 1 \\ 0.25000 \end{pmatrix}.$$

Dividing the first four components by 0.25, we find the solution $x = (\,1 \quad 2 \quad 3 \quad 4\,)^T$. In double precision arithmetic the norm of the error is $4.4409\ 10^{-16}$, but there is only one nonzero error component.

Implementing Purcell's algorithm with pivoting is particularly simple as shown in the following compact code.

```
function x = dpurcell(A,b);
A = [A, -b];
V = eye(n+1,n+1);
for i n:-1:1
 ni = n + 1 - i;
 av = A(ni,:) * V(:,1:i+1); % dot products, row vector x matrix
 [~,I] = max(abs(av));
 piv = I(1); % index of largest dot product
 m = [1:piv-1, piv+1:i+1];
 alp = -av(m) / av(piv);
 V = prank1(V(:,m),V(:,piv),alp); % rank-one update of V(:,m)
end % for i
x = V(1:n,1) / V(n+1,1);
```

Note that the matrix A is not modified in the algorithm and it is accessed one row at a time. In fact, the rows can be used in any order, as long as we keep a record of the ordering.

Purcell's algorithm and its relation to Gaussian elimination were studied by K. Zorychta [1167] in 1964. He proved that the algorithm succeeds in finding the solution if all the principal minors are different from zero, which is the same condition as for Gaussian elimination. He also showed how an LU factorization can be computed from Purcell's method without pivoting. This was also noticed by F. Sloboda [1022] in 1978, who gave a somewhat different presentation of Purcell's method.

The fact that we can obtain an LU factorization when we start from an identity matrix is easily seen from the construction of the vectors $v_k^{(i)}$. At each step we collect the first of the vectors as a new column of an $(n+1) \times n$ matrix Z with a first column equal to e_1, the first column of the identity matrix. Since the first rows of $\hat{A}$ are orthogonal to $v_1^{(i)}$, the second column of $\hat{A}Z$ has one zero at the top, the third column two zeros at the top, and so on. At the last step, $\hat{A}Z$ is a lower triangular matrix L. Without pivoting, $Z_{1:n,:}$ is an upper triangular matrix and, since $AZ_{1:n,:} = L$, we have

$$A = LU, \text{ with } U = (Z_{1:n,})^{-1}.$$

The reader may wonder if, having solved a linear system with a given right-hand side, we can easily solve another system with the same matrix and a different right-hand side. This was explained by C.J. Hegedüs [620]. In that paper he related Purcell's method to the work of E. Egerváry [393] in 1956-1960. In his first papers in 1956-1957 on Purcell's method, published in Hungarian, Egerváry showed that it can be given by his rank reduction technique. Egerváry's 1960 paper is in English. A detailed exposition of the rank reduction process can be found in A. Galántai [485].

In [620] the notation is different from what we used above. One starts from an identity matrix $V^{(1)}$ with $v_j^{(1)} = e_j$, $j = 1, \ldots, n+1$. Let us assume that, after column permutations, the pivot index is $p = i$ at the ith step. The new vectors are computed as

$$v_j^{(i+1)} = \left(I - \frac{v_i^{(i)} e_i^T \hat{A}}{e_i^T A v_i^{(i)}} \right) v_j^{(i)}, \quad j > i.$$

These vectors satisfy the first i equations. The nonzero columns span the subspace of vectors that satisfy the first i equations. It is easy to check that the matrix within parenthesis is a projection matrix. For projection methods, we must also mention T. Pietrzykowski [919], H.-Y. Huang [653], C. Brezinski [157], and A. Galántai [484]. Let us assume that we have another right-hand

side b' and that we would like to solve $Ax = b'$. According to [620], what we have to do, instead of what we have done above, is to start with $V^{(1)} = I_n$ and perform projections not for the augmented matrix, but simply for the matrix A. We store the vectors $v_1^{(1)}, v_2^{(2)}, \ldots, v_n^{(n)}$ and compute vectors

$$x^{(i)} = x^{(i-1)} + v_i^{(i)} \frac{b'_i - e_i^T A x^{(i-1)}}{e_i^T A v_i^{(i)}}, \quad i = 1, 2, \ldots, n, \quad x^{(0)} = 0.$$

However, this is not equivalent to using the augmented matrix $[A, -b]$ since the pivots are not the same. With the augmented matrix the choice of the pivot depends on the right-hand side, but this is not the case if we just use A and it has an effect on the accuracy of the solution.

The positive points for Purcell's algorithm are

- In a given step, only one row of the matrix A is needed. This can be important in some applications.
- Contrary to Gaussian elimination, the right-hand side is included in the pivot choice. However, this does not always improve the accuracy.
- It is not necessary to do row of column swaps and pivoting does not change the order of the components of the solution.
- The method is easy to parallelize. The dot products needed at one step are independent and the main operation is a rank-one update of a matrix.

The negative points are

- The storage is larger than for Gaussian elimination which can be done in place.
- The only BLAS3 operation is the rank-one update which can be seen as a matrix-matrix multiply.

K. Chen and D. Evans [243] showed that, without pivoting, the Gauss-Huard and Purcell's methods are equivalent, but this does not hold with pivoting. They modified the methods to achieve an equivalence. If we omit the entry arising from the right-hand side for choosing the pivot in Purcell's method, it is similar to the Gauss-Huard method. On Purcell's method and its relation to Gauss-Jordan, see C.-H. Lai [719, 720]. A block version of Purcell's method, for matrices partitioned in block, was proposed by K. Jbilou and A. Messaoudi [671], see also [798].

For parallel versions of Purcell's method, see K. Chen and C.-H. Lai [244] who described six parallel variants of Purcell's method. The algorithms differ in ways of pivoting and load balancing.

R. Fletcher [460] introduced in 1997 a method which, formally, looks like Purcell's. However, his method, which is called LIU (L Implicit U) factorization, operates on columns rather than on rows in Purcell's. The method computes a unit lower triangular matrix $L^{(n)}$ column by column. Let $L^{(0)} = I$ and assume that we have $L^{(k)}$ in which we have already computed the first k columns which are also those of $L^{(k+1)}$. Then, let $d_k = \ell_k^{(k)T} \ell_k$ where $\ell_j^{(k)T}$ is the jth row of $L^{(k)}$ and ℓ_j is the jth column of A. The entries of the new column below the diagonal are computed as

$$\ell_j^{(k+1)} = \ell_j^{(k)} - r_{j,k} \ell_k^{(k)}, \quad r_{j,k} = \frac{\ell_j^{(k)T} \ell_k}{d_k}, \quad j = k+1, \ldots, n.$$

At step $n-1$ we have a unit lower triangular matrix $L^{(n)}$. It is easy to see that it is the inverse of the L factor of the LU factorization and the ratios $r_{j,k}$ are the multipliers in Gaussian elimination and $L^{(n)}A$ is an upper triangular matrix. Partial pivoting can be introduced with row swaps. Let $b^{(n)} = b$, the solution of $Ax = b$ is computed as

$$x_i = \frac{\ell_i^{(n)T} b^{(i)}}{d_i}, \quad b^{(i-1)} = b^{(i)} - \ell_i x_i, \quad i = n, n-1, \ldots, 1.$$

This method is only one variant of Gaussian elimination without explicitly computing L and U. It turns out that a similar algorithm was proposed by S. Ursic [1081] in 1982.

Projection methods were considered in the 1990s by M. Benzi, see [111] and [116] with C.D. Meyer. Their method is very close to Purcell's, except that they used a normalization of the basis vectors and pivoting; see also M. Tůma [1077].

On projection methods for least squares problems, see C. Popa, T. Preclik, H. Köstler, and U. Rüde [924], and the references therein.

For the numerical experiments of figures 2.19-2.20, we use the random matrix A_{1000} and a random exact solution. We display results for Gaussian elimination (`dgetf2` and `dgetrf`), Purcell's method (`dpurcell`), and the projection method of Benzi and Meyer (`dpBM`). Figure 2.19 shows the residual norms. Purcell and the projection method give smaller residual norms than the Gaussian elimination variants. Figure 2.20 displays the error norms. There are of the same magnitude for all methods, even though those of Purcell's method are slightly smaller for some problem dimensions.

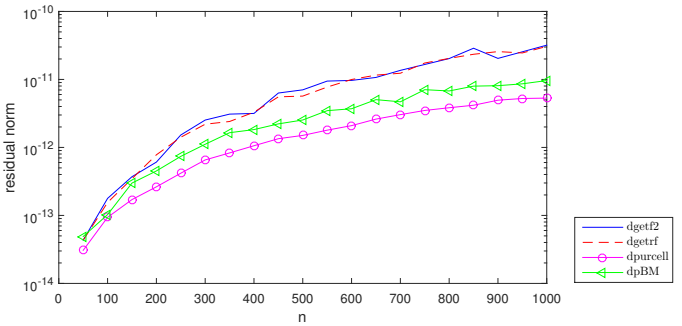


Figure 2.19. *$\|b - Ax\|$ as a function of n, with Gauss, Purcell and the projection method of Benzi and Meyer*

2.16 ▪ Conjugate direction methods

In this section we consider some conjugate direction methods because they are somewhat related to LU factorization. One of the earliest ones was considered by L. Fox, H.D. Huskey, and J.H. Wilkinson [465] in 1948. They observed that their method was almost identical with the escalator process of J. Morris [843]. The idea is to use a basis of A-conjugate vectors v_i. The solution of $Ax = b$, with A symmetric, is written as

$$x = \sum_{i=1}^{n} \alpha_i v_i, \quad v_j^T A v_i = 0, i \neq j.$$

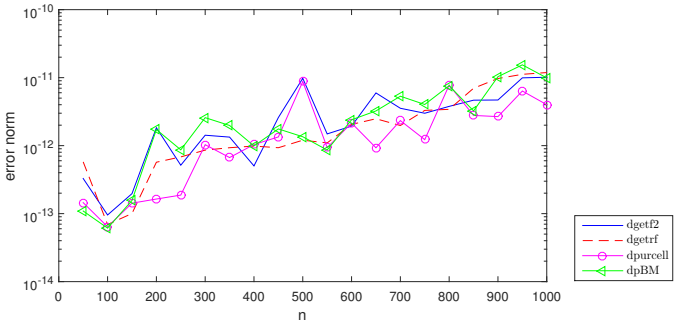


Figure 2.20. *Error norms as a function of n, with Gauss, Purcell and the projection method of Benzi and Meyer*

Therefore, we must have

$$\sum_{i=1}^{n} \alpha_i v_j^T A v_i = v_j^T b.$$

Because of the A-orthogonality, the coefficients α_i and the solution x are

$$\alpha_i = \frac{v_i^T b}{v_i^T A v_i}, \quad x = \sum_{i=1}^{n} \frac{v_i^T b}{v_i^T A v_i} v_i.$$

Let $Y = I_n$, the identity matrix of order n, The vectors v_i are obtained by A-orthogonalizing the columns of Y. First $v_i = y_1$, then

$$v_2 = y_2 + \lambda_2 v_1, \quad \lambda_2 = -\frac{v_1^T A y_2}{v_1^T A v_1} = -\frac{y_2^T A v_1}{v_1^T A v_1}.$$

The next step is

$$v_3 = y_3 + \lambda_3 v_1 + \mu_3 v_2, \ \ldots$$

and so on. It was observed by Fox, Huskey, and Wilkinson that the A-orthogonality of the basis vectors can be lost. Hence, they recommended computing the coefficients α_i by an iterative method.

This method was considered by M.R. Hestenes and E. Stiefel in Section 4 of their seminal paper [629] in 1952. In the conjugate gradient method for symmetric positive definite matrices the A-conjugate vectors are constructed by a short recurrence. They observed in Section 12 that, if the basis vectors are obtained by A-orthogonalizing the columns of the identity matrix, this is essentially Gaussian elimination without pivoting.

In 1958, M.R. Hestenes [627] constructed biorthogonal set of vectors for computing inverses of nonsymmetric matrices. His aim was to compute singular values and eigenvalues. Conjugate direction methods were also described by L. Fox [464].

A generalization of conjugacy was given by G.W. Stewart [1039] in 1973. A pair of matrices (U, V) is said to be A-conjugate if $L = V^T A U$ is lower triangular. The solution of the linear system is computed by the recurrence

$$x_k = x_{k-1} + \mu_k u_k, \quad \mu_k = \frac{v_k^T r_{k-1}}{v_k^T A u_k}, \quad r_{k-1} = b - A x_{k-1},$$

where u_k is the kth column of U. With this, we have $r_n = 0$ (in exact arithmetic) and the error vectors are given by $x - x_k = (I - P_k)(x - x_{k-1})$ where P_k is a projection matrix satisfying $P_i P_k = 0$ if $i < k$.

Let A, V, P be given matrices, Stewart computed a linear combination of $p_1, \ldots, p_k$ such that U and V are A-conjugate. To be able to do that, $V^T AP$ must be factorizable as LS where S is upper triangular. One has to find an upper triangular matrix S such that $U = PS^{-1}$. Then,

$$P = US \Rightarrow p_k = U_{k-1} s_k + \sigma_{k,k} u_k,$$

where s_k is a vector with components $(\sigma_{i,k})$. We must have $V_{k-1}^T A U_{k-1}$ lower triangular and $V_{k-1}^T A u_k = 0$,

$$(V_{k-1}^T A U_{k-1}) s_k = V_{k-1}^H A p_k.$$

The $\sigma_{k,k}$'s are scaling parameters to be chosen. Note that the choice of v_k can be made dependent on $u_1, \ldots, u_k$.

The choice $P = V = I$ gives an LU factorization, as well as $P = I$ and $V = U$. When A is symmetric, $V^T AV$ is diagonal. If $P = I$ and $V = A$, $V^T AP = A^T A = LU$. That choice leads to a QR factorization. Another choice is $p_i = A^{i-1} p_1$. If V is arbitrary and P conjugated with respect to V, it gives a reduction of A to upper Hessenberg form.

As we said above, P. Laurent-Gengoux and D. Trystram [723] showed that, for symmetric matrices, the Gauss-Huard algorithm without pivoting is equivalent to a conjugate direction method.

An inverse factorization algorithm $A^{-1} = ZD^{-1}Z^T$, with Z unit upper triangular and D diagonal, was proposed by M. Benzi, C.D. Meyer, and M. Tůma [117] for symmetric matrices. Their aim was to construct a preconditioner for sparse matrices and iterative methods by dropping some entries during the computation of Z, see Chapter 7. This method is very close to that in [116]. M. Benzi and M. Tůma extended this method to nonsymmetric matrices [120]. Two sets of A-biconjugate vectors z_i, w_i are constructed to obtain $W^T AZ = D$ diagonal with Z and W upper triangular. Without dropping entries, this method is close to what was done by L. Fox [464]. Since $A = W^{-T} DZ^{-1}$, we recognize an LDU factorization of A. The solution of a linear system is obtained by direct multiplication, $x = ZD^{-1}W^T b$.

The authors of [120] warn us that "In practice, this direct method for solving linear systems is not used on account of its cost: for a dense $n \times n$ matrix, the biconjugation process requires about twice the work as the LU factorization of A". Moreover, a loss of A-orthognality may happen which can spoil the accuracy. In that case iterative refinement is needed increasing even more the cost of the algorithm.

For the sake of constructing robust preconditioners, M. Bollhöfer and Y. Saad [140] studied how to introduce pivoting in this A-orthogonalization method. The pivots are found iteratively to satisfy some criterion. An (non-optimal) implementation of this method without any dropping of entries is the following. Most of code lines are devoted to finding the pivots.

```
function [Z,Wt,p,row,col]=biconj_ZW(A);
n = size(A,1);
EY = eye(n,n);
Z = eye(n,n); W = eye(n,n);
p = zeros(n,1); q = zeros(n,1);
row = 1:n; col = 1:n;
tol = 0.1;
npiv = 20;
for i=1:n
```

```
piv = 0; satp = 0; satq = 0;
% pivoting
k = 0;
while satp == 0 && k < npiv
 k = k + 1;
 p(i:n) = W(:,i:n)' * (A * Z(:,i));
 [val,I] = max(abs(p(i:n)));
 if abs(p(i)) < tol * val
  satq = 0;
  ip = i + I(1) - 1; % global index
  if ip $\sim$= i
   piv = 1;
   A(:,[i,ip]) = A(:,[ip,i]);
   ZI = Z - EY;
   ZI(:,[i,ip]) = ZI(:,[ip,i]);
   Z = ZI + EY;
   p([i,ip]) = p([ip,i]);
   col([i,ip]) = col([ip,i]);
  end % if ip
 end % if abs
 satp = 1;
 % pivoting
 if satq == 0
  q(i:n) = (W(:,i)' * A) * Z(:,i:n);
 end % if
 [val,I] = max(abs(q(i:n)));
 if abs(q(i)) < tol * val
  satp = 0;
  ip = i + I(1) - 1; % global index
  if ip $\sim$= i
   piv = 1;
   A([i,ip],:) = A([ip,i],:);
   WI = W - EY;
   WI(:,[i,ip]) = WI(:,[ip,i]);
   W = WI + EY;
   q([i,ip]) = q([ip,i]);
   row([i,ip]) = row([ip,i]);
  end % if ip
 end % if abs
 satq = 1;
end % while
if piv == 1
 % must recompute p and q with the new A, Z, W
 p(i:n) = W(:,i:n)' * (A * Z(:,i));
 q(i:n) = (W(:,i)' * A) * Z(:,i:n);
end % if
p1 = 1 / p(i);
pp1 = p1 * p(i+1:n);
qq1 = p1 * q(i+1:n);
for j=i+1:n
```

```
  Z(:,j) = Z(:,j) - qq1(j-i) * Z(:,i);
  W(:,j) = W(:,j) - pp1(j-i) * W(:,i);
 end % for j
end % for i
Wt = W';
```

Working along the same lines as G.W. Stewart [1039] and based on the work of J.H.M. Wedderburn [1113], M.T. Chu, R.E. Funderlic, and G.H. Golub [262] explored the relations between A-conjugation and several matrix factorizations.

Wedderburn [1113, p. 69] showed that if x and y are vectors such that $\omega = y^T Ax \neq 0$, the matrix $B = A - \omega^{-1} Axy^T A$ has rank exactly one less than A. In fact, if $B = A - \sigma^{-1} uv^T$, rank(B)=rank(A) -1 if and only if there exist vectors x and y such that $u = Ax$, $v = A^T y$, and $\sigma = y^T Ax$. This is also related to the work of E. Egerváry [393] who was probably not aware of Wedderburn's earlier work.

In the rank-reduction process a sequence of matrices is computed as

$$A_{k+1} = A_k - \omega_k^{-1} A_k x_k y_k^T A_k, \quad \omega_k = y_k^T A_k x_k, \quad A_1 = A,$$

for any sequences of vectors x_k and y_k, for which $\omega_k \neq 0$. The rank decreases by one at each step and the recurrence stops at some index $m \leq n$. The matrices A_k are called *Wedderburn matrices* in [262]. In matrix form this can be written as $A = \mathcal{U}\Omega^{-1}\mathcal{V}^T$, where Ω is a diagonal matrix with diagonal entries ω_j, $j = 1, \ldots, m$, the columns of $\mathcal{U}$ (resp. $\mathcal{V}$) are $A_j x_j$ (resp. $A_j^T y_j$).

It turns out that this process can be obtained without a reference to the Wedderburn matrices. Let $< x, y >_A = y^T Ax$. From the x_j's and y_j's, we define

$$u_k = x_k - \sum_{i=1}^{k-1} \frac{< x_k, v_i >_A}{< u_i, v_i >_A} u_i,$$
$$v_k = y_k - \sum_{i=1}^{k-1} \frac{< u_i, y_k >_A}{< u_i, v_i >_A} v_i.$$

Then,

$$Au_k = A_k x_k, \quad v_k^T A = y_k^T A_k, \quad \omega_k = y_k^T Ax_k = < u_k, v_k >_A,$$

and $< u_k, v_j >_A = < u_j, v_k >_A = 0$ for $j < k$. If U_k (resp. V_k) has columns u_j (resp. v_j) for $j = 1, \ldots, k \leq m$,

$$V_k^T AU_k = \Omega_k, \quad A = AU_m \Omega_m^{-1} V_m^T A.$$

If $m = n$, $U_n \Omega_n^{-1} V_n^T$ is the inverse of A. If X_k, Y_k are the matrices with columns x_j and y_j involved in a rank-reduction process, there exist unit upper triangular matrices $R_k^{(x)}$, $R_k^{(y)}$ such that

$$X_k = U_k R_k^{(x)}, \quad Y_k = V_k R_k^{(y)}.$$

A necessary and sufficient condition for the ω_j's to be different from zero is that $Y_k^T AX_k$ has an LDU factorization; in fact $\det(Y_k^T AX_k) = \prod_{i=1}^k \omega_i$.

If X_m and Y_m are chosen as upper trapezoidal matrices, AU_m and $A^T V_m$ are lower trapezoidal, and one obtains a trapezoidal LDU factorization of A. If A is square with all nonzero principal minors, $m = n$, and $X_n = Y_n = I_n$, one obtains an LDU factorization of A, as it was already shown by E. Egerváry [393]. If, moreover, A is symmetric, $U_m = V_m$, and we have a Cholesky-like factorization.

If A has full column rank, the choice $X = I$, $Y = A$, leads to $U = R_1^{-1}$ and $V = QD$, where R_1 is a unit upper triangular matrix, Q is an orthonormal matrix, and D is diagonal. It

gives $A = QDR_1$. On rank-reduction formulas, see also N. Mahdavi-Amiri and E. Golpar-Raboky [771].

We will meet other A-biconjugation processes when studying the Lanczos iterative algorithms.

For the numerical experiments of figures 2.21-2.22, we use our random matrix A_{1000} and a random exact solution. We display results for Gaussian elimination (`dgetf2` and `dgetrf`), Purcell's method (`dpurcell`), and the biconjugation algorithm without (`ZW`) and with iterative refinement (`ZW ref`). For the iterative refinement the residual vector is computed in extended precision with 32 decimal digits. Figure 2.21 shows the residual norms. The biconjugation algorithm gives residual norms much larger than the Gaussian elimination variants and Purcell's method, but one iteration of iterative refinement is efficient to reduce the residual norms. Figure 2.22 displays the error norms for which we observe the same trends.

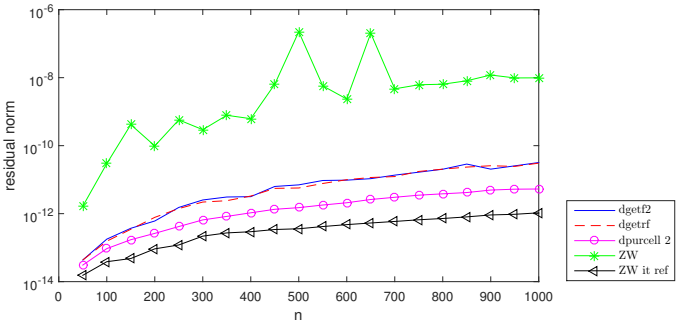


Figure 2.21. *$\|b - Ax\|$ as a function of n, with Gauss, Purcell and the biconjugation algorithm*

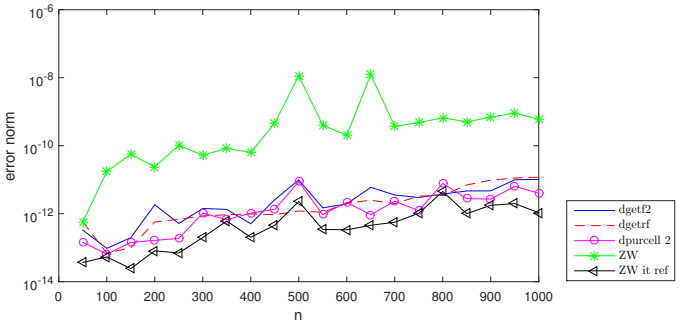


Figure 2.22. *Error norms as a function of n, with Gauss, Purcell and the biconjugation algorithm*

2.17 ▪ The WZ factorization

The WZ factorization of a matrix A was introduced in 1979 by D.J. Evans and M. Hatzopoulos [432]. It is also known on the less appealing name, Quadrant Interlocking Factorization (QIF). Their aim was to obtain a factorization with more potential parallelism than the LU factorization. In LU, we put zeros below the diagonal of A sequentially, column by column. In WZ, entries

are zeroed by columns but starting from the left and at the same time from the right. Let us show how to do that on a small example of order 6,

$$A = \begin{pmatrix} 17 & 6 & 20 & -16 & 14 & 15 \\ 19 & 11 & -10 & 20 & 16 & 1 \\ 3 & 20 & 17 & 14 & 15 & -6 \\ -19 & 20 & 3 & 1 & 8 & 1 \\ 13 & 4 & 9 & 17 & 14 & 2 \\ 2 & -20 & -19 & 19 & 4 & 17 \end{pmatrix}.$$

At the first step, we multiply to the left by the matrix (rounded to three decimal digits),

$$W^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1.239 & 1 & 0 & 0 & 0 & 1.035 \\ -0.2432 & 0 & 1 & 0 & 0 & 0.5676 \\ 1.255 & 0 & 0 & 1 & 0 & -1.166 \\ -0.8378 & 0 & 0 & 0 & 1 & 0.6216 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and the result is

$$A^{(1)} = \begin{pmatrix} 17 & 6 & 20 & -16 & 14 & 15 \\ 0 & -17.13 & -54.45 & 59.49 & 2.788 & 0 \\ 0 & 7.189 & 1.351 & 28.68 & 13.86 & 0 \\ 0 & 50.85 & 50.25 & -41.23 & 20.9 & 0 \\ 0 & -13.46 & -19.57 & 42.22 & 4.757 & 0 \\ 2 & -20 & -19 & 19 & 4 & 17 \end{pmatrix},$$

with zeros in the first and last columns. Then, we proceed with columns 2 and 5, multiplying with

$$W^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 5.022 & 1 & 0 & -5.858 & 0 \\ 0 & 11.9 & 0 & 1 & -11.37 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The final result is

$$Z = A^{(2)} = \begin{pmatrix} 17 & 6 & 20 & -16 & 14 & 15 \\ 0 & -17.13 & -54.45 & 59.49 & 2.788 & 0 \\ 0 & 0 & -157.5 & 80.14 & 0 & 0 \\ 0 & 0 & -375.2 & 186.8 & 0 & 0 \\ 0 & -13.46 & -19.57 & 42.22 & 4.757 & 0 \\ 2 & -20 & -19 & 19 & 4 & 17 \end{pmatrix}.$$

Hence, we have $W^{(2)}W^{(1)}A = Z$. The inverses of the matrices $W^{(i)}$ are obtained straightforwardly since it is enough to negate some entries. For instance,

$$[W^{(1)}]^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1.239 & 1 & 0 & 0 & 0 & -1.035 \\ 0.2432 & 0 & 1 & 0 & 0 & -0.5676 \\ -1.255 & 0 & 0 & 1 & 0 & 1.166 \\ 0.8378 & 0 & 0 & 0 & 1 & -0.6216 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Moreover, we have $A = WZ$ with

$$W = [W^{(1)}]^{-1}[W^{(2)}]^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1.239 & 1 & 0 & 0 & 0 & -1.035 \\ 0.2432 & -5.022 & 1 & 0 & 5.858 & -0.5676 \\ -1.255 & -11.9 & 0 & 1 & 11.37 & 1.166 \\ 0.8378 & 0 & 0 & 0 & 1 & -0.6216 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

It means that we do not have to compute the product of the inverses, we just have to put in proper places minus the entries that we have computed at each step.

This process can be generalized. Let us assume that A is of order n with n even and write $W^{(1)}$ and A in block form,

$$W^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ w^{(1)} & I_{n-2} & w^{(n)} \\ 0 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} a_{1,1} & a_{1,:} & a_{1,n} \\ a_{:,1} & \hat{A} & a_{:,n} \\ a_{n,1} & a_{n,:} & a_{n,n} \end{pmatrix}.$$

The first and last rows of the product $W^{(1)}A$ are those of A. In the middle, for the first and last columns that we want to zero, we have the vector equations,

$$\begin{aligned} a_{1,1}w^{(1)} + a_{:,1} + a_{n,1}w^{(n)} &= 0, \\ a_{1,n}w^{(1)} + a_{:,n} + a_{n,n}w^{(n)} &= 0. \end{aligned}$$

Taking the first two equations,

$$\begin{aligned} a_{1,1}w_1^{(1)} + a_{2,1} + a_{n,1}w_1^{(n)} &= 0, \\ a_{1,n}w_1^{(1)} + a_{2,n} + a_{n,n}w_1^{(n)} &= 0. \end{aligned}$$

This is a linear system of order 2,

$$\begin{pmatrix} a_{1,1} & a_{n,1} \\ a_{1,n} & a_{n,n} \end{pmatrix} \begin{pmatrix} w_1^{(1)} \\ w_1^{(n)} \end{pmatrix} = \begin{pmatrix} -a_{2,1} \\ -a_{2,n} \end{pmatrix}.$$

We can proceed similarly with the other equations for the other components of $w^{(1)}$ and $w^{(n)}$. Finally, we have $n-2$ independent linear systems of order 2, all with the same matrix but different right-hand sides, that can be solved in parallel. When they are solved, we can compute the term in the middle of the product $W^{(1)}A$,

$$\tilde{A} = \hat{A} + w^{(1)}a_{1,:} + w^{(n)}a_{n,:},$$

which is a rank-two update of $\hat{A}$. The next step is to use that same process on $\tilde{A}$.

Things are a little bit different if n is odd. Figure 2.23 shows the nonzero structure of W and Z for $n = 11$. In W, we have one full row $W_{6,:}$ and, in Z, the 6th row has just one nonzero entry.

A function to compute the WZ factorization without pivoting for any n is the following.

```
function [W,Z] = WZ_np(A);
n = size(A,1);
W = eye(n,n); Z = zeros(n,n);
kstart = 1; kend = n;
```

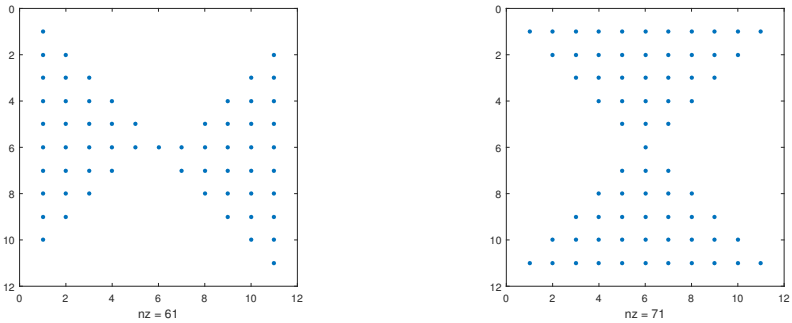


Figure 2.23. *Nonzero structure of W (left) and Z (right), $n = 11$*

```
sA = n;
while sA > 2
 % first and last rows
 W(kstart,kstart:kend) = eye(1,sA);
 W(kend,kstart:kend) = [zeros(1,sA-1) 1];
 Z(kstart,kstart:kend) = A(kstart,kstart:kend);
 Z(kend,kstart:kend) = A(kend,kstart:kend);
 % 2 x 2 matrix
 A2 = [A(kstart,kstart) A(kend,kstart);
 A(kstart,kend) A(kend,kend)];
 if abs(det(A2)) < 1e-14
  error('WZ_np: too small determinant of A2')
 end % if
 % right-hand sides
 rhs = [A(kstart+1:kend-1,kstart)';
 A(kstart+1:kend-1,kend)'];
 X2 = A2 \ rhs; \% solve all the 2 x 2 systems
 W(kstart+1:kend-1,kstart) = X2(1,:);
 W(kstart+1:kend-1,kend) = X2(2,:);
 % reduced matrix
 sA2 = sA - 2;
 UP = zeros(sA2,sA2);
 ws = -W(kstart+1:kend-1,kstart);
 we = -W(kstart+1:kend-1,kend);
 zs = Z(kstart,kstart+1:kend-1);
 ze = Z(kend,kstart+1:kend-1);
 for j=1:sA2
  % rank-two update
  UP(:,j) = A(kstart+1:kend-1,kstart+j)...
   + zs(j) * ws + ze(j) * we;
 end % for j
 A(kstart+1:kend-1,kstart+1:kend-1) = UP;
 sA = sA2;
 kstart = kstart + 1;
 kend = kend -1;
```

```
end % while
Z(kstart:kend,kstart:kend) = A(kstart:kend,kstart:kend);
```

Properties of the WZ factorization had been studied in the 1980s and 1990s; see P. Yalamov and D.J. Evans who did a rounding error analysis [1133, 1134], C.S. Rao who considered the existence and uniqueness of the factorization [936], and M. Kaps and M. Schlegl [692]. Some people implemented WZ on parallel computers, see I. García, J.J. Merelo, J.D. Bruguera, and E.L. Zapata [495] and E. Asenjo, M. Ujaldón, and EL. Zapata [44].

However, since, as it was noticed by D.J. Tylavsky [1079], the WZ factorization can be obtained from a block LU factorization with 2×2 blocks (see Section 2.5), it is enough to look at the existence and properties of this block factorization. Let us see how this works on our small example above with $n = 6$. We group the rows with the same number of nonzero entries, that is, (1,6), (2,5) and (3,4). This is done by the permutation vector $(\,1 \quad 4 \quad 5 \quad 6 \quad 3 \quad 2\,)$. Let P be the corresponding permutation matrix, then

$$PWP^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0.8378 & -0.6216 & 1 & 0 & 0 & 0 \\ 1.239 & -1.035 & 0 & 1 & 0 & 0 \\ 0.2432 & -0.5676 & 5.858 & -5.022 & 1 & 0 \\ -1.255 & 1.166 & 11.37 & -11.9 & 0 & 1 \end{pmatrix},$$

which is a block diagonal matrix, but in fact a lower unit triangular matrix. Applying the same permutations to Z,

$$PZP^T = \begin{pmatrix} 17 & 15 & 14 & 6 & 20 & -16 \\ 2 & 17 & 4 & -20 & -19 & 19 \\ 0 & 0 & 4.757 & -13.46 & -19.57 & 42.22 \\ 0 & 0 & 2.788 & -17.13 & -54.45 & 59.49 \\ 0 & 0 & 0 & 0 & -157.5 & 80.14 \\ 0 & 0 & 0 & 0 & -375.2 & 186.8 \end{pmatrix},$$

which is a truly block upper triangular matrix. Therefore, $PAP^T = (PWP^T)(PZP^T) = LU$, and $WZ = P^T LUP$.

As it is stated in [633, Section 13.1, Th. 13.2], such a factorization exists and is unique if and only if the first $m - 1$ leading principal block submatrices of PAP^T are nonsingular, where m is the number of blocks on the diagonal. The stability of block LU factorization can be different and worse from that of LU factorization. It was studied by J.W. Demmel, N.J. Higham, and R.S. Schreiber [337] who proved that in finite precision arithmetic,

$$LU = A + \Delta A_1, \quad (A + \Delta A_2)x = b,$$

$$\frac{\|\Delta A_i\|}{\|A\|} \le c_n u \left(1 + \frac{\|L\| \, \|U\|}{\|A\|}\right) + O(u^2), \quad i = 1, 2,$$

where the norm is the maximum norm. The ratio of norms in the right-hand side can be arbitrary large. However, block LU factorization is stable if A is diagonally dominant by columns, see [633]. A problem with the WZ factorization is that it is not easy to find a good pivoting strategy to improve the accuracy when solving a linear system. It is difficult to completely control the size of the multipliers that are used in the rank-two update, and it can lead to an amplification of the rounding errors. A possible strategy is to use the pivots computed from an LU factorization of the matrix `[A(kstart:kend,kstart), A(kstart:kend,kend)]` with two columns, but this is expensive.

There was a renew of interest for the WZ factorization in the 2000s, see B. Bylina and J. Bylina [189, 190, 191, 192, 193], even though these papers contain many typos and some errors. See also D. Bashir, H. Kamarulhailia, and O. Babarinsa [101].

Let us use again the random matrix A_{1000} and a random exact solution for the numerical experiments of figures 2.24-2.25. We display results for Gaussian elimination (`dgetf2` and `dgetrf`), Gaussian elimination without pivoting (`dgetf2 np`), and the WZ algorithm without (`WZ np`) and with pivoting (`WZ`). Figure 2.24 shows the residual norms. The WZ algorithm without pivoting gives residual norms larger than those of Gaussian elimination without pivoting, but pivoting brings the WZ residual norms to the same level as those of Gaussian elimination with pivoting. Figure 2.25 displays the error norms for which we observe the same trends.

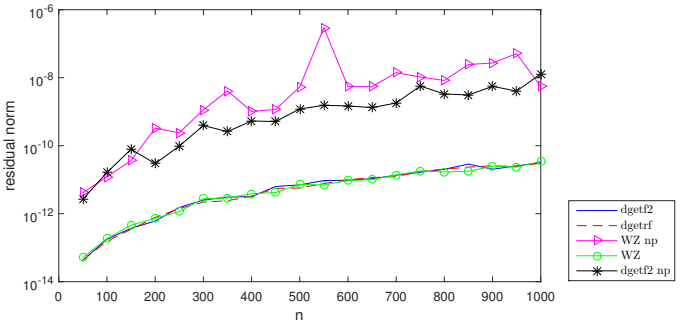


Figure 2.24. $\|b - Ax\|$ *as a function of* n*, with Gauss and WZ with and without pivoting*

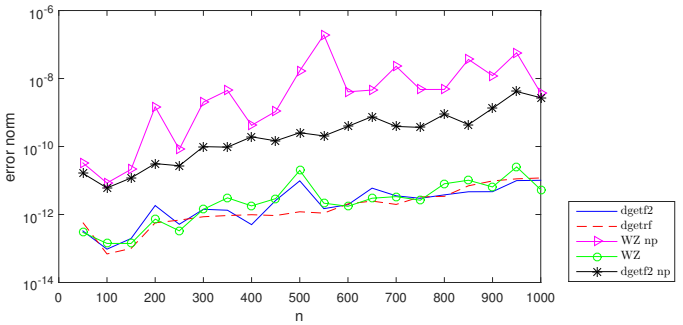


Figure 2.25. *Error norms as a function of* n*, with Gauss and WZ with and without pivoting*

2.18 ▪ More numerical experiments

The random matrix A_{1000} that we used above is relatively well-conditioned with $\kappa(A) = 4.7985\,10^3$, $\max_i \sigma_i = 63.087$ and $\min_i \sigma_i = 1.3147\,10^{-2}$. To do numerical experiments with matrices with different condition numbers, let us now use matrices generated with the "randsvd" option of Matlab `gallery` with a parameter $mode = 2$. For us, it means that all the singular values are equal to 1, except the smallest one which is equal to 10^{-k} for $k = 1, 2, \ldots, 8$. Therefore, $\|A\| = 1$ and we have

$$\frac{\|x_e - x\|}{10^k} \leq \|b - Ax\| \leq \|x_e - x\|.$$

The exact solutions x_e are random vectors and the right-hand sides are $b = Ax_e$.

Figures 2.26-2.27 show Gaussian elimination residual and error norms with different pivoting strategies as functions of the condition number. The residual norms are almost constant but much larger without pivoting. The error norms are increasing with the condition number, loosing almost five orders of magnitude from $k = 1$ to $k = 8$.

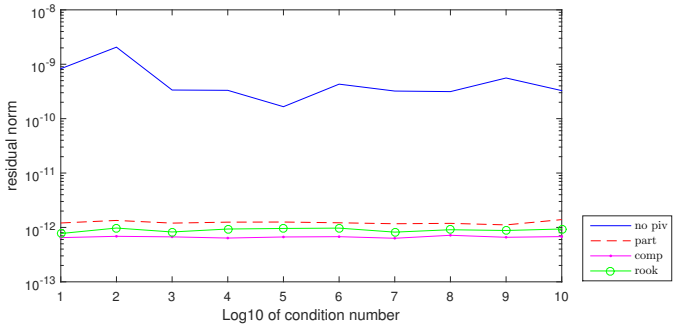


Figure 2.26. *randsvd, $\|b - Ax\|$ with different pivoting strategies as a function of the condition number*

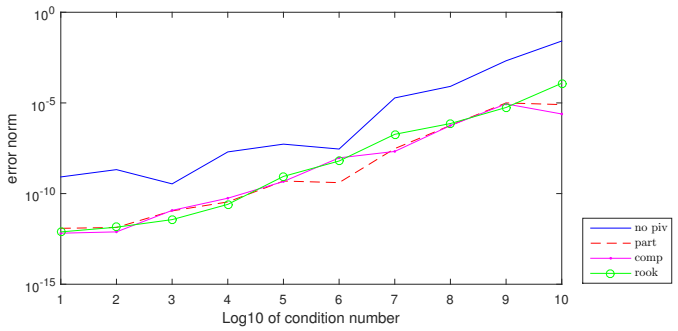


Figure 2.27. *randsvd, error norm with different pivoting strategies as a function of the condition number*

Figures 2.28-2.29 display the residual and error norms for different variants of Gaussian elimination as functions of the condition number. In Figure 2.28 we do not show the residual norms without pivoting because they are much larger than the other ones. All the variants with pivoting give more or less the same residual and error norms.

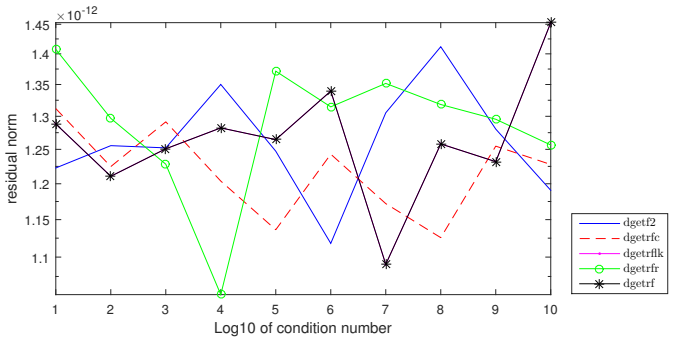


Figure 2.28. *randsvd,* $||b - Ax||$ *with variants of Gaussian elimination as a function of the condition number*

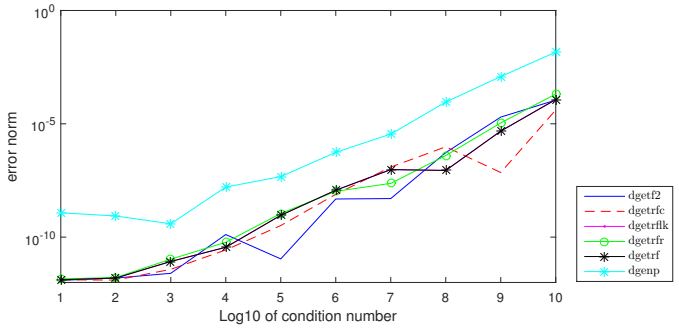


Figure 2.29. *randsvd, error norm with variants of Gaussian elimination as a function of the condition number*

Figures 2.30-2.31 show the residual and error norms for Gauss, Gauss-Jordan, and Gauss-Huard, as well as block versions of these last two. There are differences in the residual norms, but the error norms are almost all the same.

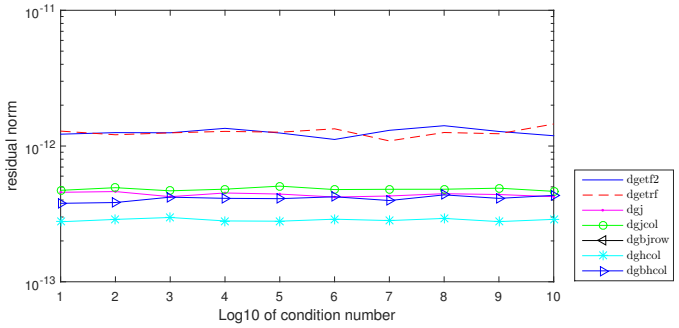


Figure 2.30. *randsvd,* $||b - Ax||$ *with Gauss, Gauss-Jordan and Gauss-Huard as a function of the condition number*

Figures 2.32-2.33 show the residual and error norms for Gauss, Purcell, and the Benzi-Meyer projection method. The residual norms of these two last methods are smaller than those of

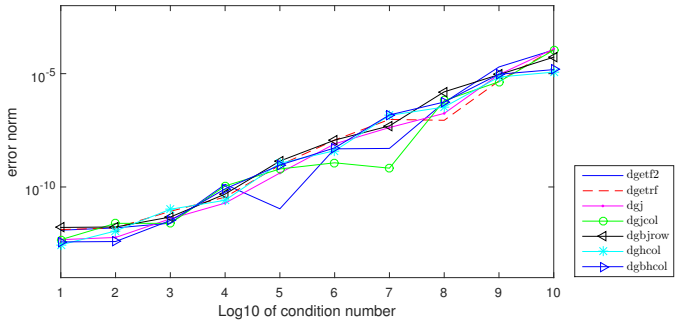


Figure 2.31. *randsvd, error norm with Gauss, Gauss-Jordan and Gauss-Huard as a function of the condition number*

Gaussian elimination, but the error norms are not much different.

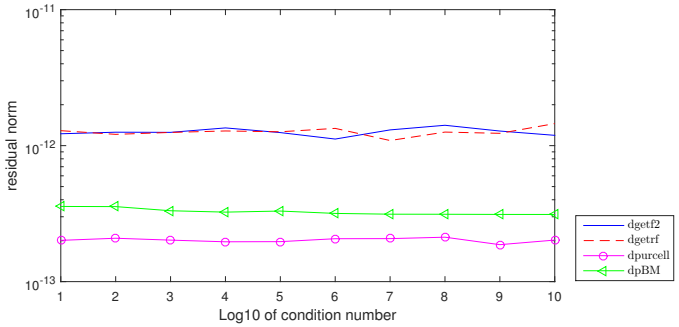


Figure 2.32. *randsvd, $\|b - Ax\|$ with Gauss, Purcell and Benzi-Meyer as a function of the condition number*

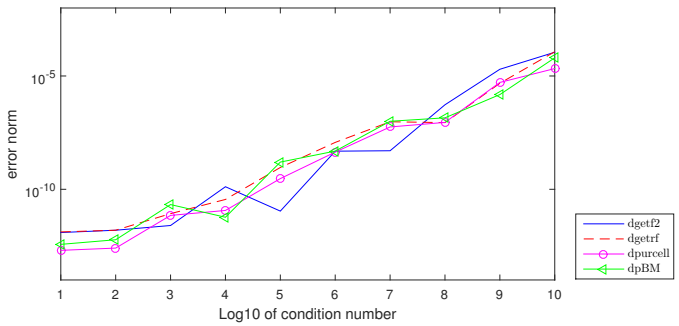


Figure 2.33. *randsvd, error norm with Gauss, Purcell and Benzi-Meyer as a function of the condition number*

Figures 2.34-2.35 show the residual and error norms for Gauss, Purcell and the ZW biconjugation method. The ZW biconjugation method is much worse than the other ones.

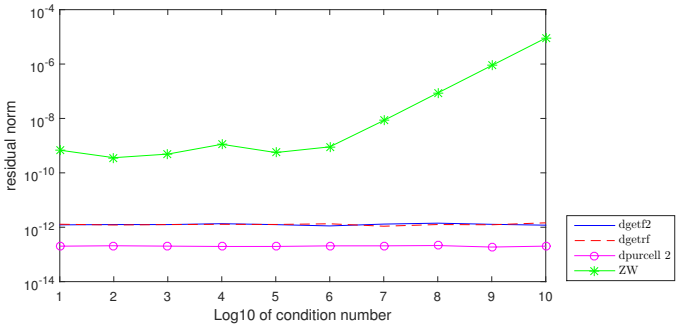


Figure 2.34. *randsvd, $||b - Ax||$ with Gauss, Purcell and ZW biconjugation as a function of the condition number*

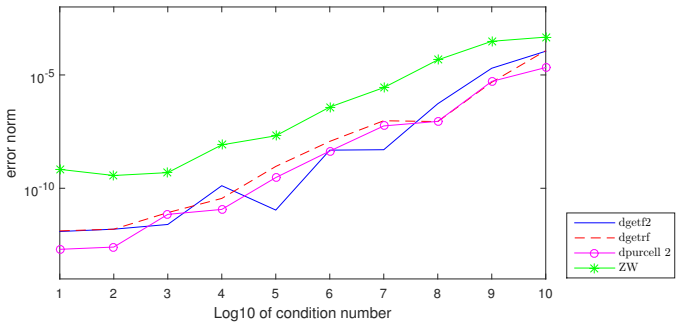


Figure 2.35. *randsvd, error norm with Gauss, Purcell and ZW biconjugation as a function of the condition number*

Figures 2.36-2.37 show the residual and error norms for Gauss and the WZ method with and without pivoting. WZ without pivoting is slightly worse than Gaussian elimination without pivoting and much worse than the other methods, even for well-conditioned matrices.

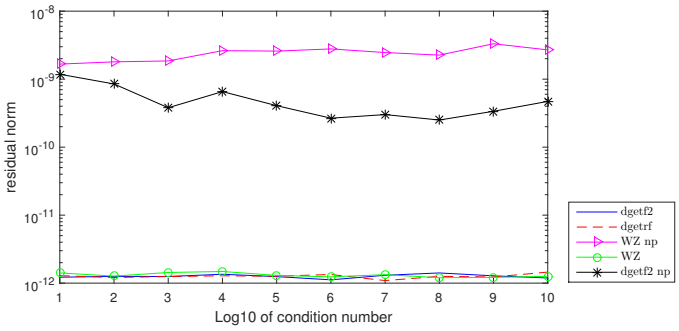


Figure 2.36. *randsvd, $||b - Ax||$ with Gauss and WZ as a function of the condition number*

Random matrices are not really representative of practical problems. In tables 2.4-2.11 we

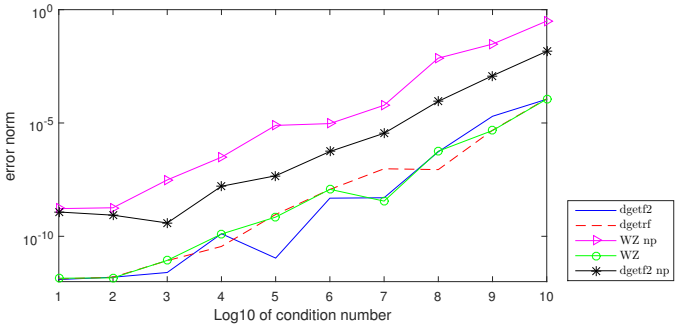


Figure 2.37. *randsvd, error norm with Gauss and WZ as a function of the condition number*

use a set of matrices from the SparseSuite[3] collection. They are ordered by increasing condition numbers. Their characteristics are given in Appendix 1. These matrices are sparse, but in this chapter we consider them as dense matrices, storing all the zero entries. The acronyms for the methods we consider are explained in Table 2.3. We give the absolute residual norms (tables 2.4-2.5) and relative residual norms (tables 2.6-2.7). The absolute error norms are in tables 2.8-2.9 and the relative error norms in tables 2.10-2.11.

For the matrices with large condition numbers, the absolute residual norms are large, but the relative ones are of the order of the machine precision. This is not the case for the error norms, since most of the relative error norms for these matrices are quite large. There are some breakdowns (NaN) for methods without pivoting (`dgetf2 np, ZW`, and `WZ np`) meaning that there was no LU factorization for these matrices.

Gaussian elimination with partial pivoting (`dgetf2, dgetrf`) does not always give the smallest residual or error norms, but they are quite robust. In our sequential implementations, `dgetrf` is the fastest method. The Gauss-Huard method is also quite reliable. Without surprise, the biconjugation ZW method does not give good results, but it is usually only used to construct parallel preconditioners with some dropping strategies.

Table 2.3. *Acronyms of methods*

meth.	
dgetf2 np	Gaussian elimination without pivoting
dgetf2	Gaussian elimination with partial pivoting
dgetrf	blocked Gaussian elimination with partial pivoting (nb = 40)
dgjsv	Gauss-Jordan with pivoting (row swaps)
dgjsvr	Gauss-Jordan with pivoting (column swaps)
dgbjsvr	block Gauss-Jordan with pivoting (column swaps)
dghsvr	Gauss-Huard with pivoting (column swaps)
dgbhsvr	block Gauss-Huard with pivoting (column swaps)
dpurcell	Purcell orthogonalization method
dBM	Benzi-Meyer projection method
ZW	biconjugation method
WZ np	WZ without pivoting
WZ	WZ with pivoting (row swaps)

[3] https://sparse.tamu.edu/

Table 2.4. *Residual norms*

meth.	pde225	gre 243	jpwh 991	jagmesh1	bfwa782 782
dgetf2 np	2.287e-14	1.446e-12	1.168e-13	NaN	2.386e-13
dgetf2	2.287e-14	5.957e-15	1.171e-13	7.861e-14	1.264e-13
dgetrf	2.299e-14	7.196e-15	1.064e-13	8.102e-14	1.073e-13
dgjsv	4.532e-14	1.276e-14	2.010e-13	2.129e-13	2.444e-13
dgjsvr	4.532e-14	1.202e-14	2.010e-13	2.479e-13	1.597e-13
dgbjsvr	2.862e-14	1.163e-14	7.743e-14	4.605e-13	1.769e-13
dghsvr	3.919e-14	1.060e-14	2.025e-13	2.422e-13	1.215e-13
dgbhsvr	1.998e-14	7.902e-15	5.918e-14	1.781e-13	6.980e-14
dpurcell	4.907e-14	7.321e-15	1.941e-13	1.747e-13	1.954e-13
dBM	4.101e-14	1.150e-14	2.034e-13	3.257e-13	1.098e-13
ZW	1.021e-13	NaN	2.809e-13	NaN	6.371e-12
WZ np	2.164e-14	5.395e-12	1.144e-13	NaN	9.718e-14
WZ	2.161e-14	9.405e-15	1.288e-13	1.119e-13	8.943e-14

meth.	jagmesh2 1009	lshp1009	fs 680 1c	fs 680 1	sherman1 1000
dgetf2 np	NaN	NaN	1.301e-14	0.09450	1.644e-14
dgetf2	9.756e-14	9.756e-14	1.592e-14	0.09450	1.587e-14
dgetrf	1.022e-13	1.022e-13	1.473e-14	0.09041	1.237e-14
dgjsv	2.875e-13	2.875e-13	4.300e-13	0.06729	7.410e-14
dgjsvr	2.690e-13	2.690e-13	1.640e-14	0.06729	3.370e-14
dgbjsvr	5.750e-13	5.750e-13	3.990e-13	0.09526	1.064e-13
dghsvr	2.772e-13	2.772e-13	7.981e-15	0.04888	2.757e-14
dgbhsvr	1.902e-13	1.902e-13	5.061e-15	0.03236	1.275e-14
dpurcell	2.109e-13	2.109e-13	1.144e-14	0.08065	3.994e-14
dBM	3.280e-13	3.280e-13	8.278e-15	0.05311	3.022e-14
ZW	NaN	NaN	1.415e-12	0.60230	9.817e-13
WZ np	NaN	NaN	1.404e-14	0.08732	1.355e-14
WZ	1.014e-13	1.014e-13	1.378e-14	0.08732	1.297e-14

meth.	nos3 960	cavity05 1182	e05r0500 236	olm1000	steam2 600
dgetf2 np	3.245e-12	1.457e-13	4.053e-12	1.501e-10	1.145e-05
dgetf2	3.245e-12	4.601e-13	1.933e-13	8.197e-11	1.178e-05
dgetrf	2.990e-12	4.314e-13	1.957e-13	8.262e-11	1.098e-05
dgjsv	9.156e-12	8.853e-12	8.619e-12	1.971e-07	1.527e-05
dgjsvr	9.156e-12	3.333e-13	1.911e-13	3.073e-09	1.486e-05
dgbjsvr	2.105e-11	7.672e-11	1.605e-09	3.914e-06	1.196e-05
dghsvr	8.254e-12	2.138e-13	1.415e-13	3.154e-09	1.706e-05
dgbhsvr	2.935e-12	1.181e-13	1.108e-13	1.221e-09	1.138e-05
dpurcell	8.852e-12	2.183e-13	1.314e-13	3.928e-10	2.566e-05
dBM	8.608e-12	2.190e-13	1.319e-13	8.033e-10	1.776e-05
ZW	3.905e-10	2.688e-10	NaN	NaN	2.495e-05
WZ np	3.502e-12	NaN	NaN	1.580e-10	1.388e-05
WZ	3.552e-12	3.287e-13	1.344e-13	1.279e-10	1.322e-05

Table 2.5. *Residual norms*

meth.	1138bus	steam1 240	nos7 729	fs 183 6	bcsstk20 485
dgetf2 np	3.316e-11	1.537e-08	1.488e-08	8.955e-08	9.002
dgetf2	3.261e-11	3.828e-08	1.488e-08	3.406e-08	8.018
dgetrf	2.102e-11	3.493e-08	1.609e-08	3.354e-08	4.134
dgjsv	1.068e-10	5.616e-07	3.005e-08	1.108e-07	1.302e+06
dgjsvr	1.325e-10	5.732e-08	3.005e-08	1.676e-08	13.43
dgbjsvr	5.499e-09	7.587e-08	9.422e-06	5.083e-08	7.559e+12
dghsvr	9.048e-11	2.624e-08	2.630e-08	6.830e-08	7.963
dgbhsvr	3.499e-11	2.100e-08	1.079e-08	6.839e-08	4.18
dpurcell	1.088e-10	8.687e-08	3.269e-08	7.309e-08	10.71
dBM	1.071e-10	2.692e-08	2.887e-08	1.528e-08	9.548
ZW	1.230e-06	6.532e-08	0.12190	0.02468	9.312e+07
WZ np	2.398e-11	4.179e-08	1.813e-08	3.933e-09	4.67
WZ	3.108e-11	6.497e-08	1.550e-08	2.984e-08	3.106

meth.	mcfe 765	nnc 261	lnsp 511
dgetf2 np	733.6	NaN	NaN
dgetf2	97.26	2.151e-12	3.351e-06
dgetrf	102.5	2.599e-12	3.073e-06
dgjsv	153.9	0.0001435	0.0004175
dgjsvr	52.2	1.481e-12	3.687e-05
dgbjsvr	130.2	0.003508	0.003324
dghsvr	75.88	1.632e-12	3.525e-05
dgbhsvr	77.8	7.349e-13	1.845e-05
dpurcell	97.33	1.977e-12	4.654e-05
dBM	76.09	1.374e-12	2.859e-05
ZW	650.7	NaN	NaN
WZ np	75.2	NaN	NaN
WZ	106.1	1.876e-12	8.292e-06

Table 2.6. *Relative residual norms* $\|b - Ax\|/\|b\|$

meth.	pde225	gre 243	jpwh 991	jagmesh1	bfwa782 782
dgetf2 np	2.858e-16	1.377e-13	6.126e-16	NaN	1.910e-15
dgetf2	2.858e-16	5.676e-16	6.141e-16	1.025e-15	1.012e-15
dgetrf	2.872e-16	6.856e-16	5.581e-16	1.056e-15	8.591e-16
dgjsv	5.663e-16	1.216e-15	1.054e-15	2.775e-15	1.956e-15
dgjsvr	5.663e-16	1.145e-15	1.054e-15	3.231e-15	1.278e-15
dgbjsvr	3.576e-16	1.109e-15	4.061e-16	6.003e-15	1.416e-15
dghsvr	4.896e-16	1.010e-15	1.062e-15	3.157e-15	9.729e-16
dgbhsvr	2.497e-16	7.529e-16	3.104e-16	2.321e-15	5.588e-16
dpurcell	6.131e-16	6.975e-16	1.018e-15	2.277e-15	1.564e-15
dBM	5.124e-16	1.096e-15	1.067e-15	4.245e-15	8.786e-16
ZW	1.276e-15	NaN	1.473e-15	NaN	5.100e-14
WZ np	2.704e-16	5.140e-13	6.002e-16	NaN	7.779e-16
WZ	2.700e-16	8.962e-16	6.756e-16	1.458e-15	7.158e-16

meth.	jagmesh2 1009	lshp1009	fs 680 1c	fs 680 1	sherman1 1000
dgetf2 np	NaN	NaN	4.162e-16	8.723e-16	4.052e-16
dgetf2	1.198e-15	1.198e-15	5.091e-16	8.723e-16	3.911e-16
dgetrf	1.254e-15	1.254e-15	4.711e-16	8.345e-16	3.050e-16
dgjsv	3.529e-15	3.529e-15	1.375e-14	6.211e-16	1.826e-15
dgjsvr	3.302e-15	3.302e-15	5.244e-16	6.211e-16	8.306e-16
dgbjsvr	7.059e-15	7.059e-15	1.276e-14	8.793e-16	2.622e-15
dghsvr	3.403e-15	3.403e-15	2.553e-16	4.512e-16	6.793e-16
dgbhsvr	2.335e-15	2.335e-15	1.619e-16	2.987e-16	3.143e-16
dpurcell	2.589e-15	2.589e-15	3.658e-16	7.445e-16	9.843e-16
dBM	4.027e-15	4.027e-15	2.648e-16	4.903e-16	7.448e-16
ZW	NaN	NaN	4.524e-14	5.560e-15	2.419e-14
WZ np	NaN	NaN	4.489e-16	8.061e-16	3.339e-16
WZ	1.244e-15	1.244e-15	4.409e-16	8.061e-16	3.197e-16

meth.	nos3 960	cavity05 1182	e05r0500 236	olm1000	steam2 600
dgetf2 np	4.637e-16	7.725e-16	1.638e-14	1.258e-16	2.131e-16
dgetf2	4.637e-16	2.439e-15	7.815e-16	6.870e-17	2.193e-16
dgetrf	4.272e-16	2.287e-15	7.912e-16	6.924e-17	2.045e-16
dgjsv	1.308e-15	4.693e-14	3.485e-14	1.652e-13	2.843e-16
dgjsvr	1.308e-15	1.767e-15	7.726e-16	2.576e-15	2.765e-16
dgbjsvr	3.008e-15	4.067e-13	6.488e-12	3.280e-12	2.226e-16
dghsvr	1.179e-15	1.133e-15	5.722e-16	2.643e-15	3.175e-16
dgbhsvr	4.194e-16	6.260e-16	4.480e-16	1.023e-15	2.117e-16
dpurcell	1.265e-15	1.157e-15	5.314e-16	3.292e-16	4.776e-16
dBM	1.230e-15	1.161e-15	5.332e-16	6.732e-16	3.305e-16
ZW	5.579e-14	1.425e-12	NaN	NaN	4.644e-16
WZ np	5.003e-16	NaN	NaN	1.324e-16	2.584e-16
WZ	5.074e-16	1.742e-15	5.434e-16	1.072e-16	2.460e-16

Table 2.7. *Relative residual norms* $\|b - Ax\|/\|b\|$

meth.	1138bus	steam1 240	nos7 729	fs 183 6	bcsstk20 485
dgetf2 np	3.082e-16	1.738e-16	4.989e-16	2.165e-16	4.319e-16
dgetf2	3.031e-16	4.331e-16	4.989e-16	8.233e-17	3.847e-16
dgetrf	1.954e-16	3.952e-16	5.392e-16	8.107e-17	1.984e-16
dgjsv	9.931e-16	6.354e-15	1.007e-15	2.678e-16	6.245e-11
dgjsvr	1.231e-15	6.485e-16	1.007e-15	4.050e-17	6.446e-16
dgbjsvr	5.111e-14	8.583e-16	3.158e-13	1.229e-16	0.0003627
dghsvr	8.409e-16	2.969e-16	8.813e-16	1.651e-16	3.820e-16
dgbhsvr	3.252e-16	2.375e-16	3.616e-16	1.653e-16	2.005e-16
dpurcell	1.011e-15	9.827e-16	1.096e-15	1.767e-16	5.136e-16
dBM	9.956e-16	3.045e-16	9.675e-16	3.692e-17	4.581e-16
ZW	1.143e-11	7.389e-16	4.084e-09	5.967e-11	4.468e-09
WZ np	2.229e-16	4.728e-16	6.078e-16	9.506e-18	2.241e-16
WZ	2.889e-16	7.350e-16	5.193e-16	7.212e-17	1.490e-16

meth.	mcfe 765	nnc 261	lnsp 511
dgetf2 np	2.339e-15	NaN	NaN
dgetf2	3.101e-16	4.932e-16	3.549e-17
dgetrf	3.269e-16	5.957e-16	3.255e-17
dgjsv	4.908e-16	3.290e-08	4.421e-15
dgjsvr	1.664e-16	3.396e-16	3.904e-16
dgbjsvr	4.151e-16	8.043e-07	3.520e-14
dghsvr	2.419e-16	3.741e-16	3.733e-16
dgbhsvr	2.481e-16	1.685e-16	1.954e-16
dpurcell	3.104e-16	4.532e-16	4.929e-16
dBM	2.426e-16	3.151e-16	3.028e-16
ZW	2.075e-15	NaN	NaN
WZ np	2.398e-16	NaN	NaN
WZ	3.383e-16	4.302e-16	8.781e-17

Table 2.8. *Error norms*

meth.	pde225	gre 243	jpwh 991	jagmesh1	bfwa782 782
dgetf2 np	8.581e-15	5.006e-12	4.025e-14	NaN	1.241e-12
dgetf2	8.581e-15	2.648e-14	4.024e-14	6.356e-13	5.074e-13
dgetrf	9.027e-15	3.410e-14	3.406e-14	5.874e-13	4.900e-13
dgjsv	1.165e-14	4.386e-14	4.878e-14	7.278e-13	3.654e-13
dgjsvr	1.165e-14	3.832e-14	4.870e-14	9.122e-13	6.968e-13
dgbjsvr	8.007e-15	4.607e-14	2.795e-14	6.265e-13	1.601e-13
dghsvr	8.844e-15	3.453e-14	3.517e-14	6.274e-13	6.549e-13
dgbhsvr	5.792e-15	4.760e-14	1.680e-14	4.490e-13	2.112e-13
dpurcell	1.240e-14	2.392e-14	3.371e-14	5.733e-13	4.868e-13
dBM	8.918e-15	3.037e-14	3.380e-14	1.189e-12	4.671e-13
ZW	2.352e-14	NaN	4.518e-14	NaN	2.180e-11
WZ np	8.555e-15	4.491e-11	4.258e-14	NaN	3.987e-13
WZ	8.401e-15	5.375e-14	5.982e-14	7.844e-13	3.642e-13

meth.	jagmesh2 1009	lshp1009	fs 680 1c	fs 680 1	sherman1 1000
dgetf2 np	NaN	NaN	1.476e-12	1.085e-12	8.609e-13
dgetf2	4.492e-13	4.492e-13	2.378e-12	1.085e-12	1.172e-12
dgetrf	4.646e-13	4.646e-13	2.044e-12	8.356e-13	9.382e-13
dgjsv	8.750e-13	8.750e-13	1.654e-12	1.065e-12	1.213e-12
dgjsvr	9.243e-13	9.243e-13	1.563e-12	1.065e-12	1.163e-12
dgbjsvr	5.689e-13	5.689e-13	1.597e-12	7.874e-13	2.458e-11
dghsvr	3.797e-13	3.797e-13	2.907e-13	1.943e-13	1.934e-13
dgbhsvr	5.915e-13	5.915e-13	3.216e-13	1.880e-13	1.799e-13
dpurcell	5.919e-13	5.919e-13	4.004e-13	2.822e-13	4.149e-13
dBM	5.156e-13	5.156e-13	2.543e-13	2.167e-13	1.433e-13
ZW	NaN	NaN	1.652e-12	1.406e-12	1.020e-12
WZ np	NaN	NaN	1.471e-12	1.023e-12	5.701e-13
WZ	3.007e-13	3.007e-13	1.161e-12	1.023e-12	1.267e-12

meth.	nos3 960	cavity05 1182	e05r0500 236	olm1000	steam2 600
dgetf2 np	8.573e-12	2.495e-12	5.840e-10	2.779e-11	9.162e-12
dgetf2	8.573e-12	2.918e-10	1.086e-10	3.029e-11	9.719e-12
dgetrf	6.502e-12	1.740e-10	1.081e-10	2.865e-11	9.687e-12
dgjsv	7.708e-12	5.098e-10	1.192e-10	3.526e-11	9.692e-12
dgjsvr	7.708e-12	3.027e-12	8.114e-12	2.042e-11	9.436e-12
dgbjsvr	3.122e-12	1.308e-09	8.736e-09	3.113e-08	8.357e-12
dghsvr	9.809e-13	4.189e-12	3.944e-12	1.327e-11	6.472e-12
dgbhsvr	3.244e-12	3.942e-12	3.282e-12	2.738e-11	6.797e-12
dpurcell	2.089e-13	3.029e-12	2.819e-12	2.259e-11	6.313e-12
dBM	2.742e-12	1.410e-12	2.092e-12	4.509e-11	6.312e-12
ZW	6.243e-12	9.625e-10	NaN	NaN	7.044e-12
WZ np	3.975e-12	NaN	NaN	2.853e-11	1.514e-11
WZ	1.078e-11	1.576e-10	4.384e-11	3.618e-11	1.147e-11

Table 2.9. *Error norms*

meth.	1138bus	steam1 240	nos7 729	fs 183 6	bcsstk20 485
dgetf2 np	4.553e-10	3.150e-12	8.812e-08	8.730e-07	2.352e-07
dgetf2	4.551e-10	4.080e-09	8.812e-08	1.459e-07	7.515e-07
dgetrf	1.570e-10	2.893e-09	3.182e-08	1.880e-07	8.707e-07
dgjsv	4.412e-10	3.025e-09	5.389e-08	4.542e-07	8.953e-07
dgjsvr	4.142e-10	3.322e-12	5.389e-08	1.965e-07	1.087e-07
dgbjsvr	7.261e-11	3.430e-09	7.103e-08	1.942e-07	2.62
dghsvr	5.757e-10	1.930e-12	1.925e-08	2.270e-07	3.691e-07
dgbhsvr	1.320e-11	2.488e-12	1.412e-08	2.131e-07	1.628e-07
dpurcell	6.383e-10	1.813e-12	3.330e-08	8.132e-08	1.374e-06
dBM	6.903e-10	2.048e-12	2.600e-08	8.431e-08	3.506e-07
ZW	9.319e-10	3.499e-12	1.163e-07	0.1336	0.005807
WZ np	2.619e-10	1.276e-11	2.229e-08	1.515e-07	4.756e-07
WZ	1.303e-10	7.907e-09	1.168e-08	3.934e-07	1.511e-06

meth.	mcfe 765	nnc 261	lnsp 511
dgetf2 np	5.521e-12	NaN	NaN
dgetf2	1.610e-12	0.003847	6.204e-09
dgetrf	1.280e-12	0.008257	4.01e-09
dgjsv	6.330e-13	0.01552	4.916e-09
dgjsvr	3.108e-14	0.008658	4.050e-10
dgbjsvr	1.786e-12	0.2182	3.197e-09
dghsvr	2.968e-14	0.002956	8.295e-12
dgbhsvr	1.866e-14	0.0019	4.954e-11
dpurcell	3.291e-14	0.001081	1.761e-11
dBM	2.961e-14	0.001872	8.370e-11
ZW	2.650e-13	NaN	NaN
WZ np	4.632e-12	NaN	NaN
WZ	8.916e-13	0.01116	1.263e-09

Table 2.10. *Relative error norms* $\|x_e - x\|/\|x_e\|$

meth.	pde225	gre 243	jpwh 991	jagmesh1	bfwa782 782
dgetf2 np	5.371e-16	2.568e-13	1.280e-15	NaN	4.481e-14
dgetf2	5.371e-16	1.359e-15	1.280e-15	2.092e-14	1.832e-14
dgetrf	5.650e-16	1.749e-15	1.084e-15	1.933e-14	1.769e-14
dgjsv	7.292e-16	2.250e-15	1.552e-15	2.395e-14	1.319e-14
dgjsvr	7.292e-16	1.966e-15	1.549e-15	3.002e-14	2.515e-14
dgbjsvr	5.012e-16	2.364e-15	8.892e-16	2.062e-14	5.778e-15
dghsvr	5.536e-16	1.772e-15	1.119e-15	2.065e-14	2.364e-14
dgbhsvr	3.625e-16	2.442e-15	5.344e-16	1.478e-14	7.622e-15
dpurcell	7.762e-16	1.227e-15	1.072e-15	1.887e-14	1.757e-14
dBM	5.582e-16	1.558e-15	1.075e-15	3.912e-14	1.686e-14
ZW	1.472e-15	NaN	1.437e-15	NaN	7.867e-13
WZ np	5.355e-16	2.304e-12	1.355e-15	NaN	1.439e-14
WZ	5.258e-16	2.758e-15	1.903e-15	2.582e-14	1.314e-14

meth.	jagmesh2 1009	lshp1009	fs 680 1c	fs 680 1	sherman1 1000
dgetf2 np	NaN	NaN	5.672e-14	4.169e-14	2.725e-14
dgetf2	1.419e-14	1.419e-14	9.136e-14	4.169e-14	3.711e-14
dgetrf	1.468e-14	1.468e-14	7.856e-14	3.211e-14	2.970e-14
dgjsv	2.764e-14	2.764e-14	6.356e-14	4.093e-14	3.840e-14
dgjsvr	2.920e-14	2.920e-14	6.007e-14	4.093e-14	3.681e-14
dgbjsvr	1.797e-14	1.797e-14	6.135e-14	3.026e-14	7.782e-13
dghsvr	1.200e-14	1.200e-14	1.117e-14	7.468e-15	6.121e-15
dgbhsvr	1.869e-14	1.869e-14	1.236e-14	7.223e-15	5.694e-15
dpurcell	1.870e-14	1.870e-14	1.539e-14	1.084e-14	1.313e-14
dBM	1.629e-14	1.629e-14	9.773e-15	8.327e-15	4.536e-15
ZW	NaN	NaN	6.349e-14	5.401e-14	3.228e-14
WZ np	NaN	NaN	5.653e-14	3.930e-14	1.805e-14
WZ	9.500e-15	9.500e-15	4.463e-14	3.930e-14	4.012e-14

meth.	nos3 960	cavity05 1182	e05r0500 236	olm1000	steam2 600
dgetf2 np	2.783e-13	7.217e-14	3.626e-11	8.798e-13	3.757e-13
dgetf2	2.783e-13	8.441e-12	6.741e-12	9.587e-13	3.986e-13
dgetrf	2.111e-13	5.035e-12	6.713e-12	9.068e-13	3.973e-13
dgjsv	2.502e-13	1.475e-11	7.404e-12	1.116e-12	3.975e-13
dgjsvr	2.502e-13	8.756e-14	5.038e-13	6.462e-13	3.870e-13
dgbjsvr	1.014e-13	3.784e-11	5.425e-10	9.855e-10	3.427e-13
dghsvr	3.185e-14	1.212e-13	2.449e-13	4.200e-13	2.654e-13
dgbhsvr	1.053e-13	1.140e-13	2.038e-13	8.667e-13	2.787e-13
dpurcell	6.781e-15	8.763e-14	1.751e-13	7.151e-13	2.589e-13
dBM	8.902e-14	4.079e-14	1.299e-13	1.427e-12	2.588e-13
ZW	2.027e-13	2.784e-11	NaN	NaN	2.889e-13
WZ np	1.290e-13	NaN	NaN	9.032e-13	6.207e-13
WZ	3.499e-13	4.560e-12	2.722e-12	1.145e-12	4.705e-13

Table 2.11. *Relative error norms* $\|x_e - x\|/\|x_e\|$

meth.	1138bus	steam1 240	nos7 729	fs 183 6	bcsstk20 485
dgetf2 np	1.344e-11	1.916e-13	3.309e-09	5.792e-08	1.059e-08
dgetf2	1.343e-11	2.482e-10	3.309e-09	9.68e-09	3.384e-08
dgetrf	4.636e-12	1.760e-10	1.195e-09	1.247e-08	3.920e-08
dgjsv	1.302e-11	1.840e-10	2.024e-09	3.013e-08	4.031e-08
dgjsvr	1.223e-11	2.021e-13	2.024e-09	1.304e-08	4.893e-09
dgbjsvr	2.143e-12	2.086e-10	2.667e-09	1.289e-08	0.1179
dghsvr	1.699e-11	1.174e-13	7.229e-10	1.506e-08	1.662e-08
dgbhsvr	3.897e-13	1.513e-13	5.302e-10	1.414e-08	7.328e-09
dpurcell	1.884e-11	1.103e-13	1.251e-09	5.395e-09	6.187e-08
dBM	2.038e-11	1.246e-13	9.764e-10	5.594e-09	1.578e-08
ZW	2.751e-11	2.129e-13	4.367e-09	0.008866	0.0002614
WZ np	7.730e-12	7.761e-13	8.370e-10	1.005e-08	2.141e-08
WZ	3.845e-12	4.810e-10	4.387e-10	2.61e-08	6.802e-08

meth.	mcfe 765	nnc 261	lnsp 511
dgetf2 np	2.007e-13	NaN	NaN
dgetf2	5.853e-14	0.0002245	2.725e-10
dgetrf	4.655e-14	0.0004818	1.762e-10
dgjsv	2.302e-14	0.0009055	2.160e-10
dgjsvr	1.130e-15	0.0005052	1.779e-11
dgbjsvr	6.493e-14	0.01273	1.404e-10
dghsvr	1.079e-15	0.0001725	3.644e-13
dgbhsvr	6.782e-16	0.0001109	2.176e-12
dpurcell	1.197e-15	6.309e-05	7.738e-13
dBM	1.077e-15	0.0001092	3.677e-12
ZW	9.635e-15	NaN	NaN
WZ np	1.684e-13	NaN	NaN
WZ	3.242e-14	0.0006511	5.55e-11

2.19 ▪ Historical and bibliographical comments

It has been written [558, 559] that the earliest solutions of linear equations were obtained by the Babylonians (2000 to 1600 BC) in Mesopotamia, the region between the Tigris and Euphrates rivers. The Babylonians were using a positional sexagesimal number system and fixed point numbers. They represented problems by a step-by-step list of rules whose evaluation was given in words. It turns out that problems written on some clay tablets correspond, when translated in our modern notation, to solving linear and nonlinear equations. However, seeing the Babylonians being the first ever people solving linear equations and inventing linear algebra may be simply an overstatement, see [650, 651].

As we said above, elimination methods were used in ancient China long before our era. The book in which we find these methods is the Chinese mathematical classic *Jiuzhang Suanshu* (which has been translated as *The Nine Chapters on Mathematical Art*). It is believed to have been compiled some time between 100 BC and 100 AD, but it is likely that the content of that book was much older than its date of compilation. Chapter 8 contains 18 practical problems which amount to solve nonsingular linear systems of order 2 to 5 and one underdetermined system with 5 equations and 6 unknowns. The procedure in described in words using examples. There is no general description. The Chinese mathematicians of those times were solving small linear systems with integer coefficients using an elimination method. Their method is not exactly what we now call Gaussian elimination, particularly in the backward phase, but it is very similar. They did not use pivoting techniques and symbols for the unknowns and did not have the concept of linear equation.

Solution by elimination of small linear systems of order 3 or 4 were done in Western Europe in the 15th and 16th centuries. An important step in the history of algebra occurred at the end of the 16th and the beginning of the 17th centuries. It was the advent of symbolic algebra, thanks to the works of François Viète (1540-1603) and René Descartes (1596-1650). During the 17th and 18th centuries many books contained examples of elimination methods on small linear systems, including one by Isaac Newton (1643-1727) published in the first years of the 18th century, and one by Leonhard Euler (1707-1783) published in 1768.

The general solution of a nonsingular linear system of any order using ratios of determinants was given by the Swiss mathematician Gabriel Cramer (1704-1754) in an appendix of his book [290] about algebraic curves published in 1750. It could have been the end of elimination methods, but is was not so because computing determinants of large order is painful by hand and costly when using any computing device.

The interest of Johann Carl Friedrich Gauss (1777-1855) for the method of least squares came from his activities in astronomy and geodesy. He had to solve what was considered as large linear systems in those times. He gave the details of his method in [498] in 1809. Gauss improved upon his predecessors by describing the elimination process in very general terms. Contrary to what was done before, he did not rewrite the equations after each elimination step, but just computed the coefficients with his own bracket notation. Since Gauss solved linear systems by elimination in a way clearly different from his predecessors, it seems to be fair to name the method we are using today "Gaussian elimination", even though the basic technique is much, much older.

In 1888, Wilhelm Jordan (1842-1899) published *Handbuch der Vermessungskunde* (Surveyor's manual), a book [687] on geodesy, in which he showed how to use elimination (that he called *Gauss'sche elimination*), but with a slightly different notation. It is thanks to that book that this method spread out [17]. The Gauss-Jordan elimination was devised by this Jordan, and not, as it is sometimes said, by the French mathematician Marie Ennemond Camille Jordan (1838-1922) whose name is associated with the Jordan canonical form.

A method similar to Gauss-Jordan was devised by the abbot Bernard Isidore Clasen (1829-

1902), from Luxembourg, in 1888 [268].

Interesting variants of Gaussian elimination were devised by Myrick Hascall Doolittle (1830-1913) in 1878 [354], André Louis Cholesky (1875-1918) around 1910, and Prescott Durand Crout (1907-1984) in 1941 [291]. It is interesting to observe that Doolittle used iterative refinement to improve the solutions he computed by hand using multiplication tables. Cholesky never published his method during his lifetime. It was published by a fellow officer, Major Ernest Benoît (1873-1956) in 1924 [110]. About Cholesky's life and work, see [171]. Cholesky's method was later independently rediscovered many times; in particular, in the square root method of Tadeusz Banachiewicz (1882-1954), a Polish astronomer and mathematician in 1938 [93].

When the first computers were developed after World War II, it was not obvious to everyone that Gaussian elimination could be used safely for computing solution of linear systems, see [1105]. In particular, Harold Hotelling (1895-1973), an American statistician and economist raised some concerns about its accuracy because of rounding errors. In 1947, John von Neumann (1903-1957) published a paper [1105] with Herman H. Goldstine (1913-2004) about the inversion of matrices in which they did an analysis of Gaussian elimination. In 1948, Alan Mathison Turing (1912-1954) published a paper titled "Rounding-off errors in matrix processes" [1078]. Together with the von Neumann and Goldstine paper, it has been influential in rehabilitating Gaussian elimination.

It is mainly through the work of James Hardy Wilkinson (1919-1986), summarized in his 1965 book [1120], that the properties of Gaussian elimination were carefully studied and it became widely known that Gaussian elimination can be successfully used. Wilkinson also developed some software to be used on the early computers available at that time. For a summary of the work on the stability of Gaussian elimination, see N.J. Higham [631, 633].

The Gauss-Huard method was proposed in 1979 by Pierre Huard de la Marre (1927-2022) who was an expert in optimization [655]. He was professor in Lille University (France) and scientific advisor for EDF, the French electricity company. He is well known for his "méthode des centres" (center method) in optimization.

The WZ factorization was devised and mainly promoted by David John Evans (1928-2005). He was professor at Loughborough University (UK).

Most of the papers published today about Gaussian elimination are concerned with the implementation of the method on parallel computers. For details on the history of Gaussian elimination, see [556, 557, 558] and [160].

3
Gaussian elimination for sparse linear systems

The matrix may be sparse, either with the non-zero elements concentrated ... or distributed in a less systematic manner. We shall refer to a matrix as dense if the percentage of zero elements or its distribution is such as to make it uneconomic to take advantage of their presence.

– J.H. Wilkinson and C. Reinsch [1121] 1971

There are many practical problems leading to linear systems with a matrix having many zero entries. Matrices of that type are called *sparse matrices*. There is no precise definition of what is a sparse matrix, that is, how many zeros entries there are or what is the percentage of zeros, see the quote above. As we have seen in Chapter 1, special techniques are used to store sparse matrices in order to store only the nonzero entries, as well as avoiding operations on zeros. The matrix and its factors can be stored by rows, by columns, or by blocks. As for dense matrices there are several variants of Gaussian elimination for sparse matrices. The storage scheme is generally adapted to the particular variant that is used. A definition that has sometimes been given is that a matrix is considered as sparse when it is beneficial (either in computer storage, that is, memory usage, or in computer time) to use special sparse techniques as opposed to the more traditional general algorithms we have described in Chapter 2. Exploiting sparsity allows to compute the solution of very large problems.

There are a few good books about direct methods for sparse linear systems. Let us mention those of J.A. George and J.W.H. Liu [511] for symmetric positive definite systems, S. Pissanetsky [920], T.A. Davis [312], I.S. Duff, A.M. Erisman and J.K. Reid [373], and J.S. Scott and M. Tůma [1002] for more general sparse systems. There is also a thorough recent review of sparse techniques by T.A. Davis, S. Rajamanickamn and W.M. Sid-Lakhdar [315].

3.1 ▪ Triangular systems

When factorizing a sparse matrix, the factors L and U are generally sparse, and the last step of the algorithm is solving two sparse triangular systems. Let us consider a sparse lower triangular matrix. The algorithm to choose depends on the storage scheme for the sparse factor L and also on the right-hand side being dense or sparse. Generally, things are not too different from the dense case. However, a sparse triangular solve is difficult to parallelize. The entries of L give a dependency graph which is a direct acyclic graph (DAG) that can be used to schedule the computation.

Early algorithms for parallel computers were described by A. George, M.T. Heath, J.W.H. Liu, and E.G. Ng [504, 505, 506], and E. Rothberg [958]. More recent references are P.R. Amestoy, I.S. Duff, A. Guermouche, and T. Slavova [28] and E. Totoni, M.T. Heath, and L.V. Kale [1070].

Another possibilities are to use inversion of submatrices as E. Anderson and Y. Saad [35] or a partition form of the inverse as F.L. Alvarado, D.C. Yu, and R. Betancourt [20], F.L. Alvarado and R.S. Schreiber [19], A. Pothen and F.L. Alvarado [925], F.L. Alvarado, A. Pothen, and R.S. Schreiber [18], and B.W. Peyton, A. Pothen, and X. Yuan [916].

3.2 ▪ The fill-in phenomenon

Unfortunately, in many cases, there are more nonzero entries in the L and U factors than in the sparse matrix A. Let us assume that there exists an LU factorization of the matrix A. As we have seen in Chapter 2, the entries $a_{i,j}^{(k+1)}$ of the reduced matrix at the kth step of Gaussian elimination are computed as

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)} a_{k,j}^{(k)}}{a_{k,k}^{(k)}}.$$

Even if $a_{i,j}^{(k)} = 0$ in the previous reduced matrix, $a_{i,j}^{(k+1)}$ can be nonzero if $a_{i,k}^{(k)} \neq 0$ and $a_{k,j}^{(k)} \neq 0$. Nonzero entries in the L and U factors in positions (i, j) for which $a_{i,j} = 0$ are called *fill-ins*. The storage scheme for L and U must be designed to account for that.

Let us consider an example with a small matrix of order 5 having a symmetric nonzero structure. The x's in A denote nonzero entries,

$$A_1 = A = \begin{pmatrix} x & x & 0 & x & 0 \\ x & x & x & 0 & 0 \\ 0 & x & x & 0 & x \\ x & 0 & 0 & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix}.$$

Let us look at the successive steps of the LU factorization, assuming that we do not need to use pivoting. Fill-ins are denoted by black bullets •,

$$A_2 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & x & x & 0 & x \\ 0 & \bullet & 0 & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix}.$$

The entry $(2, 4)$, which was initially zero, becomes nonzero because the entries $(2, 1)$ and $(1, 4)$ are nonzero. The same thing happens for the entry $(4, 2)$ because the nonzero structure is symmetric. The second step gives

$$A_3 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & \bullet & x & 0 \\ 0 & 0 & x & 0 & x \end{pmatrix}.$$

We observe that the fill-in in position $(4, 3)$ is created by the fill-in in position $(4, 2)$ at the

previous step. The next steps are

$$A_4 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & 0 & x & \bullet \\ 0 & 0 & 0 & \bullet & x \end{pmatrix}, \quad U = A_5 = \begin{pmatrix} x & x & 0 & x & 0 \\ 0 & x & x & \bullet & 0 \\ 0 & 0 & x & \bullet & x \\ 0 & 0 & 0 & x & \bullet \\ 0 & 0 & 0 & 0 & x \end{pmatrix}.$$

There are four fill-ins in A_4 but only three in A_5 which is the U matrix. Finally, the L factor is

$$L = \begin{pmatrix} x & & & & \\ x & x & & & \\ 0 & x & x & & \\ x & \bullet & \bullet & x & \\ 0 & 0 & x & \bullet & x \end{pmatrix}.$$

In this example, three entries which were initially zero in the lower triangular part of A are nonzero in L. When pivoting is used, the number of fill-ins depends on the choice of the pivots. Different pivoting strategies may lead to large differences in the number of fill-ins. Let us consider a well-known example,

$$A = \begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix} \quad \Rightarrow \quad L = \begin{pmatrix} x & & & \\ x & x & & \\ x & \bullet & x & \\ x & \bullet & \bullet & x \end{pmatrix}.$$

All the zero entries in the lower triangular part of A are filled in L. We can define a permutation matrix P such that the first element is numbered last, giving the permuted matrix,

$$PAP^T = \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & x & x \end{pmatrix}.$$

There is no fill-in in the LU factorization of PAP^T. This is called a *perfect elimination*.

The aim of sparse Gaussian elimination is to avoid doing operations on zero entries and therefore to try to decrease as much as possible the number of fill-ins. This will have the effect of decreasing the needed storage and the number of floating-point operations. The way this can be achieved depends on the properties of the matrix A. If the matrix is symmetric and, for instance, positive definite, we do not need to pivot for numerical stability, see Chapter 2. This gives the freedom to choose symmetric permutations only to decrease the fill-in. Moreover, the number and positions of fill-ins can be determined before doing the numerical factorization since it depends only on the structure of the matrix and not on the values of the entries. The LU factorization can be computed within a static data structure that is constructed in a preprocessing phase called the *symbolic factorization*. It has been shown that finding an ordering that minimizes the fill-in is an NP complete problem, see M. Yannakakis [1137]. Consequently, all the algorithms to reduce the fill-in rely on heuristics to find a "good" ordering producing a low level of fill-in. When the matrix is nonsymmetric and without any special properties, we have seen that we generally need to pivot to achieve an acceptable numerical accuracy. If in addition the matrix is sparse, we now have another requirement which is to decrease the fill-in. These two, sometimes conflicting, goals have to be dealt with at the same time. It implies that the data structure for the L and U factors cannot be determined before the numerical factorization, since the pivot rows or columns, and therefore the potential fill-ins are only known when performing the numerical factorization.

3.3 ▪ Graphs and fill-in for symmetric matrices

To set up the data structure for the triangular matrix L in a Cholesky factorization $A = LL^T$ we have to know the positions of the fill-ins. This was the topic of intense research, starting in the 1970s. It is based on the interpretation of Gaussian elimination in terms of graphs.

We have seen in Chapter 1 that a graph can be associated with any sparse matrix. It turns out that Gaussian elimination can be described by simple operations on the graph. The connection between sparse Gaussian elimination and graphs for symmetric matrices was first studied by S.V. Parter [906] in the 1960s, see also D.J. Rose [951]. We define a sequence of graphs $G^{(k)}, k = 1, \ldots, n$, corresponding to the successive steps of the elimination, $G^{(1)} = G$ being the graph of the matrix A.

Theorem 3.1. *The graph $G^{(k+1)}$ is obtained from $G^{(k)}$ by removing the node x_k from the graph as well as all its incident edges and adding edges such that all the remaining neighbors of x_k in $G^{(k)}$ are pairwise connected. This process from $G^{(k)}$ to $G^{(k+1)}$ corresponds to the kth step of Gaussian elimination.*

Proof. Let us prove this for the first step, eliminating the node x_1 (or the corresponding unknown in the linear system). Then,

$$a_{i,j}^{(2)} = a_{i,j} - \frac{a_{i,1}a_{1,j}}{a_{1,1}}.$$

The element $a_{i,j}^{(2)}$ is nonzero if either $a_{i,j} \neq 0$ or $a_{i,j} = 0$ and $a_{i,1}$ and $a_{1,j}$ are nonzero. This last possibility means that x_i and x_j are neighbors of x_1 in the graph. When x_1 is eliminated and $a_{i,j}^{(2)} \neq 0$, they will be connected by an edge representing the new nonzero entry. This occurs for all the neighbors of x_1. We do not consider zeros that arise by cancellation in the computation of $a_{i,j}^{(2)}$. Therefore, $G^{(2)}$ is the graph corresponding to the submatrix obtained from $A^{(2)}$, by deleting the first row and the first column. A similar process obviously occurs for every step of Gaussian elimination. □

Starting from the graph $G(A)$ of A and adding the edges that are created in all the $G^{(k)}$'s during the elimination, we obtain a graph $G_F, (F = L + L^T)$ which is called the *filled graph* $G_F = (X, E^F)$. Let us consider a small example of an elimination graph for the structurally symmetric matrix, Let

$$A = \begin{pmatrix} x & x & x & x & & & x \\ x & x & & & & & \\ x & & x & x & & & \\ x & & x & x & x & x & x \\ & & & x & x & & \\ & & & x & & x & x \\ x & & & x & & x & x \end{pmatrix}.$$

Figure 3.1 displays the graph $G(A)$. The graph $G^{(2)}$ is given in Figure 3.2. Eliminating x_1, we have to pairwise connect its neighbors x_2, x_3, x_4 and x_7. The edges corresponding to fill-ins are shown as grey lines.

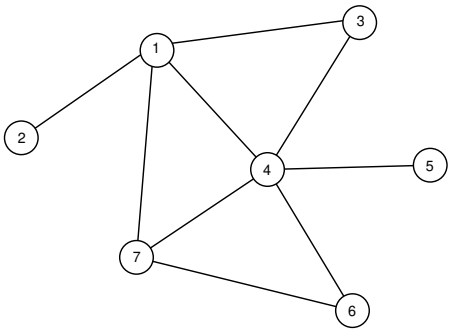


Figure 3.1. *The graph* $G(A)$

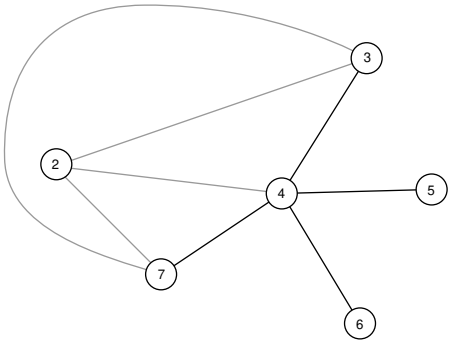


Figure 3.2. *The graph* $G^{(2)}$

The graph $G^{(2)}$ corresponds to the following matrix, without the first row and first column,

$$A^{(2)} = \begin{pmatrix} x & x & x & x & & & x \\ x & x & \bullet & \bullet & & & \bullet \\ x & \bullet & x & x & & & \bullet \\ x & \bullet & x & x & x & x & x \\ & & & x & x & & \\ & & & x & & x & x \\ x & \bullet & \bullet & x & & x & x \end{pmatrix}.$$

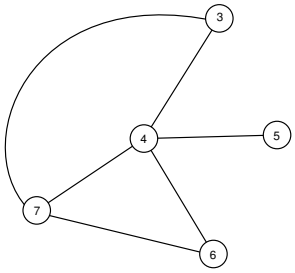


Figure 3.3. *The graph* $G^{(3)}$

Then, the elimination of x_2 does not cause any fill-in since all its neighbors form already a clique. $G^{(3)}$ is given in Figure 3.3 and $A^{(3)} \equiv A^{(2)}$, $\equiv$ meaning that the two matrices have the same structure. The next step is to eliminate x_3. Again, there is no fill-in since x_4 and x_7 are already connected. $G^{(4)}$ is displayed on Figure 3.4 and $A^{(4)} \equiv A^{(3)}$.

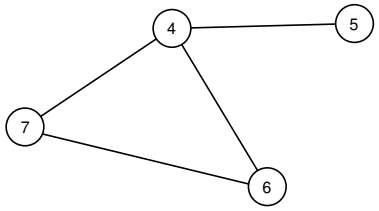


Figure 3.4. *The graph $G^{(4)}$*

Elimination of x_4 connects x_5, x_6 and x_7. The graph $G^{(5)}$ is shown on Figure 3.5 and the matrix is

$$A^{(5)} = \begin{pmatrix} x & x & x & x & & & x \\ x & x & \bullet & \bullet & & & \bullet \\ x & \bullet & x & x & & & \bullet \\ x & \bullet & x & x & x & x & x \\ & & & x & x & \bullet & \bullet \\ & & & x & \bullet & x & x \\ x & \bullet & \bullet & x & \bullet & x & x \end{pmatrix}.$$

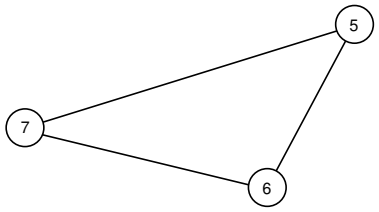


Figure 3.5. *The graph $G^{(5)}$*

$G^{(5)}$ is a clique and the corresponding 3×3 submatrix is dense thus there will be no other fill-in before the end of the elimination and $A^{(7)} \equiv A^{(5)}$. The filled graph G_F is shown in Figure 3.6.

In total there are six fill-ins in the elimination. We observe that a perfect elimination can be obtained by ordering the unknowns as

$$2, \quad 5, \quad 3, \quad 6, \quad 1, \quad 4, \quad 7$$

With this ordering, the permuted matrix is

$$A' = PAP^T = \begin{pmatrix} x & & & & x & & \\ & x & & & & x & \\ & & x & & x & x & \\ & & & x & & x & x \\ x & & x & & x & x & x \\ & x & x & x & x & x & x \\ & & & x & x & x & x \end{pmatrix}.$$

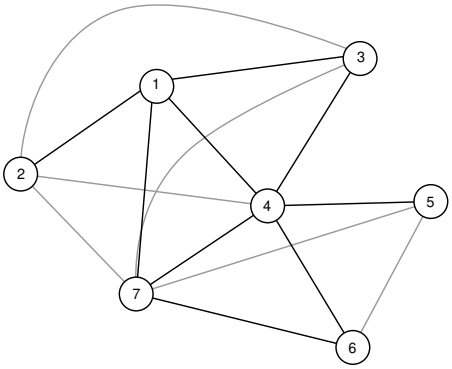


Figure 3.6. *The filled graph G_F*

One thing to remember is that the more fill-ins we create in the early stages of the elimination, the more fill-ins we will get later on, since fill-ins usually create fill-ins. The number of fill-ins depends of the order of elimination, that is, on the numbering of the vertices in the graph. From this remark, an heuristic rule is that it is likely to be beneficial to start eliminating nodes (or unknowns) that do not create many fill-ins. These are the nodes with a small number of neighbors or nodes in cliques.

Instead of reducing the fill-in one may try to minimize (or decrease) the number of floating-point operations. This is different from the minimum fill-in problem. R. Luce and E.G. Ng [768] showed that this problem is NP-hard for the sparse Cholesky factorization.

3.4 ▪ Characterization of the fill-in

To characterize the fill-in it is useful to introduce a few more definitions.

• The *elimination tree* of a symmetric matrix A of order n is a graph with n nodes such that the node p is the parent of node j if and only if

$$p = \min\{i \,|\, i > j,\, \ell_{i,j} \neq 0\}$$

where L is the Cholesky factor of $A = LL^T$, when this factorization exists. The elimination tree of A is denoted by $T(A)$ or simply T if the context makes it clear that we refer to the matrix A. Clearly, p is the index of the first nonzero entry below the diagonal in column j of L. In the example of Figure 3.1, $T(A')$ is shown in Figure 3.7 with a renumbering of the unknowns according to P.

This tree shows that x'_1, x'_2, x'_3, x'_4 (corresponding to x_2, x_5, x_3, x_6 in the initial ordering) can be eliminated in any order (or even in parallel) since there are no dependencies between the corresponding variables.

The elimination tree was introduced by R.S. Schreiber [998]. It can be computed before knowing the nonzero pattern of L, and then used to construct that pattern, see [998], and J.W.H. Liu [757, 760]. Let T_k be the tree corresponding to the principal submatrix $A_{1:k,1:k}$. The tree T_k is constructed from T_{k-1}. For each entry $a_{k,i} \neq 0$, it is enough to see if i is a descendant of k in T_k. Following the path from i to a root t in T_{k-1} implies that t is a child of k in T_k. If this is well implemented, the run time to construct T is almost linear in the number of nonzero entries in A.

Quite often, the nodes of the elimination tree are postordered. In a postordered tree, the d proper descendants of a node k are numbered $k - d$ through $k - 1$.

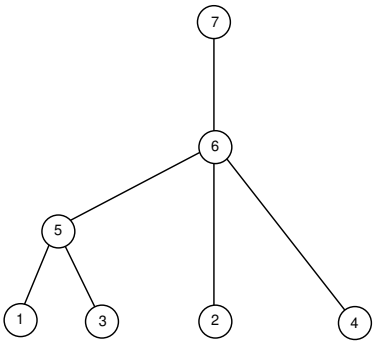


Figure 3.7. *The elimination tree of the matrix A'*

• Let $S \subset X$ and $x \in X, x \notin S$, x is said to be *reachable* from $y \notin S$ through S if there exists a path $(y, v_1, \ldots, v_k, x)$ from y to x in G such that $v_i \in S$, $i = 1, \ldots, k$. We define the set of nodes that can be reached from y as

$$Reach(y, S) = \{x \,|\, x \notin S, x \text{ is reachable from } y \text{ through } S\}$$

There can be a fill-in between x_j and x_k only if at some step m, they are not already connected together and both neighbors of a node $x_m, m < j$, $m < k$. Either they were already neighbors of x_m in G or they were put in this situation by the elimination of other nodes $x_\ell, \ell < m$. Recursively, we see that at some stage, x_j was a neighbor of one of these nodes and the same for x_k with another of these nodes. It means that in G, there is at least one path between x_j and x_k and that all the nodes on this path have numbers smaller than j and k. If there is no such path, there could not be a fill-in between x_j and x_k. To formalize this we first prove a lemma due to S.V. Parter [906].

Lemma 3.2. *The edge $\{x_i, x_j\} \in E^F$ if and only if $\{x_i, x_j\} \in E$ or $\{x_i, x_k\} \in E^F$ and $\{x_k, x_j\} \in E^F$ for some $k < \min\{i, j\}$.*

Proof. If $\{x_i, x_k\}$ is an edge of the filled graph, that is, $\in E^F$ and $\{x_k, x_j\} \in E^F$ for some $k < \min\{i, j\}$, the elimination of x_k creates a fill-in between x_i and x_j. Therefore, $\{x_i, x_j\} \in E^F$. Conversely, if $\{x_i, x_j\} \in E^F$ and $\{x_i, x_j\} \notin E$, then at some stage, x_i and x_j must be neighbors of a node, say x_k, that will be eliminated before x_i and x_j. Thus, $k < \min\{i, j\}$. □

The fill-in was characterized by A. George, see the book by A. George and J.W.H. Liu [511]. An earlier reference is [953] by D.J. Rose, R.E. Tarjan, and G.S. Lueker.

Theorem 3.3. *Let $k > j$, there will be a fill-in between x_j and x_k if and only if*

$$x_k \in Reach(x_j, \{x_1, \ldots, x_{j-1}\}).$$

Proof. Assume $x_k \in Reach(x_j, \{x_1, \ldots, x_{j-1}\})$. There exists a path $\{x_j, v_1, \ldots, v_\ell, x_k\} \in G$ with $v_i \in \{x_1, \ldots, x_{j-1}\}$, $1 \leq i \leq \ell$. If $\ell = 0$ or $\ell = 1$, the result follows from Lemma 3.2. If $\ell > 1$, we can show that $\{x_k, x_j\} \in E^F$ by induction.

Conversely, we assume $\{x_i, x_j\} \in E^F$, $j < k$. The proof is by induction on j. For $j = 1$, $\{x_1, x_k\} \in E^F$ implies $\{x_1, x_k\} \in E$ since there is no fill-in with the first node. Moreover, the

set $\{x_1, \ldots, x_{j-1}\}$ is empty. Assume the result is true up to $j-1$. From Lemma 3.2, there exists some $\ell \leq j-1$ such that $\{x_j, x_\ell\} \in E^F$ and $\{x_\ell, x_k\} \in E^F$. By assumption, there exists a path between x_j and x_ℓ and another one from x_ℓ to x_k. Clearly, this implies that there is a path from x_j to x_k whose nodes have numbers $\leq \ell \leq j-1$. □

A. George demonstrated that reachable sets can be efficiently implemented by using quotient graphs [511] that were introduced by S.C. Eisenstat, M.H. Schultz, and A.H. Sherman [412].

The fill-in can also be characterized using elimination trees. Let $T[x]$ be the subtree of $T(A)$ rooted at node x. A node $y \in T[x]$ is a *descendant* of x and x is an *ancestor* of y. From the definition of $T(A)$, if x_i is a proper ancestor of x_j in $T(A)$, then $i > j$. The following result is due to J.W.H. Liu [760].

Theorem 3.4. *For $i > j$, the numerical values of columns i of L (denoted $L_{:,i}$) depend on column j of L ($L_{:,j}$) if and only if $\ell_{i,j} \neq 0$.*

□

The row and column counts of the lower triangular matrix L can also be found from the elimination tree without computing the pattern of L. They are useful to construct a data structure for L. For the row count, see J.W.H. Liu [757], J.R. Gilbert, E.G. Ng, and B.W. Peyton [530], and J.R. Gilbert, X.S. Li, E.G. Ng, and B.W. Peyton [526]. For the column count, see A. George and J.W.H. Liu [511], R.S. Schreiber [998], and J.D. Hogg and J.A. Scott [646].

A. Pothen and S. Toledo wrote a survey [927] of algorithms and data structures for the symbolic analysis of both symmetric and nonsymmetric factorizations.

After the symbolic analysis, as for the dense case, there are several variants for the numerical factorization. Let us mention the up-looking or bordering method which proceeds one row at a time, see D.J. Rose, G.G. Whitten, A.H. Sherman, and R.E. Tarjan [954], J.W.H. Liu [757], R.E. Bank and R.K. Smith [94], and T.A. Davis [311, 312].

The left-looking method proceeds one column at a time, see S.C. Eisenstat, M.H. Schultz, and A.H. Sherman [411, 414], S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman [404], A. George and J.W.H. Liu [511], and A. George, M.T. Heath, J.W.H. Liu, and E.G. Ng [504].

The right-looking (fan-out or outer product) method modifies the trailing submatrix, see A. George, M.T. Heath, J.W.H. Liu, and E.G. Ng [505].

3.5 ▪ Band and envelope numbering schemes for symmetric matrices

The first attempts to exploit sparsity in the 1960s and 1970s used band or envelope storage schemes, trying to reduce the storage. These methods are now only of historical interest. But profile reduction orderings are still an active area of research since they are very well-suited for the frontal methods we will discuss later. Let us introduce a few definitions.

• $f_i(A) = \min\{i \,|\, a_{i,j} \neq 0\}$. $f_i(A)$ is the index of the column with the first nonzero entry of row i.

• $\beta_i(A) = i - f_i(A)$ is the *bandwidth* of row i. The bandwidth of the matrix A is defined as

$$\beta(A) = \max_i\{\beta_i(A),\, 1 \leq i \leq n\}, \quad band(A) = \{(i,j) \,|\, 0 < i-j \leq \beta(A),\, i \geq j\}.$$

• $Env(A) = \{(i,j) \,|\, 0 < i-j \leq \beta_i(A),\, i \geq j\}$ is the *envelope* of A. The *profile* of A is defined

as

$$P_r(A) = |Env(A)| = \sum_{i=1}^{n} \beta_i(A).$$

In the 1970s these definitions led to ideas for storing the matrices A and L since if $\beta_i(A)$ is almost constant as a function of i, it makes sense to store the entries corresponding to all the indices in $band(A)$. Unfortunately, most of the time this is not practical since there are often a few rows with a large bandwidth and then, too much storage is wasted by the band scheme. Then, one can use the variable band or *envelope storage scheme*, see A. Jennings [678]. For all rows, this simple storage scheme stores all the entries of the envelope in one vector. Another vector of integers is needed to point to the start of each row. The interest in this storage scheme was motivated by the following result.

Theorem 3.5. *Let* $Fill(A) = \{(i,j) \,|\, i > j,\ a_{i,j} = 0,\ \ell_{i,j} \neq 0\}$ *be the index set of the fill-ins, then*

$$Fill(A) \subset Env(A).$$

Proof. This is a direct consequence of Theorem 3.3 since there cannot be any fill-in from a node x_i to a node x_j whose number is smaller than the smallest number of the neighbors of x_i. All the paths going from x_i to x_j will have a node with a number larger than x_j. □

When using these storage schemes, it was natural to try to find orderings that reduce the bandwidth or the profile of the matrix. Unfortunately, minimizing the bandwidth or the profile is an NP-complete problem, see C.H. Papadimitriou [903]. But there are some heuristics that help to obtain low profile orderings.

3.5.1 ▪ The Cuthill-McKee and reverse Cuthill-McKee orderings

The Cuthill-McKee (`CM`) algorithm is a local minimization algorithm whose aim is to reduce the profile of A. Clearly, if at some stage of the ordering process, we want to minimize $\beta_i(A)$, we must immediately number all the unlabeled nodes in $Adj(x_i)$. The algorithm due to E.H. Cuthill and J. McKee [301] is the following.

Algorithm Cuthill-McKee (CM)
1) choose a starting node,
2) for $i = 1, \ldots, n-1$ number all the (unlabeled) neighbors of x_i in $G(A)$ in increasing order of degree,
3) update the degrees of the remaining nodes.

The profile resulting from this ordering is quite sensitive to the choice of the starting node.

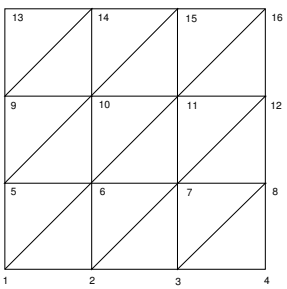


Figure 3.8. *An example of ordering*

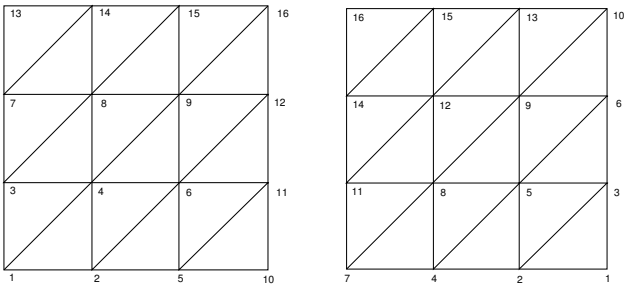


Figure 3.9. *Two reorderings*

Consider the graph shown on Figure 3.8. If we choose 1 as a starting node, we obtain the left ordering of Figure 3.9 with 38 fill-ins in L. If node 4 of the initial ordering is chosen as a starting node, the right ordering is obtained with 14 fill-ins in L. In the left ordering, the maximum difference of node numbers between neighbors is 7. With the right one, the maximum difference is 4.

Let us consider the level structures of both orderings. The level structure of the left ordering is on the left of Figure 3.10 using the numbers of the initial graph to label the nodes. The height of the structure is 4 and the width 7. The level structure of the second ordering is shown on the right. Its height is 7 and its width is 4. Clearly, the second choice is better than the first one. The higher the structure is, the narrower it is. Starting nodes which give rise to narrow level structures are clearly better choices.

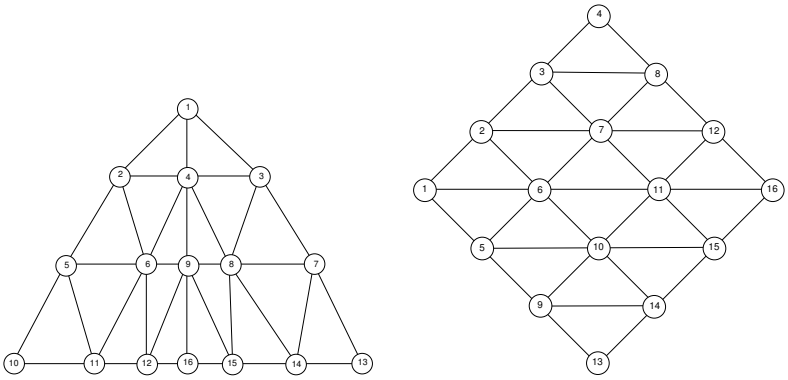


Figure 3.10. *The two level structures*

A "good" choice for a starting node will be a *peripheral* node, that is, one whose eccentricity is equal to the diameter of the graph. This gives a narrow level structure where the difference in number for a node and its neighbors is minimal. Unfortunately, peripheral nodes are not that easy to find quickly. Therefore, heuristics were devised to find *pseudo-peripheral* nodes, that is, nodes whose eccentricities are close to the diameter of the graph. An algorithm for finding such nodes was proposed by N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer [524]. It is the following.

Algorithm GPS

1) choose a starting node r,

2) build the level structure $\mathcal{L}(r)$

$$\mathcal{L}(r) = \{L_0(r), \ldots, L_{e(r)}(r)\},$$

3) sort the nodes $x \in L_{e(r)}(r)$ by increasing degree order,
4) for all nodes $x \in L_{e(r)}(r)$ in increasing degree order, build $\mathcal{L}(x)$. If the height of $\mathcal{L}(x)$ is larger than the height of $\mathcal{L}(r)$, choose x as a starting node $(r = x)$ and go to step 2).

This algorithm eventually converges since eccentricities are bounded above by the diameter of the graph. However, it can be very costly. A. George and J.W.H. Liu [508] decreased the computing time by eliminating structures with large width as soon as possible. Step 4) of the algorithm is modified as,

$4'$) let $w(x)$ be the width of $\mathcal{L}(x)$. For all $x \in L_{e(r)}(r)$ in order of increasing degree, build

$$\mathcal{L}(x) = \{L_0(x), \ldots, L_{e(r)}(x)\}.$$

At each level i, if $|L_i(x)| > w(r)$, drop the current node and pick another one x. If $w(x) \leq w(r)$ and $e(x) > e(r)$, choose x as a starting node $(r = x)$ and go to 2).

A. George and J.W.H. Liu [511] also proposed to use the following simple algorithm,

1) choose a starting node r,
2) build $\mathcal{L}(r)$,
3) choose a node x of minimum degree in $L_{e(r)}(r)$,
4) build $\mathcal{L}(x)$. If $e(x) > e(r)$, choose x as a starting node and go to 2).

As an ordering scheme, A. George and J.W.H. Liu [511] proposed to reverse the Cuthill-McKee ordering.

Algorithm Reverse Cuthill-McKee (RCM),
1) find a pseudo-peripheral starting node,
2) generate the `CM` ordering,
3) reverse the numbering. Let $x_1, \ldots, x_n$ be the `CM` ordering. The `RCM` ordering $\{y_i\}$ is given by $y_i = x_{n+i-1}$, $i = 1, \ldots, n$.

Let us consider a small example. We would like to number the graph of Figure 3.11.

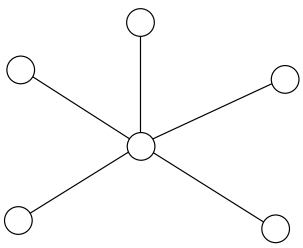


Figure 3.11. *A graph to number*

With the Cuthill-McKee (resp. reverse Cuthill-McKee) algorithm, we obtain the left (resp. right) part of Figure 3.12.

The nonzero structures of the two Cholesky factors are respectively,

$$L_{CM} = \begin{pmatrix} x & & & & & \\ x & x & & & & \\ & x & x & & & \\ & x & \bullet & x & & \\ & x & \bullet & \bullet & x & \\ & x & \bullet & \bullet & \bullet & x \end{pmatrix}, \quad L_{RCM} = \begin{pmatrix} x & & & & & \\ & x & & & & \\ & & x & & & \\ & & & x & & \\ x & x & x & x & x & \\ & & & & x & x \end{pmatrix}.$$

There are 6 fill-ins with CM and no fill-in with RCM.

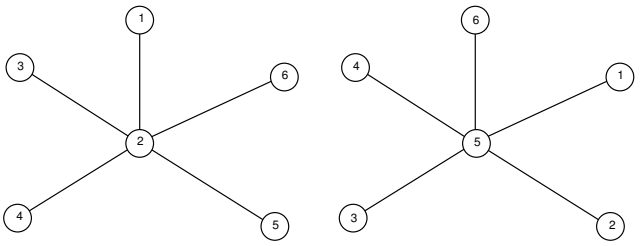


Figure 3.12. *Cuthill-McKee (left) and reverse Cuthill-McKee (right)*

Let us now show that, regarding the number of fill-ins, `RCM` is always as good as `CM`. Hence, there is no reason to use `CM`.

Theorem 3.6. *Let A be an irreducible matrix and A_{CM} be the matrix corresponding to reordering (the graph of) A by the Cuthill-McKee scheme. Then,*

$$\forall i, j,\ i \leq j,\ f_i \leq f_j.$$

Moreover, $f_i < i$ if $i > 1$.

Proof. Assume the conclusion does not hold. Then, there are a column k and rows $p, \ell, m,\ p < \ell < m$ such that

$$f_p \leq k, \quad f_\ell > k, \quad f_m \leq k.$$

It means that

$$\begin{aligned} a_{p,k} \neq 0 &\Longrightarrow x_p \in Adj(x_k), \\ a_{m,k} \neq 0 &\Longrightarrow x_m \in Adj(x_k), \\ a_{\ell,k} = 0 &\Longrightarrow x_\ell \notin Adj(x_k). \end{aligned}$$

This is a contradiction since the Cuthill-McKee algorithm has successively numbered all nodes in $Adj(x_k)$. □

Let us introduce a new definition,

• $Tenv(A) = \{(i,j) \mid j \leq i,\ \exists k \geq i,\ a_{k,j} \neq 0\}$.

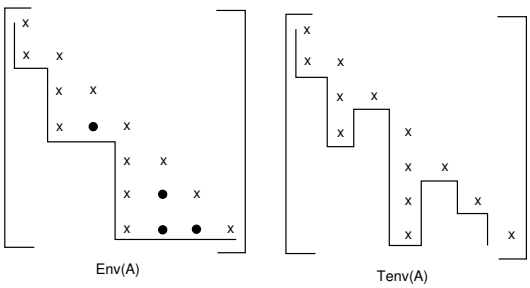


Figure 3.13. *Env(A) and Tenv(A)*

$Tenv(A)$ is the *transpose envelope* of A. Let us consider the example in Figure 3.13. If we use the reverse Cuthill-McKee algorithm, we have to reverse the ordering, and we obtain the matrix of Figure 3.14. The rows of A_{RCM} are the columns of A_{CM}.

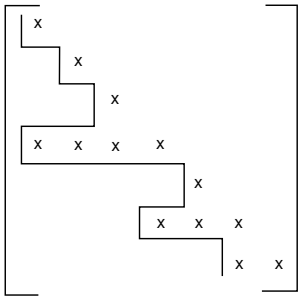


Figure 3.14. *The envelope for RCM*

Lemma 3.7.

$$|Env(A_{RCM})| = |Tenv(A_{CM})|.$$

Proof. Straightforward. □

Theorem 3.8.

$$Tenv(A_{CM}) \subseteq Env(A_{CM}).$$

Proof. Looking at the figures, the result seems obvious, but let us formalize it. Assume we have $(i,j) \in Tenv(A_{CM})$ and $(i,j) \notin Env(A_{CM})$. Then, there exists $k \geq i$ such that $a_{k,j} \neq 0$. Either

1) $a_{i,j} \neq 0 \Longrightarrow (i,j) \in Env(A_{CM})$,

or

2) $a_{i,j} = 0$. If $(i,j) \notin Env(A_{CM}) \Longrightarrow \forall \ell \leq j,\ a_{i,\ell} = 0 \Longrightarrow f_i > j$.

But, we have $f_k \leq j$. It implies $f_k < f_i$ which is impossible since $k \geq i$ by Theorem 3.6. □

Obviously, we have

$$|Env(A_{RCM})| \leq |Env(A_{CM})|.$$

A. George proved the following result.

Lemma 3.9. *If $\forall i > 1,\ f_i < i$, the envelope $Env(A)$ fills completely.*

□

This result implies

$$|Fill(A_{RCM})| \leq |Fill(A_{CM})|.$$

There are cases for which equality holds. It was shown that `RCM` can be implemented to run in $O(|E|)$ time. For a regular $N \times N$ grid and triangular finite elements, the storage for `RCM` varies as $O(N^3)$, ($\approx 0.7N^3$) that is $O(n^{\frac{3}{2}})$.

Figure 3.15 displays the nonzero structure of two matrices. The matrix on the left arises from the discretization of the Poisson model problem on a 6×6 square mesh with a lexicographic ordering. The matrix on the right is a "Wathen" matrix from N.J. Higham's matrix toolbox. The matrix A is the "consistent mass matrix" for a regular 10×10 grid of 8-node (serendipity) elements in two space dimensions. Figure 3.16 shows the nonzero structure of those matrices reordered by the Matlab `RCM` algorithm. For the Wathen matrix, the number of nonzero entries in the Cholesky factor is 2311 for the initial ordering and 2141 with `RCM`.

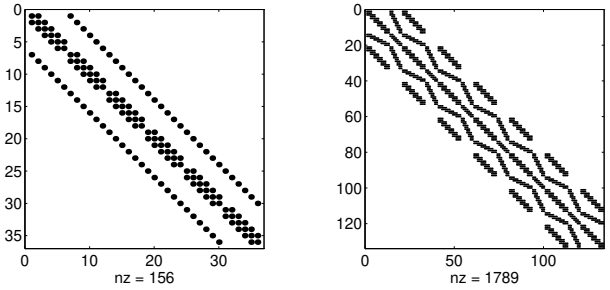


Figure 3.15. *The nonzero structure of the Poisson (left) and Wathen (right) matrices*

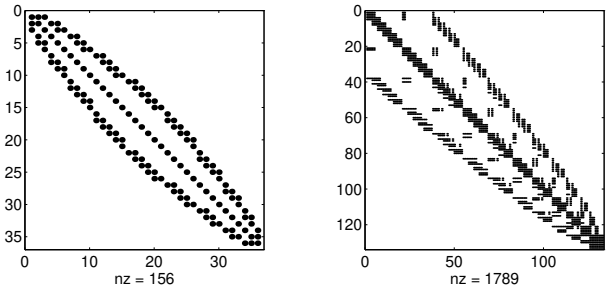


Figure 3.16. *The nonzero structure of the Poisson and Wathen matrices reordered by* RCM

Several other algorithms have been proposed to reduce the profile of a symmetric matrix. I.P. King [704] proposed a wavefront reduction algorithm. R. Levy's algorithm [732] is similar to King's algorithm, but, at each stage, all vertices are considered instead of unlabeled vertices adjacent to already labeled vertices.

E.H. Cuthill [300] compared the Cuthill-McKee algorithm with its reverse form, and I.P. King's method using a number of criteria such as bandwidth, wavefront and profile reduction. Her results showed that the CM method and RCM gave smaller bandwidths. R. Levy's algorithm gave smaller wavefronts and profiles.

Other algorithms were proposed by R.A. Snay [1028] and N.E. Gibbs [523]. G.C. Everstine [433] compared RCM, GPS, and R. Levy's algorithm on different criteria and concluded that GPS is good for maximum wavefront and profile reduction. J.G. Lewis ([733, 734] described techniques to improve GPS and the algorithm of N.E. Gibbs. W.W. Hager [608] considered exchanges of rows and columns to minimize the profile. This is useful to refine other orderings.

3.5.2 ▪ Sloan's algorithm

One of the downsides of the CM algorithm is that it is a local algorithm only using information about the neighbors of the last numbered nodes. S.W. Sloan [1021] suggested an algorithm that tries to overcome this problem. The first step is the selection of pseudo-peripheral nodes.

Sloan's algorithm

• Step 1 (selection of a pseudo-diameter)

1) choose a node s with minimum degree,

2) build the level structure $\mathcal{L}(s) = \{L_0(s), \ldots, L_k(s)\}$,

3) sort the nodes of $L_k(s)$ by increasing degree. Let m be the number of entries in $L_k(s)$ and Q be the $\lfloor \frac{m+2}{2} \rfloor$ first entries of the sorted set,

4) Let $w_{\min} = \infty$ and $k_{\max} = k$. For each node $i \in Q$ in order of ascending degree, generate $\mathcal{L}(i) = \{L_0(i), \ldots, L_k(i)\}$. If $k > k_{\max}$ and $w = \max_{i \le j \le k} |L_j(i)| < w_{\min}$, set $s = i$ and go to step 3). Otherwise, if $w < w_{\min}$, set $e = i$ and $w_{\min} = w$.

At the end of this algorithm we have a starting node s and an end node e which define a pseudo-diameter. The difference of this algorithm with GPS is the shrinking strategy of step 3). This is performed since it was observed that nodes with large degrees are not often chosen as starting nodes. The second step of Sloan's algorithm labels the nodes.

• Step 2 (node labeling)

The nodes are classified in four categories according to their status. Nodes which have been already assigned a label are *postactive*. Nodes which have not been assigned a number but are adjacent to postactive nodes are *active*. Nodes without a number adjacent to active nodes are *preactive*. All other nodes are *inactive*. The current degree n_i of a node i is defined as $n_i = m_i - c_i + k_i$, where m_i is the degree of i, c_i is the number of postactive or active nodes adjacent to i, and $k_i = 0$ if i is active or postactive, and $k_i = 1$ otherwise. The inputs of the algorithm are the two nodes s and e selected in Step 1). The algorithm maintains a list of eligible nodes each with a priority related to the current degree and the distance from the end node. Nodes with low current degree and large distance to the end have high priorities.

1) for all nodes, compute the distances $d(e, i)$ from i to e, initialize all nodes as inactive and set

$$P_i = (n_{\max} - n_i)W_1 + d(e, i)W_2,$$

where $n_{max} = \max_i n_i$, and W_1, W_2 are integer weights. The queue of eligible nodes is initialized with s which is assigned a preactive status.

2) as long as the queue is not empty,

2.1) select the node i with highest priority in the queue (ties are broken arbitrarily),

2.2) delete i from the queue. If it is not preactive, go to 2.3). Else, consider each node j adjacent to i and set $P_j = P_j + W_1$. If j is inactive, insert j in the queue and declare it preactive,

2.3) label node i and declare it postactive,

2.4) Examine every node j adjacent to i. If j is preactive, set $P_j = P_j + W_1$, declare j as active and examine each node k adjacent to j. If k is active or preactive, set $P_k = P_k + W_1$, otherwise if k is inactive, set $P_k = P_k + W_1$, insert k in the queue and declare it as preactive.

Values of the weights W_1 and W_2 determine the balance between the local information (the current degree) and the global one (the distance) to the end. S.W. Sloan [1021] chose $W_1 = 2$ and $W_2 = 1$. He reported numerical experiments showing that on certain sets of matrices his algorithm gives lower profiles than RCM and other algorithms. Figure 3.17 gives the nonzero structures for our two previous examples with the Sloan algorithm. For the Wathen matrix, the number of nonzero entries in the Cholesky factor is 1921, that is, slightly less than with RCM. For the Poisson matrix both RCM and Sloan give the same profile.

I.S. Duff, J.K. Reid, and J.A. Scott [384] have improved Sloan's algorithm by allowing it to work with weighted graphs where nodes with the same adjacency set are collapsed. G. Kumfert and A. Pothen [715] have also used the Sloan algorithm combined with other techniques.

3.6 ▪ Spectral schemes

The *Laplacian matrix* $\mathcal{L}(G)$ associated with a symmetric matrix A (or the associated graph G) is such that if $i \neq j$, $\mathcal{L}_{i,j} = -1$ if node j is a neighbor of node i in the graph G (that is, if $a_{i,j} \neq 0$). The diagonal entry $\mathcal{L}_{i,i}$ is minus the sum of the other entries in row i. The matrix $\mathcal{L}$ is a singular M-matrix, and its smallest eigenvalue λ_1 is zero. The eigenvector u_2 corresponding

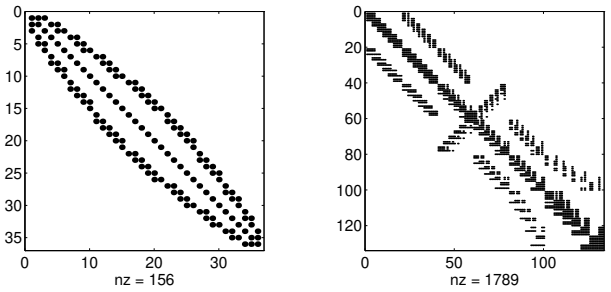


Figure 3.17. *The nonzero structure of the Poisson and Whathen matrices reordered by Sloan's algorithm*

to the smallest positive eigenvalue λ_2 has interesting properties that were studied by M. Fiedler [450].

The Laplacian matrix has been used both for envelope reduction, and for graph partitioning. The rationale for using u_2 in envelope reducing algorithms is the following. Let $row(i) = \{j \,|\, a_{i,j} \neq 0, 1 \leq j \leq i\}$. We define the 1-sum σ_1 as,

$$\sigma_1(A) = \sum_{i=1}^{n} \sum_{j \in row(i)} (i - j).$$

The work in the Cholesky factorization is proportional to

$$W = \sum_{i=1}^{n} \max_{j \in row(i)} (i - j)^2.$$

This is related to the 2-sum,

$$\sigma_2^2(A) = \sum_{i=1}^{n} \sum_{j \in row(i)} (i - j)^2.$$

The following theorem is due to A. George and A. Pothen [517].

Theorem 3.10. *Let p be the maximum number of off-diagonal nonzero entries in a row of A (or the maximum node degree in G). Then*

$$W \leq \sigma_2^2(A) \leq pW.$$

□

The spectral ordering sorts the components of the eigenvector u_2 in increasing order. It gives a permutation vector which represents the ordering. To justify this, we consider $\sigma_2(A)$ instead of W as the quantity to minimize over all the orderings. Assume, for the sake of simplicity, that n is even and let $\mathcal{P}$ be the set of vectors whose components are permutations of $\{-n/2, \ldots, -1, 0, 1, \ldots, n/2\}$. Then,

$$\min_{x \in \mathcal{P}} \sum_{i=1}^{n} \sum_{j \in row(i)} (x_i - x_j)^2 = \frac{1}{2} \min_{x \in \mathcal{P}} \sum_{a_{i,j} \neq 0} (x_i - x_j)^2.$$

The condition on x is relaxed to obtain an easier continuous problem. We consider the set $\mathcal{H}$ of vectors x such that $\sum x_i = 0$ and (x, x) given. We have

$$\frac{1}{2} \min_{x \in \mathcal{H}} \sum_{a_{i,j} \neq 0} (x_i - x_j)^2 = \min_{x \in \mathcal{H}} (x, \mathcal{L}x) = \lambda_2(u_2, u_2).$$

S.T. Barnard, A. Pothen, and H.D. Simon [98] showed that the permutation vector induced by u_2 is the closest vector in $\mathcal{P}$ to u_2 in the Euclidean norm. Figure 3.18 shows the results of the reordering for the two previous examples. For the Wathen matrix the number of nonzero entries in the Cholesky factor is slightly larger than before, being 2571.

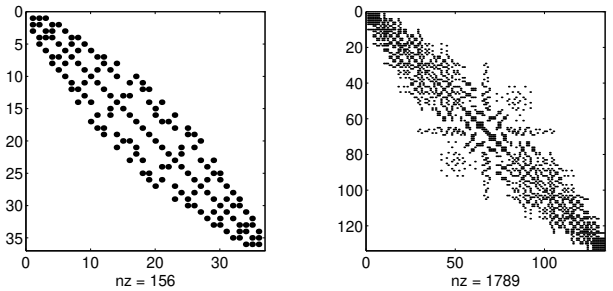


Figure 3.18. *The nonzero structure of the Poisson and Wathen matrices reordered by the spectral ordering*

The eigenvalue λ_2 and eigenvector u_2 do not have to be computed very accurately, but the previous spectral algorithm is too costly compared to other profile reduction algorithms for large examples. To decrease the cost, S.T. Barnard, A. Pothen, and H.D. Simon proposed using the algorithm on a coarsened graph with much fewer vertices than G as follows,

1) construct a series of coarser and coarser graphs that retains the structure of the original graph,
2) compute the second eigenvector of the Laplacian matrix of the coarsest graph,
3) interpolate this vector to the next finer graph,
4) refine the interpolated vector (by Rayleigh Quotient Iteration, see B.N. Parlett [905]) and go to 3) until we are back to the original graph.

There are many ways to define graph coarsening. The one chosen by S.T. Barnard, A. Pothen, and H.D. Simon was to find a maximal independent set of vertices which are to be the vertices of the coarser graph. The edges are found by growing domains from the selected vertices adding an edge when two domains intersect. Good results were reported in [98]. A similar algorithm was introduced independently by G.H. Paulino, I.F. Menezes, M. Gattass, and S. Mukherjee [907].

As we said above, G. Kumfert and A. Pothen [715] suggested using a combination of the spectral and the Sloan algorithms. The multilevel spectral algorithm is used to find the end nodes of a pseudo-diameter, and then a modified version of the Sloan algorithm is used to number the nodes. G. Kumfert and A. Pothen provided examples for which the spectral algorithm performs poorly. They introduced a variant of the Sloan algorithm for weighted graphs. They denoted by $Size(i)$ the weight of a multi-vertex i. The degree of that node is the sum of the sizes of the neighboring multi-vertices. The generalization of the current degree $Cdeg(i)$ denotes the sum of the sizes of the neighbors of i for preactive or inactive vertices. Then, they defined

$$Incr(i) = \begin{cases} Cdeg(i) + Size(i) & \text{if i is preactive,} \\ Cdeg(i) & \text{if i is active.} \end{cases}$$

Let Δ be the maximum degree in the unweighted graph, the new priority function is defined as

$$P(i) = -W_1 \lfloor d(s,e)/\Delta \rfloor Incr(i) + W_2\, d(i,e).$$

Considering the choices of weights, G. Kumfert and A. Pothen identified two classes of problems, one for which $W_1 = 2$ and $W_2 = 1$ gives good results, and another one that needs large values of W_2 to obtain small envelopes. They implemented the priority queue as a binary heap contrary to Sloan's implementation as an array. It turns out that this new implementation was much faster on selected sets of examples. In the hybrid algorithm, the start and end nodes are chosen to be the first and last nodes in the spectral ordering, and the priority function is given by

$$P(i) = -W_1 \lfloor n/\Delta \rfloor Incr(i) + W_2\, d(i,e) - W_3\, i,$$

this function being sensitive to the initial ordering through the third term. The weight W_3 is chosen by considering the eigenvector corresponding to the first nonzero eigenvalue. If the component giving the maximum of the absolute values of the components is negative, then $W_3 = -1$ and the starting and end nodes are exchanged, otherwise $W_3 = 1$. The other weights are $W_1 = W_2 = 1$. However, there are problems for which $W_1 = 1, W_2 = W_3 = 2$ give good results. Numerical results showed that the hybrid algorithm gives better envelope sizes than both `RCM` and the Sloan algorithm.

In the approach of E.G. Boman and B. Hendrickson [142], the original problem is approximated by a sequence of coarser ones. The vertices of the coarsest problem are then labeled and the results are interpolated back to the larger graphs with some refinement steps if needed. The coarsening of graphs is done by edge contraction by coalescing adjacent vertices and assigning weights to the edges of the coarse graph. If two vertices are adjacent to the same neighbor, the new edge is given a weight equal to the sum of the two old edges. In the first step, a maximum matching is found. This is a maximal set of edges for which no two edges are incident to the same vertex. The coarsest graph is numbered by the spectral ordering algorithm. The uncoarsening is done by numbering the vertices of the larger graph preserving the coarse graph ordering. A local refinement scheme is used to improve this ordering. It is a variant of the algorithm of B.W. Kernighan and S. Lin [697], the weighted 1-sum being the objective function.

3.7 ▪ The minimum degree ordering

This ordering scheme was introduced by W.F. Tinney and J.W. Walker [1062]. With some of its variants, it is one of the most often used ordering schemes today. Its aim is to locally minimize the fill-in. The *minimum degree* (`MD`) uses the elimination graphs $G^{(i)} = (X_i, E_i)$ corresponding to Gaussian elimination. The ith step of the algorithm is described as follows,

Algorithm MD

1) in $G^{(i)}$, find a node x_j such that

$$deg(x_j) = \min_{y \in X_i} \{deg(y)\},$$

and number it at the ith node.

2) form $G^{(i+1)}$ by eliminating x_j and update the degrees,

3) if $i + 1 < n$ go to 1) with $i \leftarrow i + 1$.

To use this algorithm we must have a way to represent the elimination graphs and to transform them. Of course, the degree of some nodes change after the deletion of edges incident to x_i and the addition of new edges. Quite often, there are several nodes of minimum degree. This must be

resolved by a tie breaking strategy. Unfortunately, the final number of fill-ins is quite sensitive to the tie breaking strategy, see A. George and J.W.H. Liu [512].

Since it is a local minimization algorithm, the minimum degree does not always give a globally minimum fill-in ordering. There are cases, like trees, for which it gives no fill-in at all, but there are examples for which it generates fill-in that is more than a constant time greater than the minimum fill-in, see P. Berman and G. Schnitger [124] for examples.

The first efficient implementation was proposed by A. George and D.R. McIntyre [513]; see also J.W. Huang and O. Wing [654], A. George and J.W.H. Liu [510, 509] used respectively quotient graphs and reachable sets in their implementations.

Over the years many improvements were suggested to the basic algorithm, mainly to decrease the computer time needed rather than to improve the ordering. A summary of these results can be found in A. George and J.W.H. Liu [512]. The main improvements are the following.

- Mass elimination

When x_i is eliminated, often there are nodes in $Adj_{G^{(i)}}(x_i)$ that can be eliminated immediately. This is because, when x_i is eliminated, only the degrees of nodes in $Adj_{G^{(i)}}(x_i)$ change and some of them can be $deg(x_i) - 1$. For instance, if a node in a clique is eliminated, the degree of all the other nodes in the clique decreases by 1. All these nodes can be eliminated at once, before the degrees are updated, thereby saving some degree updates. This leads to the concept of indistinguishable nodes,

• Two nodes u and v are *indistinguishable* in G if

$$Adj_G(u) \cup \{u\} = Adj_G(v) \cup \{v\}.$$

A. George and J.W.H. Liu [512] proved the following result.

Theorem 3.11. *Let $z \in Adj_{G^{(i)}}(x_i)$. Then, $deg_{G^{(i+1)}}(z) = deg_{G^{(i)}}(x_i) - 1$ if and only if*

$$Adj_{G^{(i)}}(z) \cup \{z\} = Adj_{G^{(i)}}(x_i) \cup \{x_i\}.$$

□

By merging indistinguishable nodes, we only need to update the degrees of the representatives of these nodes.

- Incomplete degree update

Let us introduce a new definition,

• v is said to be *outmatched* by u in G if

$$Adj_G(u) \cup \{u\} \subseteq Adj_G(v) \cup \{v\}.$$

Theorem 3.12. *If v is outmatched by u in $G^{(i)}$, it is also outmatched by u in $G^{(i+1)}$.*

Proof. See A. George and J.W.H. Liu [512]. □

As a consequence, if v becomes outmatched by u, it is not necessary to update the degree of v until u is eliminated.

- Multiple elimination

This variant of the basic scheme was proposed by J.W.H. Liu [756]. When x_i is chosen, we select a node with the same degree as x_i in $G^{(i)}/(Adj_{G^{(i)}}(x_i) \cup \{x_i\})$. This process is repeated until there are no nodes of the same degree, and then the degrees are updated. At each step, an

independent set of minimum degree nodes is selected. The ordering that is produced is not the same as for the basic algorithm. However, it is generally as good as the genuine minimum degree ordering.

- *Early stop*

In many implementations of Gaussian elimination for sparse matrices, a switch is done to dense matrices when the percentage of nonzero entries in the remaining matrix is large enough. Of course, the computation of the ordering can be stopped at that stage since later stages are meaningless.

- *Tie breaking*

An important issue is the choice of a tie breaking strategy. Unfortunately, not much is known about how to decide which nodes to choose at a given stage. Some experiments by A. George and J.W.H. Liu, [512]) showed that there can be large differences in the number of nonzeros and factorization times when several random tie breakers are chosen. Quite often, the initial ordering determines the way ties are broken. It was suggested that another ordering scheme, such as the reverse Cuthill-McKee algorithm, can be used before running the minimum degree algorithm.

- *Approximate minimum degree*

P.R. Amestoy, T.A. Davis, and I.S. Duff [27] proposed to use bounds on the degree of nodes instead of the real degree. It allows a faster update of the information when nodes are eliminated. Techniques based on the quotient graph are used to obtain these bounds. The quality of the orderings that are obtained are comparable to those from the genuine minimum degree algorithm, but the algorithm is much faster; see the performance results in [27].

Figure 3.19 shows the result for the two previous examples using the Matlab minimum degree ordering. For the Wathen matrix the number of nonzero entries in the Cholesky factor is 1924.

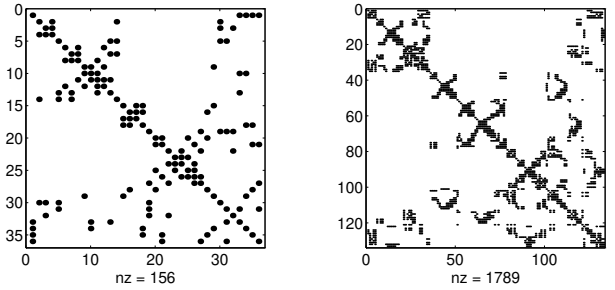


Figure 3.19. *The nonzero structure of the Poisson and Wathen matrices reordered by the minimum degree ordering*

J.W.H. Liu [759] proposed an hybrid between nested dissection (to be discussed below) and the minimum degree algorithm. F. Pellegrini, J. Roman, and P.R. Amestoy [911] proposed an hybrid between nested dissection and the approximate minimum degree algorithm.

Another idea that has been used is to choose at each stage the node that causes the smallest fill-in. This is called the *minimum deficiency* algorithm. It is very costly to implement, see R.D. Berry [125], and M. Nakhla, K. Singhal, and J. Vlach [849]. E. Rothberg and S.C. Eisenstat [959] studied several methods, an approximate minimum mean local fill, an approximate minimum increase in neighbor degree, and variants of the approximate minimum fill. Their conclusion was that the minimum deficiency gives the best quality ordering, but it is expensive. E.G. Ng and P. Raghavan [855] proposed two methods, a modified minimum deficiency and a modified multiple minimum degree.

The minimum degree algorithm was generalized to nonsymmetric problems, see COLMMD

by J.R. Gilbert, C.B. Moler, and R.S. Schreiber [528] and COLAMD by T.A. Davis, J.R. Gilbert, S.I. Larimore, and E.G. Ng [314].

3.8 ▪ The nested dissection ordering

Nested dissection was introduced by A. George [502] for problems arising from finite element discretizations of partial differential equations. It was later generalized to other linear systems. It is very close to an old idea used in Mechanics known as substructuring and also to what is now called domain decomposition. This technique is based on Theorem 3.3 telling that there cannot be a fill-in between x_i and x_j if there is a node with a number greater than x_i and x_j on every path from x_i to x_j in G

As an example, consider the graph of Figure 3.20 and its partition given in the right-hand part of the figure.

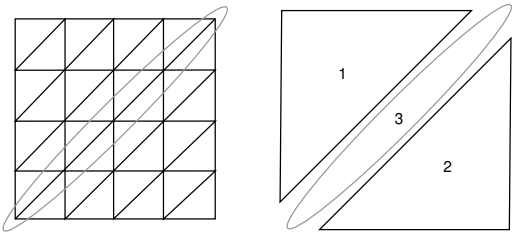


Figure 3.20. *Dissection partitioning*

With nested dissection, the graph is first split into three pieces. The diagonal numbered 3 is called a *separator*. If we first number the nodes in part 1, then the nodes in part 2 and finally the nodes of the separator 3, there cannot be any fill-in between nodes in sets 1 and 2. With this ordering and an obvious notation, the matrix has the following block structure,

$$A = \begin{pmatrix} A_1 & 0 & A_{3,1}^{\mathrm{T}} \\ 0 & A_2 & A_{3,2}^{\mathrm{T}} \\ A_{3,1} & A_{3,2} & A_3 \end{pmatrix},$$

and its Cholesky factor L is,

$$L = \begin{pmatrix} L_1 & & \\ 0 & L_2 & \\ L_{3,1} & L_{3,2} & L_3 \end{pmatrix}.$$

The matrices L_1 and L_2 are respectively the Cholesky factors of A_1 and A_2. The blocks A_1 and A_2 can be factored independently. The basis of the nested dissection algorithm is to apply this idea recursively to the sets 1 and 2. There are basically two ways to partition a rectangular mesh for a rectangle domain. The first one is to partition the graph into vertical (or horizontal) stripes. This is known as *one-way dissection*. The other way is to alternate between vertical and horizontal partitioning obtaining a partition into small rectangles. This is called *nested dissection*. Of course, one-way dissection is a little simpler to implement. One-way dissection can be generalized relatively easily to any sparse matrix by using level structures of their graph, see A. George and J.W.H. Liu [507].

Let us consider nested dissection for a square regular Cartesian mesh. A partition function Π is defined for integers i from 0 to N $(= 2^{\ell})$ as

$$\Pi(0) = 1$$

$$\Pi(N) = 1$$
$$\Pi(i) = p + 1, \text{ if } i = 2^p(2q+1).$$

For example, for $N = 16$, we obtain

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\Pi(i)$	1	1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	1

For $k = 1, \ldots, \ell$ we define sets P_k of mesh nodes (i, j) as

$$P_k = \{(i, j) | \max(\Pi(i), \Pi(j)) = k\}.$$

For a 17×17 mesh we obtain the partition of Figure 3.21 where the numbers refer to the set P_k to which the nodes belong to. Nodes in P_1 are numbered first, then nodes in P_2, etc... up to nodes in P_ℓ. A. George [502] proved that the number of operations for the factorization is $O(N^3)$.

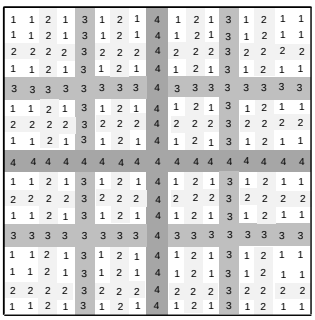


Figure 3.21. *Nested dissection partitioning and numbering*

The partition function was generalized by I.S. Duff, A. Erisman, and J.K. Reid [372] to the case $N \neq 2^\ell$.

The storage schemes we have described previously are not well suited to nested dissection orderings. In nested dissection for mesh problems, there is a natural block structure, each block corresponding to subsets of each P_k. Diagonal blocks are stored by rows in a one dimensional array together with an integer pointer that gives the position of the diagonal element. Non-diagonal blocks are stored in a one dimensional array. It is necessary to know pointers to the start of each block, see Figure 3.22.

Numerical experiments showed that for an $N \times N$ 2D grid, the storage for nested dissection is smaller than for `RCM` when $N > 37$. Figure 3.23 shows the Poisson model problem matrix reordered with nested dissection and the corresponding Cholesky factor.

Nested dissection was revisited by C.C. Ashcraft, I.S. Duff, J. Hogg, J.A. Scott, and S. Thorne [50] in 2016.

3.9 ▪ Generalization of dissection algorithms

For a general sparse matrix there is no underlying mesh, we have to work directly on the graph of the matrix and it is not so easy to find a small separator for partitioning the graph in two or more components of almost an equal number of nodes. There are many ways to handle this problem.

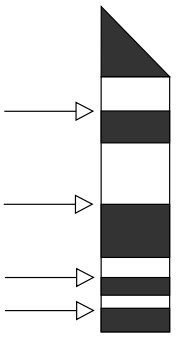


Figure 3.22. *Storage scheme for nested dissection on a regular mesh*

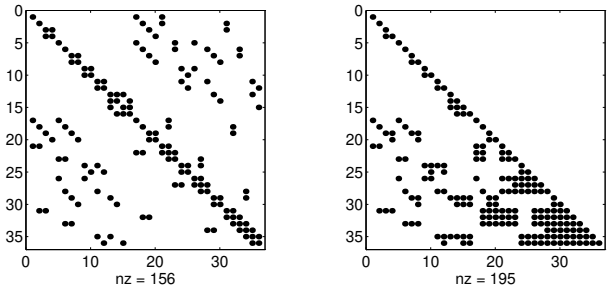


Figure 3.23. *The Poisson matrix with nested dissection (left) and its Cholesky factor (right)*

3.9.1 ▪ General dissection algorithms

General theorems were proved about the existence of good separators using graph theory, see R.J. Lipton, D.J. Rose, and R.E. Tarjan [754], J. Roman [949], and P. Charrier and J. Roman [237, 238]. Without giving too many details, let us describe the kind of results that have been proved.

• Let G be a graph. There is an $f(n)$-separator theorem for $\mathcal{S}$ if there exists α and β, positive real numbers such that the nodes of G can be partitioned into three subsets A, B, C with the following properties

- there is no edges between nodes of A and nodes of B,
- $|A| \leq \alpha n$, $|B| \leq \alpha n$,
- $|C| \leq \beta f(n)$.

C is called a *separator* of G.

R.J. Lipton and R.E. Tarjan [755] proved that $f(n) = \sqrt{n}$ for planar graphs and 2D finite element graphs. The elements of the separator C are numbered last and the same partition algorithm is applied recursively to A and B. It was shown by J. Roman [949] that the following holds.

Theorem 3.13. *Let G be a graph satisfying the previous definition with $f(n) = \sqrt{n}$ with n nodes and m edges,*
- the time to construct the partitions is $O(n + m)$,

- the number of nonzero entries in L is $O(n \log_2 n)$*,*
- the number of floating-point operations for the factorization is $O(n\sqrt{n})$*.*

□

Therefore, we have the same results as for a mesh grid. These results can be slightly generalized, see J. Roman [949].

Over the years many methods have been devised for partitioning graphs. Most of them proceed by recursive bisection. Geometric (or greedy) algorithms can be used that start from the nodes on the "boundary" of the graph accumulating nodes until about one half of the vertices are collected in a subset, see P. Ciarlet Jr. and F. Lamour [265, 266, 267], and C. Farhat [439, 440]. A refinement algorithm is generally used to improve the found partition.

The algorithm by B.W. Kernighan and S. Lin [697] is one of the most used methods for partitioning graphs into two subsets, dating back to 1970. The goal of this heuristic algorithm is to approximately minimize the number of edges joining vertices in both subsets. Let us assume that we have two sets of vertices V_1 and V_2. In this iterative algorithm, at every iteration, we move vertices from one set to the other until we cannot improve any longer the gain which is defined as the number of edges joining V_1 to V_2. If the algorithm moves vertices from V_1 to V_2, then at the next iteration it tries to move vertices from V_2 to V_1 to improve the balance in the number of vertices in both sets. Unfortunately, this process can converge to a local minima. To avoid this situation, the algorithm uses moves with a negative gain for a given number of iterations. The best partition found so far is recorded and the algorithm returns to that partition if the negative moves do not give an improvement after a while. C.M. Fiduccia and R.M. Mattheyses [449] gave an efficient implementation of this algorithm in $O(|E|)$ time where E is the set of edges. The main problem is to find an efficient way to update the gains. The results of the Kernighan and Lin algorithm depend very much on the initial partition. This is why it is often used as a refinement step in other methods.

3.9.2 ▪ Graph bisection improvement techniques

The Kernighan and Lin algorithm is a technique to improve the partition of a graph in two domains. It improves an edge separator. For nested dissection we are more interested in improving a vertex separator. Powerful graph techniques were used and extended by C.C. Aschraft and J.W.H. Liu [53]. The main tool is the Dulmage-Mendelsohn decomposition which has often been used to extract a vertex separator from an edge separator, see [926]. To describe the work in [53] we need a few more definitions,

• a *bisector* is a vertex separator S whose removal gives two components B and W, $Adj(B) \subseteq S$, $Adj(W) \subseteq S$. The partition is denoted $[S, B, W]$. A cost function γ is defined as

$$\gamma(S, B, W) = |S| \left(1 + \alpha \frac{\max\{|B|, |W|\}}{\min\{|B|, |W|\}}\right),$$

where α is a constant whose choice allows to switch between the separator size $|S|$ and the other term measuring the imbalance of the partition.

• Let Y be a subset of vertices. The *interior* of Y is $Int(Y) = \{y \in Y \mid Adj(y) \subseteq Y\}$. The *boundary* of Y is the set of nodes not in Y that are adjacent to Y. The *border* of Y is the boundary of the interior of Y.

C.C. Ashcraft and J.W.H. Liu tried to improve the partition by moving subsets Z that reduce the cost function. This is done by moving a subset from S to the smaller portion W. If it cannot be done, the algorithm tries to move a subset to the larger portion B. The algorithm stops when no more reduction can be obtained.

The choice of which subset of the separator to move is done by using graph matching techniques. When we move a subset Z from S to W, the separator size becomes $|S|-|Z|+|Adj(Z)\cap B|$. If we are able to find a subset Z such that $|Z| > |Adj(Z) \cap B|$, the separator size will be improved. J.W.H. Liu used bipartite graph matching to choose the subset Z. In a bipartite graph the vertices are divided into two subsets X and Y such that every edge had one endpoint in X and one in Y. A *matching* is a subset of edges such that no two edges in this subset have a node in common. A vertex that is incident to an edge in this subset is said to be covered, otherwise it is exposed. The number of edges in the subset is the size of the matching. A *maximum matching* is one with the largest possible size. A *complete matching* is one with a size equal to the size of the smallest of the two sets X and Y.

For the partition $[S, B, W]$, assume that B is the largest subset and consider the bipartite graph $H = (S, border(B), E_H)$ where E_H is the set of edges between the vertices in S and those in $border(B)$. There is a result stating that there exists a subset Z of S satisfying $|Z| \leq |Adj(Z) \cap B|$ if and only if the bipartite graph H has a complete matching. Therefore, we are able to find a move that improves the size of the separator if there are exposed vertices in a maximum matching. J.W.H. Liu introduced the notion of an *alternating path*. For a matching M, this is a path with no repeated vertices if the alternate edges belong to the matching. He proved the following result: if $x \in S$ is an exposed vertex in a maximum matching of H, let $S_x = \{s \in S \mid s \text{ is reachable from } x \text{ via alternating paths}\}$, then $|S_x| - |Adj(S_x \cap B)| = 1$. S_x can be found by a breadth-first search starting from x.

The *Dulmage-Mendelsohn decomposition* is the partition of S into three disjoint subsets: $S = S_I \cup S_R \cup S_X$ with

$S_I = \{s \in S \mid s \text{ is reachable from some exposed vertices in } S \text{ via alternating paths }\}$,
$S_X = \{s \in S \mid s \text{ is reachable from some exposed vertices in } B \text{ via alternating paths }\}$,
$S_R = S \setminus (S_I \cup S_X)$.

This decomposition is independent of the maximum matching used for the definition of the paths. It has been proved that S_I is the smallest subset of S with the maximum reduction of the separator size and $S_I \cup S_R$ is the largest subset with the maximum reduction. Moving S_I or $S_I \cup S_R$ give the same reduction, but the balance of the sizes may be different.

When a reduction of the separator size is not possible, there is still some hope of improving the cost function by reducing the imbalance. When S_I is empty, S_R can be used to reduce the imbalance, see [53].

C.C. Ashcraft and J.W.H. Liu [53] extended the Dulmage-Mendelsohn decomposition to work with weighted graphs. This is useful when working with compressed graphs, saving some computing time. They also related the Dulmage-Mendelsohn decomposition to the solution of a maximum network flow problem. Solving a max flow-min cut problem can be used to improve the partition. Numerical experiments in [53] support the fact that these powerful graph techniques are efficient to refine partitions.

3.9.3 ▪ The multisection algorithm

It is generally accepted that the minimum degree and the nested dissection algorithms produce good orderings. However, the results are not uniformly good. The minimum degree algorithm can produce results far from optimal. Optimal results are given by the nested dissection ordering on grid problems. But there are more general problems for which the recursive ordering of the separators can be improved.

In the approach of C.C. Ashcraft and J.W.H. Liu [54], nested dissection algorithms are used to find recursive bisectors. But, the numbering is different from nested dissection since all the vertices in the union of the separators are numbered last. This is referred as a *multisector*. The constrained minimum degree algorithm is used to order the vertices in the remaining domains.

To order the vertices in the multisector, C.C Ashcraft and J.W.H. Liu considered the elimination graph after the elimination of the vertices in the domains. The vertices of that graph are ordered by the multiple minimum degree. Experimental results in [54] showed that this method performs uniformly well on large sets of examples, at least better than both the minimum degree and nested dissection.

3.10 ▪ Supernodal methods

In sparse matrix factorization there are often columns and rows with a similar nonzero pattern. A *supernode* is a set of such contiguous columns. Formally, a supernode is a set of contiguous nodes $\{j, j+1, \ldots, j+s\}$, such that

$$Adj_G(T[j]) = \{j+1, \ldots, j+s\} \cup Adj_G(T[j+s]).$$

Supernodal factorizations take advantage of that to save computing time and storage by storing les integer information and using dense matrix techniques for a supernode which is stored as a lower trapezoidal dense matrix for the nonzero entries. In the elimination tree the nodes of a supernode can be almagamated and treated as a single computational unit.

Consider an example with the graph of Figure 3.24 and the corresponding elimination tree in Figure 3.25. The supernodal elimination tree is given in Figure 3.26. We have

$$Adj_G(T[7]) = \{8, 9\} = \{8, 9\} \cup Adj_G(T[9]) = \{8, 9\} \cup \emptyset.$$

Therefore, $\{7, 8, 9\}$ is a supernode. All the nodes in a supernode are eliminated in the same step of the algorithm.

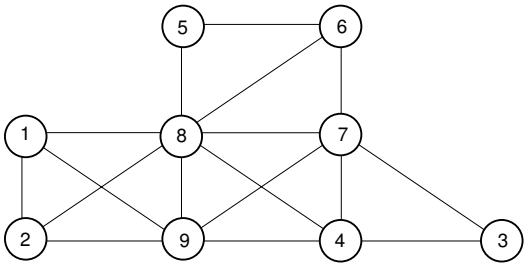


Figure 3.24. *An example of graph*

Sometimes, supernodes are extended to columns having almost the same pattern to obtain larger supernodes. The numerical factorization computes the supernodes of L one after the other.

Although there were early papers mainly using these block ideas, C.C. Ashcraft, R.G. Grimes, J.G. Lewis, B.W. Peyton, and H.D. Simon [52] described the first left-looking supernodal factorization. E. Rothberg and A. Gupta [960, 961] proposed a right-looking supernodal algorithm. An efficient method for finding supernodes was proposed by J.W.H. Liu, E.G. Ng, and B.W. Peyton [763].

Supernodal methods were later extended to nonsymmetric problems, see J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Liu [333].

3.11 ▪ The multifrontal method

The multifrontal method was introduced by I.S. Duff and J.K. Reid [381, 382] as a generalization of the frontal method developed by B. Irons [666] for finite element problems in the 1970s. The

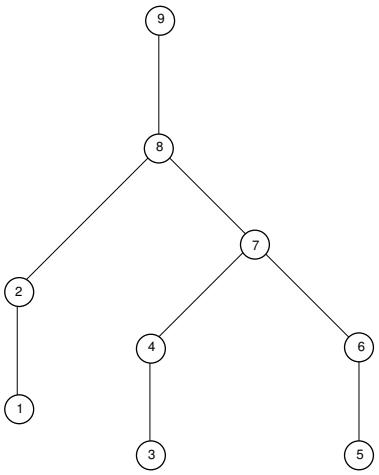


Figure 3.25. *The elimination tree*

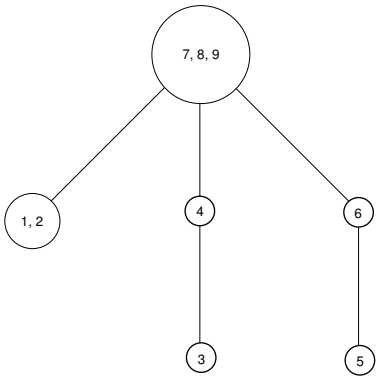


Figure 3.26. *The supernodal elimination tree*

origin of the multifrontal method can be seen in the generalized element method developed by B. Speelpenning [1037].

The main goal of the multifrontal method is to be able to use dense matrix technology for sparse matrices, and also to allow more parallelism. A possible downside of the method is that technical details are quite complex and many refinements are necessary to make the method efficient. A nice description of the principles of the method was given by J.W.H. Liu [761].

The basis of the method for a symmetric matrix is the block outer product Cholesky factorization,

$$A = \begin{pmatrix} D & B^{\mathrm{T}} \\ B & C \end{pmatrix} = \begin{pmatrix} L_D & 0 \\ BL_D^{-\mathrm{T}} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & C - BD^{-1}B^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} L_D^{\mathrm{T}} & L_D^{-1}B^{\mathrm{T}} \\ 0 & I \end{pmatrix},$$

with $D = L_D L_D^{\mathrm{T}}$. The Schur complement $C - BD^{-1}B^{\mathrm{T}}$ is the next matrix to be factored. Let D be of order $j-1$,

$$BD^{-1}B^{\mathrm{T}} = (BL_D^{-\mathrm{T}})(L_D^{-1}B^{\mathrm{T}}) = \sum_{k=1}^{j-1} \begin{pmatrix} \ell_{j,k} \\ \vdots \\ \ell_{n,k} \end{pmatrix} (\ell_{j,k} \quad \dots \quad \ell_{n,k}),$$

$\ell_{i,k}$ being the entries of the Cholesky factors.

During the symbolic phase, the elimination tree is computed, as well as the *assembly tree* after amalgamation of some nodes. It allows the factorization to be done as the partial factorizations of several smaller dense matrices (or fronts) located at each node of the assembly tree. The constraint is that the computation of a parent front must be done after the computation of all its child fronts is finished. Dense matrix kernels are used for the frontal matrices. I.S. Duff and J.K. Reid [381] implemented the first multifrontal method in the code MA27.

Let us look at a small example,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & & & x & x & x \\ & x & & & x & \\ & & x & & x & x \\ x & & & x & & x \\ x & x & x & & x & x \\ x & & x & x & x & x \end{pmatrix} \end{array}.$$

The graph of the matrix A is shown in Figure 3.27.

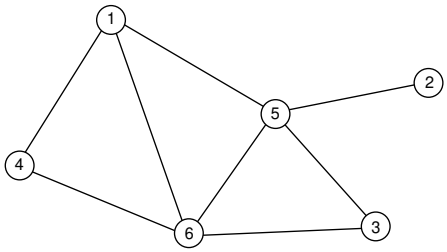


Figure 3.27. *The graph of A*

Gaussian elimination on A yields

$$L = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & & & & & \\ & x & & & & \\ & & x & & & \\ x & & & x & & \\ x & x & x & \bullet & x & x \\ x & & x & x & x & x \end{pmatrix} \end{array}.$$

The elimination tree $T(A)$ is shown in Figure 3.28.

From the elimination tree, we see that we can eliminate the nodes $1, 2$ and 3 independently. If we consider node 1, we can restrict ourselves to the following matrix (rows and columns where there are nonzero entries in the first row and first column),

$$F_1 = \begin{array}{c} \\ 1 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{4,1} & & & \\ a_{5,1} & & & \\ a_{6,1} & & & \end{pmatrix} \end{array}.$$

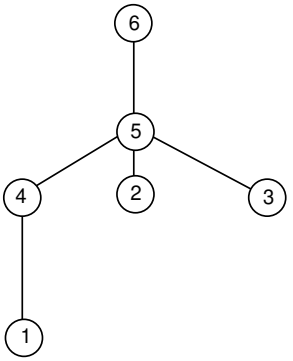


Figure 3.28. *The elimination tree of A*

Eliminating node 1 creates contributions in a reduced matrix $\bar{U}_4$,

$$\bar{U}_4 = \begin{array}{c} \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} x & \bullet & x \\ \bullet & x & x \\ x & x & x \end{pmatrix} \end{array},$$

where the bullet $\bullet$ represents a fill-in. In parallel, we can eliminate node 2, defining

$$F_2 = \begin{array}{c} \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 5 \end{array} \\ \begin{pmatrix} a_{2,2} & a_{2,5} \\ a_{5,2} & \end{pmatrix} \end{array}.$$

Elimination of node 2 creates a contribution to the $(5,5)$ term,

$$\bar{U}_5^2 = \begin{array}{c} \\ 5 \end{array} \begin{array}{c} \begin{array}{c} 5 \end{array} \\ \begin{pmatrix} x \end{pmatrix} \end{array}.$$

Simultaneously, we can also eliminate node 3,

$$F_3 = \begin{array}{c} \\ 3 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{3,3} & a_{3,5} & a_{3,6} \\ a_{5,3} & & \\ a_{6,3} & & \end{pmatrix} \end{array}.$$

Elimination of node 3 creates contributions,

$$\bar{U}_5^3 = \begin{array}{c} \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cc} 5 & 6 \end{array} \\ \begin{pmatrix} x & x \\ x & x \end{pmatrix} \end{array}.$$

Then, we eliminate node 4. To do this, we have to consider the matrix resulting from the elimination of node 1, that is,

$$F_4 = \begin{array}{c} \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 6 \end{array} \\ \begin{pmatrix} a_{4,4} & 0 & a_{4,6} \\ 0 & & \\ a_{6,4} & & \end{pmatrix} \end{array} + \bar{U}_4.$$

Elimination of node 4 creates contributions,

$$\bar{U}_5^4 = \begin{array}{c} \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cc} 5 & 6 \end{array} \\ \begin{pmatrix} x & x \\ x & x \end{pmatrix} \end{array}.$$

Before eliminating node 5, we must sum the contributions from the original matrix and what we get from the eliminations of nodes $2, 3$ and 4. We must extend $\bar{U}_5^2$ to the proper set of indices, that is, $5, 6$. We do this as in J.W.H. Liu [761] by considering an operator that we denote by $\circ$. For two matrices A and B, the set of indices of $A \circ B$ is the union of the sets of indices of A and B and whenever they coincide, and the result is the sum of the entries. Let

$$\bar{U}_5 = \bar{U}_5^2 \circ \bar{U}_5^3 \circ \bar{U}_5^4,$$

then

$$F_5 = \begin{pmatrix} a_{5,5} & a_{5,6} \\ a_{6,5} & 0 \end{pmatrix} + \bar{U}_5.$$

Elimination of node 5 gives a matrix of order 1 that is added to $a_{6,6}$ to give the last entry of the factorization.

In this example, we have seen that all the elimination steps can be carried out by working on small dense matrices of different orders, extending and summing these matrices by looking at the elimination tree.

Let us formalize the process of the multifrontal method. Giving the elimination tree $T(A)$, we define the subtree update matrix for column j as

$$\bar{U}_j = - \sum_{k \in T[j] - \{j\}} \begin{pmatrix} \ell_{j,k} \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{j,k} & \ell_{i_1,k} & \dots & \ell_{i_r,k} \end{pmatrix},$$

where, $i_0 = j, i_1, \dots, i_r$ are the row indices of the nonzero entries in column j of L, that is, $(L_{:,j})$.

The matrix $\bar{U}_j$ contains the contributions for the columns preceding j which are proper descendants of j in the tree. The frontal matrix F_j is defined as

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \dots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & & \\ a_{i_r,j} & & & \end{pmatrix} + \bar{U}_j.$$

The first column of F_j contains all the nonzero updated entries to column j. Then, we perform one step of Gaussian elimination on F_j,

$$F_j = \begin{pmatrix} \ell_{j,j} & 0 \\ \ell_{i_1,j} & \\ \vdots & I \\ \ell_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & U_j \end{pmatrix} \begin{pmatrix} \ell_{j,j} & \ell_{i_1,j} & \dots & \ell_{i_r,j} \\ 0 & & I & \end{pmatrix}.$$

The dense matrix U_j is called the update matrix. It was proved by J.W.H. Liu [761] that

$$U_j = - \sum_{k \in T[j]} \begin{pmatrix} \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{i_1,k} & \dots & \ell_{i_r,k} \end{pmatrix}.$$

If $c_1, \ldots, c_s$ denotes the children of j in $T(A)$,

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \ldots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & & \\ a_{i_r,j} & & & \end{pmatrix} \circ U_{c_1} \circ \ldots \circ U_{c_s}.$$

The multifrontal method is defined as

for $j = 1 : n$
1) form the update matrix $U_{c_1} \circ \ldots \circ U_{c_s}$,
2) form the frontal matrix F_j,
3) factorize F_j,
end

Many issues have to be considered to obtain an efficient multifrontal code. The first one is the storage of the frontal and update matrices. Update matrices must be stored and easily retrieved when they are needed in the algorithm to contribute to a frontal matrix. A way to do this is to use a topological ordering of $T(A)$ and to number the nodes in every subtree consecutively. The update matrices can be stored in a stack using a last-in first-out algorithm. Using this technique, the update matrices appear at the top of the stack in the order they are needed.

This was not very well suited for parallel computations. To manage the storage working space, I.S. Duff proposed to use a buddy system. In that technique, each block of storage has a buddy with which it can be combined to form a larger block. In a binary buddy system, the sizes of the block are $c2^i$. Each block keeps some associated information, a flag to indicate if the block is free or not and the logarithm of the size of the block. Free blocks are linked through a doubly linked list. There is also a free list for each block size. When a working area of memory of size m is needed, the system allocates a block of list i, where $c2^i \geq m$. If there is no block on the i-th free list, a level $i+1$ block must be split in two, and part is used to serve the request, the other part is put on the i-th free list. When a block is deallocated, the system checks if the block's buddy is free and of the correct size. If the answer is positive, the two blocks are combined and put on the $i+1$-rst free list.

The second issue is the use of dense techniques. As we said above, an advantage of the multifrontal method is to allow to use dense matrix kernels in the sparse case. Dense algorithms based on the use of Level 3 BLAS can be used when factoring the frontal matrices. Furthermore, there is much less indirect addressing than in pure sparse factorizations.

The third issue is node amalgamation. There are some performance advantages to have large submatrices. It was suggested by I.S. Duff and J.K. Reid [382] to amalgamate some nodes, treating some zero entries as nonzero, see P.R. Amestoy [23].

3.12 ▪ Nonsymmetric sparse matrices

There is an additional difficulty in Gaussian elimination for nonsymmetric sparse problems, namely the need of pivoting to improve the numerical stability. When dealing with sparse symmetric positive definite systems, the ordering of the unknowns can be chosen for the purpose of maintaining, as much as possible, the sparsity of the factors during elimination. This is not true any longer for nonsymmetric problems, except in special cases.

We would like to obtain an LU factorization of a nonsymmetric sparse matrix A, or of a permutation of A. An early paper considering the nonsymmetric case was by N. Sato and W.F. Tinney [991] in 1963. Symbolic factorization without pivoting was studied by D.J. Rose and R.E. Tarjan [952] using directed graphs, see also S.C. Eisenstat and J.W.H. Liu [406, 408].

If we choose the pivots as for dense systems, for instance, using partial pivoting, there is no room for preserving sparsity. For sparse matrices, we have to relax the constraints for choosing the pivot. A common strategy is to consider candidate pivots satisfying the inequality,

$$|a_{i,j}^{(k)}| \geq \omega \max_l |a_{l,j}^{(k)}|,$$

where ω is a user defined parameter such that $0 < \omega \leq 1$. This limits the overall growth of the entries since

$$\max_i |a_{i,j}^{(k)}| \leq \left(1 + \frac{1}{\omega}\right)^{p_j} \max_i |a_{i,j}|,$$

where p_j is the number of off-diagonal entries in column j of U, see I.S. Duff, A.M. Erisman and J.K. Reid [373]. From these pivot candidates, one is selected that minimizes

$$(r_i^{(k)} - 1)(c_j^{(k)} - 1),$$

where $r_i^{(k)}$ is the number of nonzero entries in row i of the remaining $(n-k) \times (n-k)$ matrix in A_k, and $c_j^{(k)}$ is the number of nonzero entries in column j. This is known as the Markowitz criterion since something similar was used by H. Markowitz [786] in 1957. This choice modifies the smallest number of entries in the remaining submatrix. If A is symmetric, this is exactly the minimum degree algorithm. Many variations of the Markowitz criterion have been studied over the years, see [373]. Most of these other methods are generally not as efficient as the Markowitz criterion. Another possibility is to choose the (not too small) entry that introduces the least amount of fill-in at step k. This is much more expensive than the Markowitz criterion. Moreover, having a local minimum fill-in does not always gives a globally optimal fill-in count. There are even some examples where the Markowitz criterion is better at globally reducing the fill-in. As for the minimum degree algorithm, the tie-breaking strategy is important when using the Markowitz criterion. Details are discussed in [373].

About this type of pivoting, see A.R. Curtis and J.K. Reid [298], Y.T. Chen and R.P. Tewarson [249], I.S. Duff and J.K. Reid [379, 380, 383], Z. Zlatev [1165], O. Østerby and Z. Zlatev [886], I.S. Duff [368], M. Arioli, J.W. Demmel, and I.S. Duff [40], T.A. Davis and I.S. Duff [313], and T.A. Davis [310].

The structures of the triangular factors of a nonsymmetric matrix without pivoting can be characterized, see J.R. Gilbert and J.W.H. Liu [527]. They used two directed acyclic graphs (DAGs). Acyclic means that there is no directed cycles. Let $w = (w_1, \ldots, w_n)^{\mathrm{T}}$. We define

$$Struct(w) = \{i \in \{1, \ldots, n\} \,|\, w_i \neq 0\}.$$

Theorem 3.14. *If $Lx = b$, then $Struct(x)$ is given by the set of vertices reachable from vertices of $Struct(b)$ in the DAG $G(L^T)$.*

□

An economical way to represent the information contained in a DAG G is to consider its *transitive reduction* G^0. In Theorem 3.14, we can replace $G(L^T)$ by $G^0(L^T)$. The *transitive closure* G^* of a directed graph G is a graph having an edge (u, v) whenever G has a directed path from u to v. With nonzero diagonal entries and without numeric cancellation $G^*(A) = G(A^{-1})$.

Let A be factored as $A = LU$ without pivoting. $G^0(L)$ and $G^0(U)$ are called the lower and upper elimination DAGs (eDAGs) of A. For a symmetric matrix, $G^0(L)$ and $G^0(U)$ are both equal to the elimination tree.

If B and C are two matrices with nonzero diagonal entries, $G(B)+G(C)$ is the union of the graphs of B and C, that is, the graph whose edge set is the union of those of $G(B)$ and $G(C)$. $G(B) \cdot G(C)$ is the graph with an edge (i,j) if (i,j) is an edge of $G(B)$ or (i,j) is an edge of $G(C)$ or if there is a k such that (i,k) is an edge of $G(B)$ and (k,j) is an edge of $G(C)$. J.R. Gilbert and J.W.H. Liu [527] proved the following result.

Theorem 3.15. *If $A = LU$ and there is a path in $G(A)$ from i to j, there exists a $k, 1 \leq k \leq n$ such that $G^0(U)$ has a path from i to k and $G^0(L)$ has a path from k to j. That is,*

$$G^*(A) \subseteq G^{0^*}(U) \cdot G^{0^*}(L).$$

□

If there is no numeric cancellation in the factorization $A = LU$, then

$$G(L) \cdot G(U) = G(L) + G(U).$$

From these results, the row and column structures of L and U can be obtained.

Theorem 3.16. *If $\ell_{i,j} \neq 0$, there exists a path from i to j in $G^0(L)$. Let $i > j$. Then $\ell_{i,j} \neq 0$ if and only if there exists $k \leq j$ such that $a_{i,k} \neq 0$ and there is a directed path in $G^0(U)$ from k to j,*

$$Struct(L_{:,j}) = Struct(A_{*,j}) \cup \bigcup \{Struct(L_{:,k}) \mid k < j, u_{k,j} \neq 0\} - \{1, \ldots, j-1\}.$$

□

The structure of U can be characterized in the same way. From these results, an algorithm can be derived for the symbolic fill computation when there is no pivoting. The use of elimination trees in sparse nonsymmetric elimination was discussed by S.C. Eisenstat and J.W.H. Liu [409, 410].

Symbolic LU factorization with pivoting was studied by A. George and E.G. Ng [514, 515]. If $PA = LU$ and $A = QR$, with Q orthogonal and R upper triangular, the pattern of R is an upper bound on the pattern of U. They relied on that result to preallocate storage for L and U. Their result was improved by J.R. Gilbert and E.G. Ng [529], and J.R. Gilbert and L. Grigori [525]. They showed that the upper bound is tight if A has the *strong Hall property*, which means that it cannot be permuted to block upper triangular form with more than one block. When the matrix is not strong Hall, see L. Grigori, M. Cosnard, and E.G. Ng [573], and L. Grigori, J.R. Gilbert, and M. Cosnard [576].

When the nonsymmetric matrix A has a (nearly) symmetric nonzero structure, the nonzero structures of L and U are identical to those of the Cholesky factors of a matrix with the nonzero structure of $A + A^T$, and an ordering to reduce the fill-in in $A + A^T$ can be used. There are also cases for which it is not necessary to use pivoting. For instance, when A is strictly diagonally dominant.

Over the years, like for the symmetric case, many variants of Gaussian elimination were used for nonsymmetric linear systems. A left-looking variant was used by A.H. Sherman [1004, 1003], J.R. Gilbert and T. Peierls [531], S.C. Eisenstat and J.W.H. Liu [407], T.A. Davis [312], and X. Chen, Y. Wang, and H. Yang [248]. Right-looking LU factorization was used by S.C. Eisenstat, M.H. Schultz, and A.H. Sherman [413], and A. George and E.G. Ng [514, 516].

As we said above, J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li and J.W.H. Liu [333] introduced the idea of nonsymmetric supernodes, and used them in a left-looking method. Like for the symmetric case, a supernode is a range of columns in L corresponding to a full triangular diagonal block and with the same structure below the diagonal block. This allows a supernode to do column updates using BLAS routines. The authors observed that the corresponding rows in U have a special nonzero pattern and they took advantage of that. They also computed an upper bound on the nonzero structure of L and U that allowed to do partial pivoting with row swaps. The *column elimination tree* T is defined as the elimination tree of A^TA. It gives the dependencies among columns in the LU factorization. In particular, if $\ell_{i,j} \neq 0$ then, i is an ancestor of j in T, if $u_{i,j} \neq 0$ then, j is an ancestor of i in T. Before factoring the matrix, the columns are permuted according to a postordering of the column elimination tree. For an overview of the SuperLU code and its parallel implementations, see X.S. Li [740, 741]. For a recent update on that software, see X.S. Li, P. Lin, Y. Liu, and P. Sao [742].

Another code using supernodes was described by O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker [993].

The multifrontal algorithm we have previously described for symmetric matrices was generalized to nonsymmetric matrices by I.S. Duff and J.K. Reid [382]. Their idea was to consider the sparsity pattern of $A + A^{\mathrm{T}}$ to construct the elimination tree. Although the frontal matrices may be numerically nonsymmetric, they are square and with a symmetric structure. Numerical pivoting takes place within the frontal matrices. This works well if the pattern of A is nearly symmetric. The results may be poor if the pattern of A is far from being symmetric.

T.A. Davis and I.S. Duff introduced a nonsymmetric pattern multifrontal algorithm using rectangular frontal matrices that are related either by the column elimination tree or by a directed acyclic graph (DAG), see T.A. Davis and I.S. Duff [313], and T.A. Davis [308, 309, 310, 312].

P.R. Amestoy and C. Puglisi [34] introduced a nonsymmetric version of the multifrontal method that can be regarded as being an intermediate between the nonsymmetric-pattern and the symmetric-pattern multifrontal methods.

A. Gupta [581, 582] used two assembly DAGs that can be constructed before the symbolic factorization for the nonsymmetric pattern multifrontal method.

3.13 ▪ Numerical stability for sparse matrices

Componentwise error analysis for sparse systems was considered in M. Arioli, J.W. Demmel, and I.S. Duff [40]. They computed estimates of the backward error. The perturbation f of the right-hand side is computed a posteriori and is not equal to $|b|$ to keep the perturbation on A sparse and the iterative refinement algorithm convergent.

Let $w = |A|\,|y| + |b|$, y being the computed solution. A threshold τ_i is chosen for each w_i such that if $w_i > \tau_i$, then $f_i = |b_i|$. Otherwise, if $w_i \leq \tau_i$, f_i is chosen larger. The value of τ_i suggested in [40] is $\tau_i = 1000\, n\, u(\|A_{i,*}\|_\infty \|y\|_\infty + |b_i|)$, where $A_{i,*}$ is the ith row of A.

Let $f^{(1)}$ (resp. $f^{(2)}$) be the components of f for which $w_i > \tau_i$ (resp. $w_i \leq \tau_i$). Then $f^{(2)}$ is defined as $f^{(2)} = \|b\|_\infty e$ where e is the vector of all ones. With this choice, we can compute an estimate of the backward error,

$$\frac{|b - Ay|_i}{(|A||y| + f)_i}.$$

Remember that the componentwise condition number is defined as

$$K_{BS}(A, b) = \frac{\| \, |A^{-1}|(E|x| + f)\|_\infty}{\|x\|_\infty}.$$

If $E = |A|$, we may use the estimate $\| \, |A^{-1}|\,|A| \, \|_\infty = \| \, |A^{-1}|\,|A|e\|_\infty$. This can be obtained by

an algorithm due to W.W. Hager [607] that uses multiplications by the matrix and its transpose.

Numerical experiments in [40] using iterative refinement showed that it is possible to guarantee solutions of sparse linear systems that are exact solutions of a nearby system with a matrix of the same nonzero structure. Estimates of the condition number and of the backward error can be easily obtained.

3.14 ▪ Parallel algorithms for sparse matrices

As for dense matrices, it is important to be able to solve efficiently sparse linear systems on parallel computers. On one hand, it is easier to solve sparse linear systems rather than dense ones since there is more natural parallelism. Data dependencies are weaker in the sparse case since in the LU factors some columns are independent of each other. On the other hand, it is more difficult to obtain significant performances since the granularity of independent tasks is often quite small and indirect addressing could lead to a poor data locality.

Thousands of papers have been written on parallel methods for sparse matrices, and a lot of research is still going on to obtain efficient algorithms since the computer architectures are evolving with time.

Let us first consider symmetric matrices. We have seen that there is no need to pivot for stability, and Gaussian elimination is done in three phases: ordering, symbolic factorization and numerical factorization. The problem researchers are facing is to have parallel implementations of the three phases. For the first phase, it is not only necessary to find an ordering that reduces the fill-in and gives a good degree of parallelism during the factorization phase, but also, ideally, to be able to compute this ordering in parallel.

We have seen that a widely used algorithm for reducing fill-in is the minimum degree algorithm. Unfortunately, this method is quite sequential by nature. It must be modified to run efficiently on parallel computers. J.W.H. Liu [756] proposed to look for multiple elimination of independent nodes of minimum degree.

An ordering that is more promising regarding parallelism is nested dissection since it is a "divide and conquer" algorithm. Good graph partitioners like MeTis, ParMeTis, Scotch and PT-Scotch [910, 250] are available to generate a nested dissection ordering. Combinations of the minimum degree and nested dissection were proposed by J.W.H. Liu [759].

I.S. Duff, N.I. Gould, M. Lescrenier, and J.K. Reid [374] compared the minimum degree and nested dissection orderings. They concluded that minimum degree produces very tall and thin trees while nested dissection produces short and large trees which thus give more potential parallelism.

An approach to introduce parallelism in the factorization phase is to first use an ordering reducing the fill-in. Then, it is modified by restructuring the elimination tree to introduce more parallelism. This was described by J.A.G. Jess and H.G.M. Kees [680]. Their method considers PAP^T, with a permutation P chosen to preserve sparsity. Then, the natural ordering is a perfect elimination one for $F = L+L^T$. Their goal was to find another permutation matrix Q giving also a perfect elimination, but with more parallelism. A node in the graph G_F whose adjacency set is a clique is called *simplicial*. Such a node can be eliminated without causing any fill-in. Two nodes are independent if they are not adjacent in G_F. Until all nodes are eliminated, the Jess and Kees algorithm chooses a maximum set of independent simplicial nodes, numbers them consecutively and eliminates these nodes. It was shown by J.W.H. Liu [758] that this method gives an ordering that has the shortest elimination tree over all orderings that yield a perfect elimination of F. An implementation using clique trees was described by J.G. Lewis, B.W. Peyton, and A. Pothen [735]. Another one was proposed by J.W.H. Liu and A. Mirzaian [762]. In their method, the cost of detecting simplicial nodes is $O(n\nu(F))$ where $\nu(F)$ is the number of off diagonal entries in

$L + L^T$.

Larger elimination trees, having more leaf nodes, introduce more parallelism. The number of nodes being fixed, larger trees mean shorter trees. Therefore, finding an ordering that gives a short tree would increase the level of parallelism. J.W.H. Liu [758] proposed to use tree rotations to reach this goal. The purpose of this algorithm is to find a reordering by working on the structure of the elimination tree. A node x in a tree $T(B)$ is eligible for rotation if

$$Adj_{G(B)}(T[x]) \neq Anc(x),$$

where $Anc(x)$ is the set of ancestors of x in T and

$$Adj_{G(B)}(T[v]) = Anc(v), \ \forall v \text{ ancestor of } x.$$

A tree rotation at x is a reordering of $G(B)$ such that the nodes in $Adj_{G(B)}(T[x])$ are labeled last while keeping the relative order of the nodes. Implementation details and experimental results were given in [758]. Unfortunately, tree rotations do not always give a tree of minimum height.

For the numerical factorization, the first parallel algorithms that were studied were column oriented with a distribution of columns in the processors' memories. Leaf nodes of the elimination tree are independent of each other and can be processed first. A fan-in algorithm was described by C.C Ashcraft, S.C. Eisenstat, and J.W.H. Liu [51].

An important issue in these column oriented algorithms is the mapping of columns to the processors. Most implementations used a static mapping of computational tasks to processors. This can lead to load balancing problems. In the fan-out or fan-in algorithms, the assignment of columns to processors is guided by the elimination tree. The goals are good load balancing and low processor communications. The first implementations were based on wrap mapping of the levels of the elimination tree starting from the bottom up. It gives good load balancing properties but too many communications. Another technique is the subtree to subcube mapping, see A. George, M.T. Heath, J.W.H. Liu, and E.G. Ng [506]. This was specifically designed for hypercube architectures but can be easily generalized to other distributed memory architectures. Of course, one can assign supernodes to processors instead of single columns.

All these mappings are based on column distribution of the matrix to the processors. E. Rothberg and A. Gupta [962] used a block oriented approach for sparse Gaussian elimination.

The multifrontal method is well suited for parallel computers. We have seen that there is a natural parallelism in the early (bottom) stages of the multifrontal method. Dense frontal matrices are assigned to one processor and can be processed in parallel. When moving towards the root of the tree, there is less and less parallelism. However, frontal matrices are getting larger and larger and dense techniques, using BLAS3 kernels, can be used to handle these matrices if they are distributed on several processors. This was discussed by I.S. Duff [369, 370, 371] who showed how to interleave both tree and node parallelism during the factorization.

The multifrontal method was implemented by G. Kapyris and V. Kumar [693] using a subforest to subcube mapping. In this method, many subtrees of the elimination tree are assigned to each subcube. They are chosen in order to balance the work.

The first nonsymmetric pattern multifrontal algorithm and the code UMFPACK were described by T.A. Davis and I.S Duff [313]. A parallel nonsymmetric version of the multifrontal method was studied by S.M. Hadfield and T.A. Davis [603]. They used different levels of parallelism and provided experimental results. Later on, T.A. Davis [310] added a symbolic preordering and analysis, using the COLAMD ordering.

P.R. Amestoy, I.S Duff, and J.Y. L'Excellent [29] and P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, and J. Koster [30] presented a fully asynchronous approach with distributed dynamic scheduling. This led to the parallel code MUMPS.

P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, and X.S.Li [31] compared the code SuperLU DIST, implementing a synchronous supernodal method with static pivoting and iterative refinement, and MUMPS, implementing an asynchronous multifrontal method with partial threshold and delayed pivoting. P.R. Amestoy, A. Guermouche, J.Y. L'Excellent, and S. Pralet [33] improved the MUMPS scheduling strategy.

Since many high performance computers rely on GPUs to obtain most of their performance, it became necessary to port sparse codes to these architectures or to write new ones. This started at the end of the 2000s.

These architectures work well for regular parallelism, but pose a challenge for sparse direct methods because of the irregular nature of the algorithms. D. Pierce, Y. Hung, C.C. Liu, Y.H. Tsai, W. Wang, and D. Yu [918] factorized large frontal matrices in the GPU in their multifrontal Cholesky code. G. Krawezik and G. Poole [714] accelerated the ANSYS direct sparse solver with GPUs. Multifrontal Cholesky or LU factorizations using GPUs were described by R.F. Lucas, G. Wagenbreth, D.M. Davis, and R. Grimes [767], T. George, V. Saxena, A. Gupta, A. Singh, and A.R. Choudhury [518], and S.C. Rennich, D. Stosic, and T.A. Davis [941, 942]. C.D. Yu, W. Wang, and D. Pierce [1148] extended GPU acceleration to the nonsymmetric pattern multifrontal method. A sparse symmetric indefinite solver with pivoting for GPUs was described by J.D. Hogg, E. Ovtchinnikov, and J.A. Scott [644]. High performance sparse multifrontal solvers on modern GPUs were considered by P. Ghysels and R. Synk [521]; see also the contribution of Y. Xia, P. Jiang, G. Agrawal, and R. Ramnath [1130].

P. Sao, R. Vuduc, and X.S. Li [990] presented a GPU-based algorithm for the supernodal code SuperLU_DIST. In [989] they introduced what they called a communication-avoiding 3D algorithm. The last developments of this code and the use of GPUs were described by X.S. Li, P. Lin, Y. Liu, and P. Sao [742].

The use of heterogeneous and distributed architectures for the supernodal code PaSTiX is discussed in X. Lacoste's thesis [718].

3.15 ▪ Low rank approximations

There are some problems for which submatrices taken from the lower and upper triangular parts are of low rank, or can be approximated by low rank matrices. This can be exploited to decrease the storage and the complexity of the linear solves. These low-rank properties are generally defined recursively, based on block partitioning,

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}.$$

The HODLR (*Hierarchically Off-Diagonal Low-Rank*) class is defined with a binary *cluster tree* $\mathcal{T}_p$ with $p+1$ levels which represents a recursive partitioning of the set of integers $\mathcal{I} = \{1, 2, \ldots, n\}$. The root of the tree is $I_1^0 = \mathcal{I}$. The nodes at level ℓ are denoted as $I_1^\ell, \ldots, I_{2^\ell}^\ell$ with

$$I_i^\ell = \{n_{i-1}^{(\ell)} + 1, \ldots, n_i^{(\ell)} - 1, n_i^{(\ell)}\},$$

for integers $0 = n_0^{(\ell)} \leq n_1^{(\ell)} \leq \cdots \leq n_{2^\ell}^{(\ell)} = n$ with $0 \leq \ell \leq p$. Each node has two children and, at each level, we have a partitioning of $\mathcal{I}$. The nodes at level ℓ gives a partitioning of a matrix A of order n into $2^\ell \times 2^\ell$ blocks $A_{I_i^\ell, I_j^\ell}$ with $i, j = 1, \ldots, 2^\ell$.

Given a cluster tree $\mathcal{T}_p$ and an integer k, a matrix A is an HODLR matrix if, for all $\ell = 1, \ldots, p$, every off-diagonal block $A_{I_i^\ell, I_j^\ell}$, where I_i^ℓ and I_j^ℓ are nodes on the same level under the same parent node, is of rank at most k.

A matrix A is an HSS (*Hierarchically SemiSeparable*) matrix if every HSS block row or column is at most of rank k. An HSS block row (resp. column) is $A_{I_i^\ell, \mathcal{I}\setminus I_i^\ell}$ (resp. $A_{\mathcal{I}\setminus I_i^\ell, I_i^\ell}$).

A matrix A is called *semiseparable* if every submatrix taken out from the upper or lower triangular part of A has rank at most one, see [1093]. For more general classes of hierarchical matrices, see W. Hackbusch [601].

Solvers that exploit these low-rank properties may be used as direct solvers or as preconditioners for iterative methods, depending on the accuracy that is required.

These type of matrices have been used in supernodal and multifrontal matrices. This is based on the assumption that submatrices close to the diagonal have a high rank while submatrices far away from the diagonal have low rank. J. Xia, S. Chandrasekaran, M. Gu, and X.S.Li [1128, 1129] relied on an HSS representation of the frontal matrices in a multifrontal method. These submatrices are partitioned into four blocks. The two off-diagonal blocks are compressed through a truncated SVD and the diagonal blocks are recursively partitioned; see also J. Xia [1127], S. Wang, X.S. Li, F.H. Rouet, J. Xia, and M.V. De Hoop [1112], as well as F.H. Rouet, X.S. Li, P. Ghysels, and A. Napov [963]. The STRUMPACK code has support for several types of hierarchical matrices.

P.R. Amestoy, C.C. Ashcraft, O. Boiteau, A. Buttari, J.Y. L'Excellent, and C. Weisbecker [21] discussed the use of block low-rank (BLR) representations in the multifrontal code MUMPS. The performance and scalability on multicore architectures were reported by P.R. Amestoy, A. Buttari, J.Y. L'Excellent, and T. Mary [25, 26]; see also T. Mary's PhD. thesis [788]. G. Pichon, E Darve, M. Faverge, P. Ramet, and J. Roman [917] presented a sparse supernodal solver using block low-rank compression.

A Matlab toolbox for manipulating HODLR and HSS matrices was described by S. Massei, L. Robol, and D. Kressner [789]. Use of GPUs with hierarchically low-rank approximations was discussed by C. Chen and P.-G. Martinsson [240].

Fast hierarchical solvers for more general structured matrices were considered by H. Pouransari, P. Coulier, and E. Darve [928].

3.16 ▪ Mixed precision

Since many modern computers offer the possibility to use several precisions, the sparse matrix community did not escaped the mixed precision wave. It is sometimes used for iterative refinement, see E.C. Carson and N.J. Higham [204, 205].

A. Buttari, J.J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov [187] used mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. J.D. Hogg and J.A. Scott [645] described a mixed-precision solver for symmetric systems. P.R. Amestoy, O. Boiteau, A. Buttari, M. Gerest, F. Jézéquel, J.Y. L'Excellent, and T. Mary [22] used mixed precision low rank approximations. M. Zounon, N.J. Higham, C. Lucas, and F. Tisseur [1168] studied the performance impact of precision reduction in sparse linear solvers.

3.17 ▪ Historical and bibliographical comments

We do not know who used the expression "sparse matrix" for the first time. An Internet search for "sparse matrix" did not give answers relative to mathematics before the end of the 1950s. The first occurrence with these words in the title was a paper by W. Orchard-Hays [883] in some conference proceedings in 1956.

The connection between sparse matrices and graphs was established at the end of the 1950s. It was considered by D. Rosenblatt [955] in 1957, and by F. Harary [616] in 1959. The relationship to Gaussian elimination was pointed out by S.V. Parter [906] in 1961.

The study of sparse matrices really started in the 1960s. The first works considered band matrices and envelope storage schemes. The envelope schemes and algorithms for storing sparse matrices are mainly due to A. Jennings, see [678]. Other storage schemes were proposed in the 1960s and 1970s. A.R. Curtis and J.K. Reid [298] in 1971, and F.G. Gustavson [587] in 1972 used what is now known as the compressed storage by rows (CSR).

The main reordering techniques to minimize the bandwidth or the fill-in became popular in the 1970s with the works of W.F. Tinney and J.W. Walker [1062] in 1967 and E.H. Cuthill and J. McKee [301] in 1969. Nested dissection was developed by A. George and his collaborators [502, 503] in the 1970s. These results were summarized in the book by A. George and J.W.H. Liu [511]. J.W.H. Liu studied several improvements of the minimum degree algorithm [512] in the 1980s. Spectral methods were proposed in the 1990s by A. Pothen, H.D. Simon and their collaborators [98].

An important tool to study the relations between Gaussian elimination for symmetric matrices and graphs is the elimination tree, see R.S. Schreiber [998] in 1982 and J.W.H. Liu [757, 760] in 1986-1990.

Efficient sparse techniques were developed through the work of I.S. Duff and J.K. Reid and their collaborators, starting in the 1970s. It gave rise to sparse codes in the Harwell Subroutine Library (HSL) for different types of matrices. An important development was the multifrontal method which was first derived for symmetric matrices [381], following ideas of the frontal method devised by B. Irons [666] in 1970 for finite element problems. A nice exposition of the principles of the method was given by J.W.H. Liu [761] in 1992. The method was later extended to nonsymmetic problems in different instances. I.S. Duff and J.K. Reid [382] considered the sparsity pattern of $A + A^T$ to construct the elimination tree in 1984. The parallel aspects of the multifrontal method were considered by I.S. Duff in [369] in 1986. In the 1990s, T.A. Davis and I.S. Duff introduced another extension of the multifrontal algorithm to nonsymmetric matrices [313]. This led to the codes UMFPACK and MUMPS.

The supernodal method was also first developed for symmetric problems. C.C. Ashcraft, R.G. Grimes, J.G. Lewis, B.W. Peyton, and H.D. Simon [52] proposed a left-looking method in 1987. A supernodal symbolic factorization to identify the supernodes was described by J.W.H. Liu, E.G. Ng, and B.W. Peyton [763] in 1993. E. Rothberg and A. Gupta [960, 961] showed in 1991-1993 that the method can be implemented as a right-looking algorithm quite similar to the multifrontal method. In 1993, E.G. Ng and B.W. Peyton [854] described a parallel left-looking algorithm. The generalization of the supernodal method to nonsymmetric matrices was far from being trivial. In 1995-1999, J.W Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Liu [333] introduced the idea of nonsymmetric supernodes and described a left-looking code named SuperLU. Parallel versions of this code were developed later.

In the last years much research was devoted to the implementation of sparse techniques on parallel computers. Details can be found in the review paper [315] by T.A. Davis, S. Rajamanickam, and W.M. Sid-Lakhdar in 2016. A list of available software for solving sparse linear systems is given in that paper.

A short history of sparse techniques for linear systems is given in [160, pp. 84-94].

4

Classical iterative methods

Almost every evening I make a new edition of the tableau, which is always easy to improve. With the monotony of the measuring business, this always makes a pleasant distraction; you also always immediately see if something doubtful has crept in, what still remains to be done, etc. I recommend this method for imitation. You will hardly ever again eliminate directly, at least not if you have more than two unknowns. The indirect method can be performed half asleep or you can think about other things when you use it.

– Carl Friedrich Gauss, translation of a letter to Christian Ludwig Gerling, December 26, 1823

An iterative method for solving a linear system $Ax = b$ constructs a sequence of vectors $\{x_k\}$, $k = 0, 1, \ldots$, which is expected to converge to the solution x, starting from a given vector x_0. The method is said to be convergent if $\lim_{k\to\infty} \|x - x_k\| = 0$.

The classical iterative methods studied in this chapter are not used too much anymore because they were superseded by Krylov methods that we will consider in the next chapters. Nevertheless, it is worth knowing their properties since they are sometimes used as preconditioners or smoothers in other methods.

Most classical iterative methods use a splitting of the nonsingular matrix A, denoted as

$$A = M - N,$$

where M is a nonsingular matrix. Then, the sequence x_k is defined by

$$Mx_{k+1} = Nx_k + b, \tag{4.1}$$

and x_0 is given. Clearly, if this method is convergent, it converges towards the unique solution of the linear system. We would like to find conditions for this sequence of vectors to converge.

Let $\varepsilon_k = x - x_k$ be the error at iteration k. Since $Mx = Nx + b$, we obtain

$$\begin{aligned} M(x - x_{k+1}) &= N(x - x_k), \\ \varepsilon_{k+1} &= M^{-1}N\varepsilon_k. \end{aligned}$$

Iterating this relation, we obtain the characterization of the error,

$$\varepsilon_{k+1} = (M^{-1}N)^{k+1}\varepsilon_0. \tag{4.2}$$

From relation (4.2), the iterative method defined by (4.1) converges for all starting vectors x_0 if and only if

$$\lim_{k\to\infty} (M^{-1}N)^k = 0.$$

A condition for the limit of matrix powers to be zero was given in Theorem 1.25 using the spectral radius ρ. We have the following fundamental result.

Theorem 4.1. *The iterative method defined by (4.1) converges for every x_0 if and only if*

$$\rho(M^{-1}N) < 1.$$

Proof. This results straightforwardly from Theorem 1.25. □

Unfortunately, it is usually not easy to check if $\rho(M^{-1}N) < 1$ since, for most problems, the eigenvalues of $M^{-1}N$ are not explicitly known. Hence, we have to rely mainly on sufficient conditions for convergence. In this chapter we describe a few well known iterative methods.

4.1 ▪ The Jacobi method

Even though this is not the simplest method one can think of (see Section 4.5), let us start with the Jacobi method. We split the matrix as $A = D + L + U$ where D is a diagonal matrix with the same diagonal entries as A, L (resp. U) is the strictly lower (resp. upper) triangular part of A. A simple choice for the iterative method (4.1) is to take $M = D$, $N = -(L + U)$. This method is known as the (point) Jacobi method. The iteration matrix of this method is usually denoted by $J(A)$,

$$J(A) = M^{-1}N = -D^{-1}(L + U). \tag{4.3}$$

It is straightforward to solve the linear system with matrix M in (4.1) since M is diagonal. Writing (4.1) componentwise, the components of x_{k+1} are given by

$$[x_{k+1}]_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{i,j}[x_k]_j\right). \tag{4.4}$$

A simple code for the Jacobi method is the following.

```
[x,nit,res] = jacobi(A,b,x0,nitmax,epss)
res = zeros(1,nitmax+1);
d = spdiags(A,0);
d1 = 1 ./ d;
nb = norm(b);
x = x0;
dx = d .* x;
r = b - A * x;
nit = 0;
resid = norm(r);
res(1) = resid;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 dx = r + dx;
```

```
 x = d1 .* dx;
 r = b - A * x;
 resid = norm(r);
 res(nit+1)= resid;
end % while
res = res(1,1:nit+1);
```

Sufficient convergence conditions for the Jacobi method are easily obtained.

Theorem 4.2. *Let A be a strictly diagonally dominant matrix. Then, the Jacobi method converges.*

Proof.

$$\left(D^{-1}(L+U)\right)_{i,j} = \frac{a_{i,j}}{a_{i,i}} \quad \forall i \neq j \text{ and } 0 \text{ otherwise.}$$

Hence,

$$\|D^{-1}(L+U)\|_\infty = \max_i \sum_{j\neq i} \frac{|a_{i,j}|}{|a_{i,i}|}.$$

The matrix A being strictly diagonally dominant, $\sum_{j\neq i} \frac{|a_{i,j}|}{|a_{i,i}|} < 1$, for all i. Then,

$$\|D^{-1}(L+U)\|_\infty < 1.$$

This is a sufficient condition for

$$\lim_{k\to\infty} J(A)^k = 0,$$

and the method converges. □

This result can be generalized from strictly diagonally dominant matrices to H-matrices.

Theorem 4.3. *Let A be a nonsingular H-matrix. Then, the Jacobi method converges.*

Proof. From Theorem 1.47 and Definition 1.33, there exists a diagonal matrix E with positive diagonal entries such that $E^{-1}AE$ is strictly diagonally dominant. The iteration matrix $J(A)$ can be written as,

$$J(A) = -D^{-1}(L+U) = -D^{-1}(A-D) = I - D^{-1}A.$$

The matrix $E^{-1}AE$ has the same diagonal as A and

$$J(E^{-1}AE) = I - D^{-1}(E^{-1}AE) = E^{-1}(I - D^{-1}A)E = E^{-1}J(A)E.$$

This is a consequence of $E^{-1}D^{-1} = D^{-1}E^{-1}$. Hence, $J(A)$ and $J(E^{-1}AE)$ are similar, and they have the same eigenvalues. The matrix $E^{-1}AE$ being strictly diagonally dominant, we have

$$\rho(J(A)) = \rho(J(E^{-1}AE)) < 1.$$

□

From Theorem 4.3, we can derive the following results which are useful in some practical cases.

Corollary 4.4. *Let A be a matrix having one of the two following properties,*

1) A is irreducibly diagonally dominant,

2) A is a nonsingular M-matrix.

Then, the Jacobi method converges.

Proof. Each of the two conditions implies that A is an H-matrix, and we can apply Theorem 4.3. □

When A is an L-matrix, we have a stronger result which also gives a characterization of a nonsingular M-matrix.

Theorem 4.5. *Let A be a nonsingular L-matrix. Then, A is a nonsingular M-matrix if and only if $\rho(J(A)) < 1$.*

Proof. If A is an M-matrix, Corollary 4.4 shows that $\rho(J(A)) < 1$. Conversely, we observe that, A being an L-matrix, we have

$$L \leq 0, \quad U \leq 0, \quad d_{i,i} > 0,$$

and

$$M^{-1} = D^{-1} \geq 0, \quad N = -(L + U) \geq 0.$$

By Definition 1.48, $A = M - N$ is a regular splitting with $\rho(M^{-1}N) < 1$. Hence, by Theorem 1.49, A is an M-matrix. □

The (point) Jacobi method is very well suited for vector and parallel computing. The method is inherently parallel since the computation of the components of x_{k+1} depends only on x_k. Therefore, the computation can be split into as many tasks as we need depending on how the matrix is stored and partitioned between computational units.

This method looks satisfactory but, unfortunately, its convergence can be very slow. For the Poisson model problem (that is, the finite difference discretization of $-\Delta u = f$ in the unit square) we are able to compute the spectral radius $\rho(J(A))$ since

$$D = 4I, \quad J(A) = I - \frac{1}{4}A.$$

The eigenvalues $\lambda_{p,q}$ of $J(A)$ are

$$\lambda_{p,q} = \frac{1}{2}(\cos p\pi h + \cos q\pi h), \quad p, q = 1, \ldots, m,$$

where $h = \frac{1}{m+1}$ is the mesh size, and the order of A is $n = m^2$. It is easy to see that $\max_{p,q} |\lambda_{p,q}| = |\lambda_{1,1}| = |\lambda_{m,m}|$ and that

$$\rho(J(A)) = \cos \pi h.$$

Unfortunately $\lim_{h \to 0} \rho(J(A)) = 1$. As we use more discretization points, the convergence slows down, and the number of iterations becomes larger and larger since

$$\rho(J(A)) = 1 - \frac{\pi^2 h^2}{2} + O(h^4).$$

An heuristic explanation for this slow convergence is obtained by considering a constant vector y, having all its components equal. Then, most components of Ay are zero. The Jacobi method

makes very small modifications to smooth modes of the error vector. The situation is similar for matrices arising from discretization of elliptic partial differential equations. For example, consider the problem of solving $Ax = 0$ with

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix},$$

a matrix of order 100. Starting from a random vector with components in $[0, 1]$, it takes about $28,000$ iterations to obtain $\max_i |[x_k]_i| < 10^{-6}$.

One may try to improve the performance using the Jacobi method in block form. Assuming that the matrix A is partitioned in blocks with square diagonal blocks, the definition of the method is straightforward. Note that at each iteration we have to solve independent linear systems whose matrices are the diagonal blocks. Regarding convergence, one can show the following result.

Theorem 4.6. *Let A be a strictly diagonally dominant matrix. Then, the block Jacobi method is convergent.*

Proof. The proof uses the same technique as the more general proof used for Theorem 4.20 below. □

Similarly to what we did for the point version, we can generalize this result to H-matrices.

Theorem 4.7. *Let A be a nonsingular H-matrix. Then, the block Jacobi method is convergent.*

Proof. The proof is the same as for Theorem 4.3. □

For symmetric matrices, results can be given both for point and block Jacobi methods.

Theorem 4.8. *Let A be a symmetric positive definite matrix. If $2D - A$ is positive definite, then the Jacobi method is convergent.*

Proof. This a straightforward application of the Householder-John theorem 1.57. with $Q = M + M^T - A = 2D - A$. □

Even the block Jacobi method is not really efficient. For the model problem, if the diagonal blocks are tridiagonal matrices corresponding to mesh lines, the spectral radius also tends to 1 when $h \to 0$.

A common technique to improve the convergence is relaxation, that is, taking an average between the last iterate and the previous one. For the Jacobi method it yields,

$$\begin{aligned} D\widetilde{x}_{k+1} &= -(L + U)x_k + b, \\ x_{k+1} &= \omega\widetilde{x}_{k+1} + (1 - \omega)x_k, \end{aligned}$$

where ω is a real parameter, $\omega > 0$. This method is called the relaxed Jacobi method. When $\omega = 1$, we recover the Jacobi method. We can eliminate $\widetilde{x}_{k+1}$ in these relations,

$$\begin{aligned} Dx_{k+1} &= -\omega(L + U)x_k + \omega b + (1 - \omega)Dx_k, \\ \frac{1}{\omega}Dx_{k+1} &= b - (L + U)x_k + \frac{1 - \omega}{\omega}Dx_k. \end{aligned} \tag{4.5}$$

The corresponding splitting is

$$M = \frac{1}{\omega}D, \quad N = \frac{1-\omega}{\omega}D - (L+U).$$

The iteration matrix is denoted as $J_\omega(A)$. For the relaxed Jacobi method we have results similar to those for the case $\omega = 1$.

Theorem 4.9. *Let A be a symmetric positive definite matrix. If $\frac{2}{\omega}D - A$ is positive definite, then the relaxed Jacobi method (4.5) converges.*

Proof. This is a straightforward application of Theorem 1.57. □

The problem is to know when $\frac{2}{\omega}D - A$ is positive definite. The answer to this question was given by D.M. Young in his book [1144].

Proposition 4.10. *Let A be a symmetric matrix with D positive definite. Then, $\frac{2}{\omega}D - A$ is positive definite if and only if $0 < \omega < \frac{2}{1-\mu_{\min}}$ where $\mu_{\min}$ is the smallest eigenvalue of $J(A)$.*

Proof. We first observe that $J(A) = -D^{-1}(L+L^T)$ is similar to $-D^{-\frac{1}{2}}(L+L^T)D^{-\frac{1}{2}}$ which is a symmetric matrix with real eigenvalues μ_i that we order as $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_n$. The diagonal entries of $J(A)$ are zero. Hence, trace$[J(A)] = \mu_1 + \mu_2 + \cdots + \mu_n = 0$. It implies that $\mu_n \geq 0$ and $\mu_1 \leq 0$. The matrix $\frac{2}{\omega}D - A$ is positive definite if and only if $D^{-\frac{1}{2}}(\frac{2}{\omega}D - A)D^{-\frac{1}{2}}$ is positive definite. But,

$$D^{-\frac{1}{2}}(\frac{2}{\omega}D - A)D^{-\frac{1}{2}} = \frac{2}{\omega}I - D^{-\frac{1}{2}}AD^{\frac{1}{2}} = (\frac{2}{\omega} - 1)I + D^{\frac{1}{2}}J(A)D^{-\frac{1}{2}},$$

because $D^{-1}A = I - J(A)$. The eigenvalues of $(\frac{2}{\omega} - 1)I + D^{\frac{1}{2}}J(A)D^{-\frac{1}{2}}$ are $\frac{2}{\omega} - 1 + \mu_i$, $i = 1, \ldots, n$. They are all positive if $\omega < \frac{2}{1-\mu_{\min}}$. □

One may ask what is the value of ω for which the spectral radius is minimum. Let μ_i^ω be the eigenvalues of $J_\omega(A)$

$$\begin{aligned} J_\omega(A) &= \omega D^{-1}(\frac{1-\omega}{\omega}D - (L+U)), \\ &= (1-\omega)I + \omega J(A), \\ \mu_i^\omega &= (1-\omega) + \omega\mu_i = 1 - \omega(1-\mu_i). \end{aligned}$$

We have to study $|\mu_i^\omega|$ as a function of ω, see Figure 4.1.

The optimal value ω_{opt} is given by the intersection of the graphs of the functions $1-\omega(1-\mu_n)$ and $\omega(1-\mu_1) - 1$. It yields

$$\omega_{opt} = \frac{2}{2 - (\mu_1 + \mu_n)}.$$

This tells us again that the method converges for $0 < \omega < \frac{2}{1-\mu_{min}}$. The downside of this method is that we need to know the extreme eigenvalues of $J(A)$ to compute the optimal parameter ω_{opt}. Unfortunately, in most cases, we do not know the eigenvalues. However, for the Poisson model problem,

$$\mu_1 = \cos(\pi h), \quad \mu_n = -\cos(\pi h).$$

It yields $\omega_{opt} = 1$, and the Jacobi method is optimal! This is the case for all matrices which have property A, see Definition 1.35.

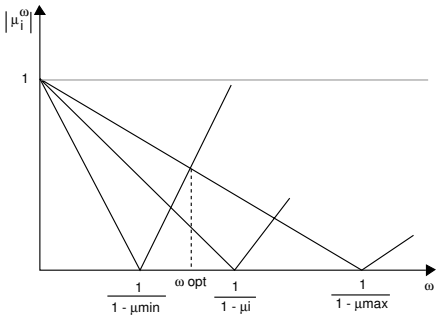


Figure 4.1. $|\mu_i^\omega|$ *as a function of* ω

4.2 ▪ The Gauss-Seidel and SOR methods

We observe that in the Jacobi method we only use x_k to compute all the components of x_{k+1}. We compute the components of x_{k+1} sequentially one at a time and it was tempting to use components of x_{k+1} as soon as they become available. We can also use a relation parameter ω.

The (point) successive overrelaxation (SOR) method is defined by

$$[x_{k+1}]_i = \frac{\omega}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j}[x_{k+1}]_j - \sum_{j=i+1}^{n} a_{i,j}[x_k]_j \right) + (1-\omega)[x_k]_i, \tag{4.6}$$

where ω is a real parameter. Writing (4.6) in matrix form with $A = D + L + U$, we have

$$(D + \omega L)x_{k+1} = \omega b - \omega U x_k + (1-\omega)Dx_k. \tag{4.7}$$

Taking $\omega = 1$ gives the Gauss-Seidel method. Relation (4.7) also defines the block SOR method if D, L and U are in block form. The iteration matrix is denoted by $\mathcal{L}_\omega$,

$$\mathcal{L}_\omega = \left(\frac{1}{\omega} D + L \right)^{-1} \left[\frac{1-\omega}{\omega} D - U \right].$$

A code for the SOR method is the following.

```
[x,nit,res] = sor(A,b,x0,nitmax,epss,om)
res = zeros(1,nitmax+1);
nb = norm(b);
n = size(A,1);
d = diag(A);
DD = spdiags(d,0,n,n);
L = tril(A,-1);
U = triu(A,1);
x = x0;
r = b - A * x;
resid = norm(r);
res(1) = resid;
DDom = DD + om * L;
nit = 0;
```

```
while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 rhs = om * b - om * U * x + (1 - om) * DD * x;
 x = DDom \ rhs;
 r = b - A * x;
 resid = norm(r);
 res(nit+1) = resid;
end % while
res = res(1,1:nit+1);
```

The relaxation parameter ω cannot take any real value since we have a necessary condition for the convergence of the method.

Proposition 4.11. *If the SOR method converges then,* $0 < \omega < 2$.

Proof. The iteration matrix can be written as

$$\mathcal{L}_\omega = (I + \omega D^{-1}L)^{-1}[(1-\omega)I - \omega D^{-1}U].$$

The matrix $I + \omega D^{-1}L$ is lower triangular with 1's on the diagonal, and $(1-\omega)I - \omega D^{-1}U$ is an upper triangular matrix with $1-\omega$ on the diagonal. Therefore, the determinant of the iteration matrix is

$$\det \mathcal{L}_\omega = (1-\omega)^n.$$

Since $\det \mathcal{L}_\omega$ is the product of the eigenvalues of $\mathcal{L}_\omega$, we obtain

$$|1-\omega|^n \leq \rho(\mathcal{L}_\omega)^n.$$

Therefore, $|1-\omega| \leq \rho(\mathcal{L}_\omega)$. If SOR converges, $\rho(\mathcal{L}_\omega) < 1$. This implies that $|1-\omega| < 1$, which yields

$$0 < \omega < 2.$$

□

Let us consider strictly diagonally dominant matrices.

Theorem 4.12. *Let A be a strictly diagonally dominant matrix and $0 < \omega \leq 1$. Then, the point SOR method converges for every x_0.*

Proof. Let $\varepsilon_k = x - x_k$ be the error vector. We have

$$[\varepsilon_{k+1}]_i = -\omega \sum_{j=1}^{i-1} \frac{a_{i,j}}{a_{i,i}} [\varepsilon_{k+1}]_j - \omega \sum_{j=i+1}^{n} \frac{a_{i,j}}{a_{i,i}} [\varepsilon_k]_j + (1-\omega)[\varepsilon_k]_i,$$

$$|\,[\varepsilon_{k+1}]_i| \leq \omega \sum_{j=1}^{i-1} \left|\frac{a_{i,j}}{a_{i,i}}\right| \cdot \|\varepsilon_{k+1}\|_\infty + \omega \sum_{j=i+1}^{n} \left|\frac{a_{i,j}}{a_{i,i}}\right| \cdot \|\varepsilon_k\|_\infty + (1-\omega)\|\varepsilon_k\|_\infty.$$

Assume $\|\varepsilon_{k+1}\|_\infty = \max_i |[\varepsilon_{k+1}]_i|$ is given for $i = \ell$. Then,

$$\left(1 - \omega \sum_{j=1}^{\ell-1} \left|\frac{a_{\ell,j}}{a_{\ell,\ell}}\right|\right) \|\varepsilon_{k+1}\|_\infty \leq \omega \sum_{j=\ell+1}^{n} \left|\frac{a_{\ell,j}}{a_{\ell,\ell}}\right| \cdot \|\varepsilon_k\|_\infty + (1-\omega)\|\varepsilon_k\|_\infty.$$

With $\omega > 0$, strict diagonal dominance implies

$$\omega \sum_{\substack{j=1 \\ j \neq \ell}}^{n} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| < \omega \;\Rightarrow\; \omega \sum_{j=\ell+1}^{n} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| < \omega \left(1 - \sum_{j=1}^{\ell-1} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| \right).$$

Then,

$$\begin{aligned} \left(1 - \omega \sum_{j=1}^{\ell-1} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| \right) \|\varepsilon_{k+1}\|_\infty &< \omega \left(1 - \sum_{j=1}^{\ell-1} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| \right) \|\varepsilon_k\|_\infty + (1-\omega)\|\varepsilon_k\|_\infty, \\ &< \left(1 - \omega \sum_{j=1}^{\ell-1} \left| \frac{a_{\ell,j}}{a_{\ell,\ell}} \right| \right) \|\varepsilon_k\|_\infty, \end{aligned}$$

and

$$\|\varepsilon_{k+1}\|_\infty < \|\varepsilon_k\|_\infty < \cdots < \|\varepsilon_0\|_\infty.$$

Hence, ε_k is a converging sequence with limit ε. But, since A is nonsingular, $A\varepsilon = 0$, and $\varepsilon = 0$. □

Corollary 4.13. *Let A be a strictly diagonally dominant matrix. Then, the point Gauss-Seidel method converges for every x_0.*

Proof. We just have to set $\omega = 1$. □

Let A be a nonsingular H-matrix and $0 < \omega \leq 1$. Theorem 4.12 shows that the (point) SOR and the Gauss-Seidel methods converge. However, a better upper bound for ω can be obtained for SOR. To establish that we need an auxiliary result.

Lemma 4.14. *Let $A = D + L + U$ be an H-matrix and $\widetilde{A}_\omega = \frac{1-|1-\omega|}{\omega}|D| - |L| - |U|$. Then, $\widetilde{A}_\omega^{-1} \geq 0$ if $0 < \omega < \frac{2}{1+\rho(|J(A)|)}$.*

Proof.

$$\begin{aligned} \widetilde{A}_\omega &= \frac{1-|1-\omega|}{\omega}|D| - |L| - |U|, \\ &= |D| \left(\frac{1-|1-\omega|}{\omega} I - |J(A)| \right). \end{aligned}$$

We first consider the case with $0 < \omega < 1$. Then, $\frac{1-|1-\omega|}{\omega} = 1$, and $\widetilde{A}_\omega = |D|(I - |J(A)|)$. Since A is an H-matrix, $\rho(|J(A)|) = \rho(J(M(A))) < 1$, see Chapter 1 for the definition of $M(A)$. Theorem 1.26 proves that $\widetilde{A}_\omega^{-1}$ exists and

$$\widetilde{A}_\omega^{-1} = (I + |J(A)| + \cdots)|D|^{-1} \geq 0.$$

Now, assume $1 < \omega < 2$. Then,

$$\widetilde{A}_\omega = |D|(\frac{2-\omega}{\omega} I - |J(A)|).$$

The matrix $\frac{2-\omega}{\omega} I - |J(A)|$ is nonsingular if

$$\frac{\omega}{2-\omega} \rho(|J(A)|) < 1.$$

It implies

$$\omega < \frac{2}{1+\rho(|J(A)|)}.$$

Theorem 1.26 shows that $\widetilde{A}_\omega^{-1} \geq 0$. □

Let us now consider the convergence of SOR for H-matrices.

Theorem 4.15. *Let A be a nonsingular H-matrix. The SOR method converges if*

$$0 < \omega < \frac{2}{1+\rho(|J(A)|)},$$

Proof. Let $A = D + L + U$ be a splitting of the matrix A. Then, $J(A) = I - D^{-1}A$. Let $M(A) = |D| - |L| - |U|$ and $J(M(A)) = I - |D|^{-1}(|D| - |L| - |U|)$. The diagonal entries of matrices $|J(A)|$ and $J(M(A))$ are equal to zero and the off-diagonal entries are $|\frac{a_{i,j}}{a_{i,i}}|$. Therefore $|J(A)| = J(M(A))$, but $M(A)$ is an M-matrix and by Theorem 4.3, we have

$$\rho(|J(A)|) = \rho(J(M(A))) < 1.$$

Let $M_\omega = \frac{1}{\omega}D + L$, $N_\omega = \frac{1-\omega}{\omega}D - U$. Then,

$$\mathcal{L}_\omega = M_\omega^{-1} N_\omega.$$

Denote

$$\widetilde{M}_\omega = \frac{1}{\omega}|D| - |L|, \quad \widetilde{N}_\omega = \frac{|1-\omega|}{\omega}|D| + |U|,$$

and

$$\widetilde{\mathcal{L}}_\omega = \widetilde{M}_\omega^{-1}\widetilde{N}_\omega, \quad \widetilde{A}_\omega = \widetilde{M}_\omega - \widetilde{N}_\omega = \frac{1-|1-\omega|}{\omega}|D| - |L| - |U|.$$

Clearly, $\widetilde{N}_\omega \geq 0$ and $|N_\omega| \leq \widetilde{N}_\omega$. Moreover,

$$|M_\omega^{-1}| \leq \left|(\frac{1}{\omega}D + L)^{-1}\right| = |(I + \omega D^{-1}L)^{-1}\omega D^{-1}|.$$

Since $D^{-1}L$ is strictly lower triangular, we have $(D^{-1}L)^n = 0$ and

$$(I + \omega D^{-1}L)^{-1} = I - \omega D^{-1}L + \cdots + (-1)^{n-1}(\omega D^{-1}L)^{n-1}.$$

Hence,

$$\begin{aligned} |M_\omega^{-1}| &\leq \omega|I - \omega D^{-1}L + \cdots + (-1)^{n-1}(\omega D^{-1}L)^{n-1}| \cdot |D^{-1}|, \\ &\leq \omega(I + \omega|D^{-1}L| + \cdots + \omega^{n-1}|D^{-1}L|^{n-1}) \cdot |D^{-1}|. \end{aligned}$$

Since D^{-1} is diagonal, $|D^{-1}L| = |D^{-1}| \cdot |L|$ and

$$0 \leq |M_\omega^{-1}| \leq (\frac{|D|}{\omega} - |L|)^{-1} = \widetilde{M}_\omega^{-1}.$$

It proves that $\widetilde{A}_\omega = \widetilde{M}_\omega - \widetilde{N}_\omega$ is a regular splitting. We conclude using Lemma 1.51. From that lemma, we know that $\widetilde{A}_\omega^{-1} \geq 0$ implies $\rho(\widetilde{M}_\omega^{-1}\widetilde{N}_\omega) < 1$. But, we have also shown that

$$|\mathcal{L}_\omega| = |M_\omega^{-1}N_\omega| \leq |M_\omega^{-1}||N_\omega| \leq \widetilde{M}_\omega^{-1}\widetilde{N}_\omega = \widetilde{\mathcal{L}}_\omega.$$

By Lemma 1.52, this shows that

$$\rho(\mathcal{L}_\omega) \leq \rho(\widetilde{\mathcal{L}}_\omega) < 1,$$

if $0 < \omega < \frac{2}{1+\rho(|J(A)|)}$. □

M. Neumann and R.S. Varga [852] have shown that this bound is sharp. They gave examples of H-matrices for which if $\omega = \frac{2}{1+\rho(|J(A)|)}$, then $\rho(\mathcal{L}_\omega) = 1$. These results imply that SOR converges for all H-matrices if and only if $0 < \omega \leq 1$.

If we now assume that the matrix A is symmetric, we have the following result.

Theorem 4.16. *Let A be a symmetric matrix with positive diagonal entries. Then, SOR converges for $0 < \omega < 2$ if and only if A is positive definite.*

Proof. This result is called the Ostrowski-Reich theorem [937, 887]. It is a straightforward consequence of the more general Householder-John theorem 1.57 since

$$Q = \frac{2-\omega}{\omega} D.$$

□

An important issue for SOR is the choice of the relaxation parameter ω. Obviously, we would like to choose the ω that minimizes the spectral radius of the iteration matrix $\rho(\mathcal{L}_\omega)$. This problem was solved for a large class of matrices by D.M. Young (1950) in his Ph.D. thesis [1143]. This is explained in great detail in his book [1144]. Young proves that, under some hypothesis, there exists an optimal value ω_b,

$$\omega_b = \frac{2}{1 + (1 - \rho(J(A))^2)^{\frac{1}{2}}}, \tag{4.8}$$

and $\rho(\mathcal{L}_{\omega_b}) = \omega_b - 1$. Young's theory relies on the fact that, for matrices with a *consistent ordering*, there is a relationship between the eigenvalues λ of $\mathcal{L}_\omega$ and μ of $J(A)$,

$$(\lambda + \omega - 1)^2 = \omega^2 \mu^2 \lambda.$$

It is unfortunate that we need $\rho(J(A))$ to compute the optimal ω_b because in many practical problems $\rho(J(A))$ is not known. Trying to find approximate values of ω_b was the subject of many research papers. Some researchers devised clever schemes to obtain an approximation of ω_b during the SOR iterations. For details, see the book by L.A. Hageman and D.M. Young [606]. For the Poisson model problem it is easy to compute ω_b since we know the eigenvalues of $J(A)$. We obtain

$$\omega_b = \frac{2}{1 + \sin(\pi h)} \quad \text{and} \quad \rho(\mathcal{L}_\omega) = \frac{1 - \sin(\pi h)}{1 + \sin(\pi h)}.$$

It yields $\rho(\mathcal{L}_{\omega_b}) = 1 - 2\pi h + O(h^2)$. One can see that for $\omega = 1$, that is, for the Gauss-Seidel method, $\rho(\mathcal{L}) = 1 - \pi^2 h^2 + O(h^4)$. For small values of h, SOR with ω_b gives a very large improvement over Gauss-Seidel. This explains the great interest for this method in the 1950s and 1960s. However, $\rho(\mathcal{L}_{\omega_b}) \to 1$ when $h \to 0$.

Figure 4.2 shows the $\log_{10}$ of the maximum norm of the error when solving the Poisson model problem with a 20×20 mesh for Jacobi, Gauss-Seidel and SOR with the optimal parameter. The convergence of SOR is much faster than that of the two other methods. For this problem Gauss-Seidel converges twice as fast as Jacobi.

Figure 4.3 shows the spectral radius of the SOR iteration matrix $\mathcal{L}_\omega$ as a function of ω for a 10×10 mesh. We observe that the spectral radius if non-differentiable for $\omega = \omega_b$ and that the curve is a straight line for $\omega > \omega_b$. If we do not know ω_b exactly, it is clearly better to over-relax.

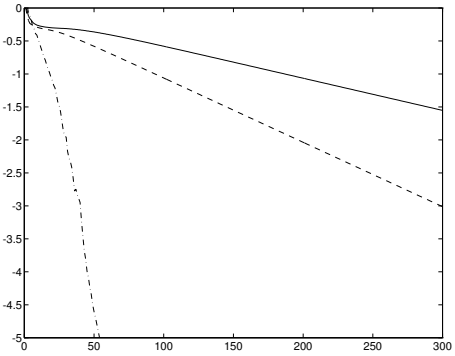


Figure 4.2. *Poisson problem with a* 20×20 *mesh, Jacobi (solid), Gauss-Seidel (dashed), SOR with optimal* ω *(dot-dashed)*

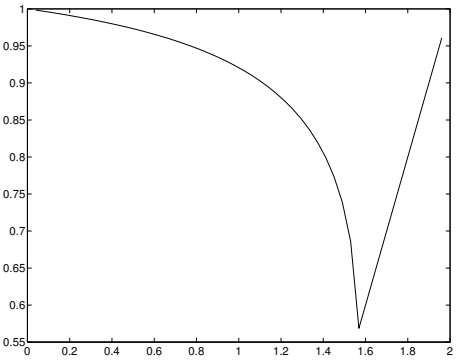


Figure 4.3. *Spectral radius of the SOR iteration matrix as a function of* ω *for the Poisson problem with a* 10×10 *mesh*

In many cases, Gauss-Seidel and SOR converge faster than Jacobi. However, the Jacobi method has the great advantage of being well suited for parallel computers. In Gauss-Seidel and SOR the components of the iterates have to be computed one after the other, and the two algorithms appear to be very sequential. For sparse matrices, one way to introduce parallelism is to consider the graph of the matrix and to assign colors to the nodes in such a way that a node is not connected to another node of the same color. The matrix can be reordered in block form such that the diagonal blocks are diagonal matrices. Then, the components of a given color can be computed in parallel. However, if we reorder the unknowns, the SOR iteration matrix may not have the same eigenvalues and the convergence of the method can be different. But, there are cases for which the convergence does not deteriorate.

Let us consider the simplest case. For problems arising from five point difference approximations, we can use the so-called *Red-Black* ordering. A small example of a square domain is shown on Figure 4.4. The Red points $\circ$ (resp. Black points x) are only connected to Black points (resp. Red points). Using this ordering, the linear system can be written in block form as

$$\begin{pmatrix} D_R & F \\ F^T & D_B \end{pmatrix} \begin{pmatrix} x^R \\ x^B \end{pmatrix} = \begin{pmatrix} b^R \\ b^B \end{pmatrix}, \tag{4.9}$$

where D_R and D_B are diagonal matrices. The matrix F represents the coupling between Red

and Black points. The Gauss-Seidel method can be written in block form as

$$\begin{aligned} D_R x_{k+1}^R &= b^R - F x_k^B, \\ D_B x_{k+1}^B &= b^B - F^T x_R^{k+1}. \end{aligned} \tag{4.10}$$

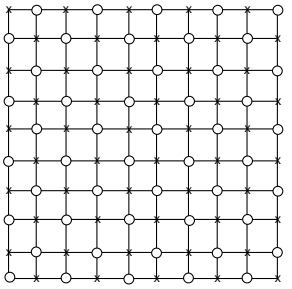


Figure 4.4. *The Red-Black ordering*

It is clear that we can first compute all the components of x_{k+1}^R simultaneously and then, all the components of x_{k+1}^B. Thus, this method can be used on a parallel computer. What are the convergence properties of the method defined by (4.10)? The interesting fact is that for a large class of matrices, the eigenvalues of the iteration matrix for (4.10) are the same as those for the natural ordering. Let us compute the eigenvalues of the iteration matrix,

$$\begin{pmatrix} 0 & -D_R^{-1}F \\ 0 & D_B^{-1}F^T D_R^{-1}F \end{pmatrix}.$$

Note that 0 is a multiple eigenvalue of this matrix and that there exists a set of linearly independent eigenvectors associated with the eigenvalue 0. Let

$$x = \begin{pmatrix} x^R \\ x^B \end{pmatrix},$$

be an eigenvector. Then

$$-D_R^{-1}Fx^B = \lambda x^R, \quad D_B^{-1}F^T D_R^{-1}Fx^B = \lambda x^B.$$

Therefore, λ is an eigenvalue of $D_B^{-1}F^T D_R^{-1}F$. Let

$$y = \begin{pmatrix} y^R \\ y^B \end{pmatrix},$$

be an eigenvector of the Jacobi iteration matrix for (4.9) corresponding to an eigenvalue μ,

$$D_R^{-1}Fy^B = \mu y^R, \quad D_B^{-1}F^T y^R = \mu y^B.$$

Eliminating y^R gives

$$D_B^{-1}F^T D_R^{-1}Fy^B = \mu^2 y^B.$$

It shows that μ^2 is an eigenvalue of the Gauss-Seidel iteration matrix. We note that the Jacobi iterates are independent of the ordering of the unknowns. Therefore, the Red-Black Jacobi iteration matrix has the same eigenvalues as that of the natural ordering Jacobi matrix. For the

Poisson model problem, we have proved that the squares of the Jacobi eigenvalues are eigenvalues of $\mathcal{L}$. Hence, the Red-Black Gauss-Seidel iteration matrix has the same eigenvalues as $\mathcal{L}$. This is more generally true for matrices having a consistent ordering in the terminology of D.M. Young [1144]. For more general problems with larger stencils or finite element problems, more than two colors have to be used to introduce parallelism. The influence of the ordering on the convergence of SOR was studied by L.M. Adams and H.F. Jordan [3].

For extensions of the SOR method, see [4, 604, 605].

4.3 ▪ The SSOR method

The results of the SOR method are dependent upon the ordering of the unknowns. The given ordering may not be well suited to the physical problem. In the 1950s appeared the idea of symmetrizing the iteration. This is done by doing one SOR sweep computing the components from 1 to n, and then another sweep computing the components from n to 1. This trick gives a symmetric iteration matrix. This method is known as SSOR (Symmetric SOR). An iteration is defined by

$$\begin{aligned}(D+\omega L)x_{k+\frac{1}{2}} &= \omega b + (1-\omega)Dx_k - \omega U x_k,\\ (D+\omega U)x_{k+1} &= \omega b + (1-\omega)Dx_{k+\frac{1}{2}} - \omega L x_{k+\frac{1}{2}}.\end{aligned} \tag{4.11}$$

To obtain the iteration matrix, we eliminate $x_{k+\frac{1}{2}}$. It yields

$$\begin{aligned}x_{k+1} = (\frac{1}{\omega}D+U)^{-1}(\frac{1-\omega}{\omega}D-L)(\frac{1}{\omega}D+L)^{-1}(\frac{1-\omega}{\omega}D-U)x_k\\ +(\frac{1}{\omega}D+U)^{-1}(\frac{2-\omega}{\omega})D(\frac{1}{\omega}D+L)^{-1}b.\end{aligned} \tag{4.12}$$

We have a splitting of $A = M - N$ with $M = \frac{\omega}{2-\omega}(\frac{1}{\omega}D+L)D^{-1}(\frac{1}{\omega}D+U)$ and $N = \frac{\omega}{2-\omega}(\frac{1}{\omega}D+L)D^{-1}(\frac{1-\omega}{\omega}D-L)(\frac{1}{\omega}D+L)^{-1}(\frac{1-\omega}{\omega}D-U)$. The iteration matrix denoted by $\mathcal{S}_\omega$ is

$$\mathcal{S}_\omega = (\frac{1}{\omega}D+U)^{-1}(\frac{1-\omega}{\omega}D-L)(\frac{1}{\omega}D+L)^{-1}(\frac{1-\omega}{\omega}D-U).$$

As above, for the Gauss-Seidel and SOR methods, it can be shown that if A is strictly diagonally dominant and if $0 < \omega \leq 1$, then the method converges. This result can be extended to H-matrices. G. Alefeld and R.S. Varga [11] proved the following stronger result.

Theorem 4.17. *Let A be a nonsingular H-matrix. Then, the SSOR method converges if*

$$0 < \omega < \frac{2}{1+\rho(|J(A)|)}.$$

Proof. The proof is similar to that of Theorem 4.15. □

A. Neumaier and R.S. Varga [851] proved the following result which gives insights on the convergence or divergence of the method. Let ν such that $0 \leq \nu < 1$, $\mathcal{H}_\nu = \{A \mid A$ is an H-matrix with $\rho(|J(A)|) = \nu\}$ and

$$\hat{\omega}(\nu) = \begin{cases} 2, & \text{if } 0 \leq \nu \leq \frac{1}{2},\\ \frac{2}{1+\sqrt{2\nu-1}}, & \text{if } \frac{1}{2} < \nu < 1.\end{cases}$$

For each matrix A in $\mathcal{H}_\nu$, for every ω with $0 < \omega < \hat{\omega}(\nu)$, $\rho(\mathcal{S}_\omega) < 1$. For every ω with $\omega \leq 0$ or $\omega > \hat{\omega}(\nu)$, there exists a matrix in $\mathcal{H}_\nu$ for which $\rho(\mathcal{L}_\omega) \geq 1$.

Of course, one can also define block SSOR methods. Let us now consider the case of symmetric positive definite matrices. The following result also holds for the block case.

Theorem 4.18. *Let A be a symmetric positive definite matrix with a positive definite diagonal D. Then, the SSOR method converges for $0 < \omega < 2$.*

Proof. This is a consequence of the Householder-John theorem 1.57, because M is symmetric and it is easy to see that

$$Q = 2M - A = \frac{\omega}{2-\omega}(A + 2LD^{-1}L^T).$$

Since D is positive definite, $LD^{-1}L^T$ is positive semi-definite and Q is positive definite. □

We have seen that the SOR iteration matrix is nonsymmetric even when A is symmetric and, consequently, it may have complex eigenvalues. This could lead to difficulties when one tries to accelerate this method. One nice property of the SSOR method is that $\mathcal{S}_\omega$ has real positive eigenvalues as shown in the next theorem.

Theorem 4.19. *Let A be a symmetric matrix with D positive definite. Then, the eigenvalues of $\mathcal{S}_\omega$ are real and non-negative.*

Proof.

$$\mathcal{S}_\omega = (\frac{1}{\omega}D + L^T)^{-1}(\frac{1-\omega}{\omega}D - L)(\frac{1}{\omega}D + L)^{-1}(\frac{1-\omega}{\omega}D - L^T).$$

Let $\overline{L} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$. Then,

$$\mathcal{S}_\omega = D^{-\frac{1}{2}}(\frac{1}{\omega}I + \overline{L}^T)^{-1}(\frac{1-\omega}{\omega}I - \overline{L})(\frac{1}{\omega}I + \overline{L})^{-1}(\frac{1-\omega}{\omega}I - \overline{L}^T)D^{\frac{1}{2}}.$$

We observe that as $\overline{L}$ is strictly lower triangular, $(\frac{1}{\omega}I + \overline{L})^{-1}$ is a polynomial in $\overline{L}$ (since by the Cayley-Hamilton theorem $\overline{L}^n = 0$), and it commutes with $(\frac{1-\omega}{\omega}I - \overline{L})$. We may rewrite the iteration matrix as

$$\mathcal{S}_\omega = D^{-\frac{1}{2}}(\frac{1}{\omega}I + \overline{L}^T)^{-1}(\frac{1}{\omega}I + \overline{L})^{-1}(\frac{1-\omega}{\omega}I - \overline{L})(\frac{1-\omega}{\omega}I - \overline{L}^T)D^{\frac{1}{2}}.$$

The matrix

$$\begin{aligned}\mathcal{S}'_\omega &= (\frac{1}{\omega}I + \overline{L}^T)D^{\frac{1}{2}}\mathcal{S}_\omega D^{-\frac{1}{2}}(\frac{1}{\omega}I + \overline{L}^T)^{-1},\\ &= (\frac{1}{\omega}I + \overline{L})^{-1}(\frac{1-\omega}{\omega}I - \overline{L})(\frac{1-\omega}{\omega}I - \overline{L}^T)(\frac{1}{\omega}I + \overline{L}^T),\end{aligned}$$

has the same eigenvalues as $\mathcal{S}_\omega$. If $G(\omega) = (\frac{1}{\omega}I + \overline{L})^{-1}(\frac{1-\omega}{\omega}I - \overline{L})$, then

$$\mathcal{S}'_\omega = G(\omega)G(\omega)^T,$$

and $\mathcal{S}_\omega$ is similar to a symmetric positive definite matrix. □

When A is symmetric positive definite, it can be shown (see D.M. Young [1145]) that there exists an optimal parameter but, unlike for SOR, the analytical form of this optimal value is not known. Nevertheless, Young [1145] showed that a "good" parameter is

$$\omega = \frac{2}{1 + \sqrt{2(1 - \rho(J(A)))^2}}. \tag{4.13}$$

With this parameter the convergence speed is almost twice as fast as with SOR. But, unfortunately, an SSOR iteration cost is twice that of an SOR iteration. Therefore, the benefits are not so obvious. However, there are two facts which are nice with SSOR. First, the number of iterations is less sensitive to the value of ω than it is with SOR and second, SSOR can be used in combination with acceleration methods. Figure 4.5 shows the spectral radius for the model problem as a function of ω for the Poisson model problem with a 10×10 mesh. One can see that the curve is smooth close to its minimum.

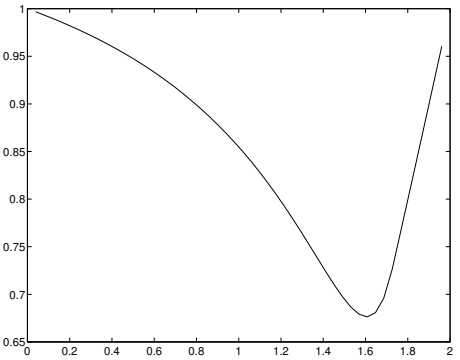


Figure 4.5. *Spectral radius of the* SSOR *iteration matrix for the Poisson problem on a* 10×10 *mesh*

Regarding parallel computing, the problem with SSOR is clearly the same as with SOR.

4.4 ▪ Alternating direction methods

This type of method originated in the 1950s for solving linear problems arising from elliptic partial differential equations (PDEs) on rectangles. If one uses methods like Jacobi or Gauss-Seidel, one space direction is favored over the other. A natural idea was to switch or alternate directions. These methods use a splitting,

$$A = D + H + V.$$

Originally, D was diagonal, and the matrix H (resp. V) represented what comes from the horizontal (resp. vertical) direction in the discretization stencil.

In 1955, D.W. Peaceman and H.H. Rachford [908] defined the following method which was named ADI (Alternating Direction Implicit),

$$\begin{aligned}
(\frac{1}{2}D + H + \rho_{k+1}I)x_{k+\frac{1}{2}} &= b - (\frac{1}{2}D + V - \rho_{k+1}I)x_k, \\
(\frac{1}{2}D + V + \rho'_{k+1}I)x_{k+1} &= b - (\frac{1}{2}D + H - \rho'_{k+1}I)x_{k+\frac{1}{2}},
\end{aligned} \tag{4.14}$$

where ρ_{k+1} and ρ'_{k+1} are real positive parameters. When this method is applied to a problem arising from a PDE on a rectangle, we have to solve as many tridiagonal systems as there are

rows in the mesh in the first step, and as many tridiagonal systems as there are columns in the mesh in the second step.

If $\rho'_{k+1} = \rho$, $\rho'_{k+1} = \rho'$, the method is called the stationary Peaceman-Rachford method. That method is a special case of a class of methods written as

$$\begin{aligned}(D + H + F)x_{k+\frac{1}{2}} &= b + (F - V)x_k,\\ (D + V + G)x_{k+1} &= b + (G - H)x_{k+\frac{1}{2}}.\end{aligned} \tag{4.15}$$

The decomposition $A = D + H + V$ and the matrices F, G can be chosen as we wish as long as $D + H + F$ and $D + V + G$ are nonsingular. Different choices of F and G lead to some variants,

- $F = \rho I - \frac{1}{2}D$, $G = \rho' I - \frac{1}{2}D$ gives the stationary Peaceman-Rachford method,
- $F = G = 0$ gives the block Jacobi alternating method,
- if $H = H_L + H_U$ where H_L (resp. H_U) is the lower (resp. upper) triangular part of H and the same for $V = V_L + V_U$. With $F = V_L$, $G = H_L$, this is the block Gauss-Seidel alternating method,
- $F = -H$, $G = -V$ this is the point Jacobi method.

For a matrix B, we denote by $I_i(B) = \{j \mid b_{i,j} \neq 0\}$ the index set of columns with nonzero entries in row i.

Theorem 4.20. *Let A be a strictly diagonally dominant matrix. Assume H and V have zero diagonals and*

$$I_i(V_U) \cap I_i(H + V_L) = \emptyset, \quad I_i(H_U) \cap I_i(V + H_L) = \emptyset$$

for all rows. Then, the block Gauss-Seidel alternating method is convergent.

Proof. The hypothesis on the index sets simply means that an entry of A is either in H or in V, but not split between the two. Let $\varepsilon_k = x - x_k$, then

$$\begin{aligned}(D + H + V_L)\varepsilon_{k+\frac{1}{2}} &= (-V_U)\varepsilon_k,\\ (D + V + H_L)\varepsilon_{k+1} &= (-H_U)\varepsilon_{k+\frac{1}{2}}.\end{aligned}$$

Therefore,

$$\varepsilon_{k+1} = (D + V + H_L)^{-1}H_U(D + H + V_L)V_U\varepsilon_k = T\varepsilon_k.$$

Let λ, u be an eigenvalue and the corresponding eigenvector of the iteration matrix T, $Tu = \lambda u$. Then,

$$\begin{aligned}(D + H + V_L)^{-1}V_U u &= v,\\ (D + V + H_L)^{-1}H_U v &= \lambda u.\end{aligned}$$

Let us show that $\|v\|_\infty < \|u\|_\infty$. We have,

$$(D + H + V_L)v = V_U u.$$

Componentwise, this can be written as

$$v_i = \sum_{j \in I_i(V_U)} \frac{a_{i,j}u_j}{a_{i,i}} - \sum_{j \in I_i(H+V_L)} \frac{a_{i,j}v_j}{a_{i,i}},$$

$$|v_i| \leq \|u\|_\infty \Big(\sum_{j \in I_i(V_U)} \frac{|a_{i,j}|}{|a_{i,i}|} \Big) + \|v\|_\infty \Big(\sum_{j \beta I_i(H+V_L)} \frac{|a_{i,j}|}{|a_{i,i}|} \Big).$$

Suppose that $\|u\|_\infty \leq \|v\|_\infty$, then

$$|v_i| \leq \|v\|_\infty \Big(\sum_{j \in I_i(H+V)} \frac{|a_{i,j}|}{|a_{i,i}|} \Big) < \|v\|_\infty.$$

This is a contradiction, therefore $\|v\|_\infty < \|u\|_\infty$. In the same way we can show that

$$\|\lambda u\|_\infty < \|v\|_\infty < \|u\|_\infty.$$

Hence $|\lambda| < 1$ and the method converges. □

With the same kind of proof it can be shown that the block Gauss-Seidel and Jacobi methods are convergent. If the diagonals of F and G are zero and the matrices $A + 2F$ and $A + 2G$ are strictly diagonally dominant, then the method (4.15) is convergent. Unfortunately, the stationary Peaceman-Rachford method is not covered by Theorem 4.20 because of the diagonal entries.

Theorem 4.21. *Let A be a nonsingular H-matrix. Assume H and V have zero diagonals and*

$$I_i(V_U) \cap I_i(H + V_L) = \emptyset, \quad I_i(H_U) \cap I_i(V + H_L) = \emptyset$$

for all rows. Then the block Gauss-Seidel alternating method converges.

Proof. As we did with some other methods, we can prove that there exists a diagonal matrix E with a positive diagonal such that

$$T(E^{-1}AE) = E^{-1}T(A)E.$$

This proves that the method converges. □

Let us now consider the stationary Peaceman-Rachford method. It can be written as,

$$\begin{aligned} (D + D_F + H)x_{k+\frac{1}{2}} &= (D_F - V)x_k + b, \\ (D + D_G + V)x_{k+1} &= (D_G - H)x^{k+1\frac{1}{2}} + b, \end{aligned} \tag{4.16}$$

where D_F and D_G are diagonal matrices with positive diagonals. To prove a convergence result, we use the same techniques as in Theorem 4.15.

Theorem 4.22. *Let A be a nonsingular H-matrix with a positive diagonal and let D_F and D_G be two diagonal matrices with positive diagonals. Assume that the diagonals of H and V are 0 and $I_i(H) \cap I_i(V) = \emptyset$ for all rows. Then, the method defined by (4.16) is convergent.*

Proof. Let

$$M_1 = D + D_F + H, \quad N_1 = D_F - V.$$

Then,

$$|N_1| = |D_F - V| \leq D_F + |V| = \widetilde{N}_1,$$
$$|M_1^{-1}| = |[I + (D + D_F)^{-1}H](D + D_F)^{-1}|.$$

But,

$$|(D + D_F)^{-1}H| \leq (D + D_F)^{-1}|H|.$$

By Lemma 1.52,

$$\rho((D+D_F)^{-1}H) \leq \rho((D+D_F)^{-1}|H|).$$

But $(D+D_F,\ |H|)$ is a regular splitting of $D+D_F-|H|$ which by hypothesis is an M-matrix. Therefore, $\rho((D+D_F)^{-1}|H|) < 1$. It implies that $[I+(D+D_F)^{-1}H]^{-1}$ exists and is equal to the sum of the series of matrices,

$$I-(D+D_F)^{-1}H+((D+D_F)^{-1}H)^2-\cdots$$

Bounding the absolute value of the series we obtain,

$$|\,[I+(D+D_F)^{-1}H]^{-1}| \leq (I-(D+D_F)^{-1}|H|).$$

Hence,

$$|M_1^{-1}| \leq (D+D_F-|H|)^{-1} = \widetilde{M}_1^{-1},$$

and

$$\widetilde{M}_1-\widetilde{N}_1 = D+D_F-|H|-D_F-|V| = D-|H|-|V|.$$

In the same way, if we denote

$$M_2 = D+D_G+V, \quad N_2 = D_G-H.$$

Then,

$$\begin{aligned} |M_2^{-1}| &\leq (D+D_G-|V|)^{-1} = \widetilde{M}_2^{-1}, \\ |N_2| &\leq D_G+|H| = \widetilde{N}_2, \\ \widetilde{M}_2-\widetilde{N}_2 &= D+D_G-|V|-|D_G|-|H| = D-|H|-|V|. \end{aligned}$$

By hypothesis $D-|H|-|V|$ is an M-matrix. Now, we first apply Lemma 1.53. Since $D-|H|-|V|$ is an M-matrix, its inverse is positive and the sequence defined by $\widetilde{M}_k, \widetilde{N}_k$ is convergent. Then, since $|M_i^{-1}| < \widetilde{M}_i^{-1}$, $|N_i| \leq \widetilde{N}_i$ $i=1,2$ by Lemma 1.54, the method (4.16) is convergent. □

As a consequence of Theorem 4.22, we obtain a convergence theorem for the Peaceman-Rachford method.

Corollary 4.23. *Let A be a matrix satisfying the hypothesis of Theorem 4.22. If ρ and ρ' are such that*

$$\rho \geq \frac{1}{2}\max_i a_{i,i},$$

$$\rho' \geq \frac{1}{2}\max_i a_{i,i},$$

then the stationary Peaceman-Rachford method is convergent.

□

Using the same techniques, G. Alefeld [10] proved that the Peaceman-Rachford method is convergent. The problem with that method is to find the optimal parameters. This question was only solved for the Poisson model problem, see D.M. Young [1144].

Regarding parallel computing, the situation is the same for the block Gauss-Seidel alternating method as for the ordinary block Gauss-Seidel method. It can even be worse because the data on the mesh has to be accessed by rows and by columns and that may cause problems on some

computers. The situation is simpler for the Peaceman-Rachford method since all the tridiagonal systems to be solved are independent of each other and can be solved in parallel mode.

Let us now look at the alternating Gauss-Seidel method when the matrix A is symmetric. To study this method, we need the following result which is interesting by itself.

Lemma 4.24. *Let A be a symmetric positive definite matrix and $A = M - N = P - Q$ be two splittings of A. Consider the iterative method defined by*

$$\begin{aligned} Mx_{k+\frac{1}{2}} &= Nx_k + b, \\ Px_{k+1} &= Qx_{k+\frac{1}{2}} + b. \end{aligned} \tag{4.17}$$

Assume $M^T + N$ and $P^T + Q$ are positive definite. Then, the method (4.17) is convergent.

Proof. This is a direct generalization of the Householder-John theorem 1.57. It can be proved using exactly the same technique. □

Theorem 4.25. *Let A be a symmetric positive definite matrix, H and V being symmetric. Assume $D + H$ and $D + V$ are positive definite. Then, the alternating block Gauss-Seidel method is convergent.*

Proof. In this case, we have

$$\begin{aligned} M = D + H + V_L, \quad N = -V_U, \quad M^T + N = D + H + V_L^T - V_U = D + H, \\ P = D + V + H_L, \quad Q = -H_U, \quad P^T + Q = D + V + H_L^T - H_U = D + V. \end{aligned}$$

□

Alternating directions methods are not frequently used any longer since they are closely related to finite difference discretizations on rectangular meshes. However, these methods can be used as preconditioners in a more general framework, see T.P. Mathew, P.L. Polyakov, G. Russo, and J. Wang [790].

More recently, some methods that look formally like alternating direction methods were developed. A method having a structure quite similar to ADI, although it has no geometrical meaning, is the Hermitian and skew-Hermitian splitting (HSS) iteration. Note that here HSS has nothing to see with the class of hierarchical matrices we have seen in Chapter 3. These methods were proposed by Z.-Z. Bai, G.H. Golub, and M.K. Ng [85, 86] for solving a large class of non-Hermitian linear systems. Let $H = (A + A^*)/2$ be the Hermitian part of A and $S = (A - A^*)/2$ be its skew-Hermitian part, the HSS method is defined by

$$\begin{aligned} (H + \alpha I)x_{k+\frac{1}{2}} &= (\alpha I - S)x_k + b, \\ (S + \alpha I)x_{k+1} &= (\alpha I - H)x_{k+\frac{1}{2}} + b, \end{aligned}$$

where α is a positive parameter. This method alternates between the Hermitian and skew-Hermitian parts of A. Of course, for this method to be efficient, we need fast methods to solve linear systems with the Hermitian and skew-Hermitian parts.

The convergence cannot be proved by the techniques used above for ADI methods because there is some overlap between the two parts of the splitting. Nevertheless, it was proved that if H is positive definite, the method converges for any choice of α. This type of method attracted much attention and many papers were devoted to its various aspects and applications, see [84, 83, 82]. A generalization of the method that can outperform the standard HSS method in some situations was proposed by M. Benzi [113]. It can be used as an effective preconditioner for certain linear systems in saddle point form.

4.5 ▪ Richardson methods

As we have seen above, the Jacobi method is quite simple. However, an even simpler method was introduced by L.F. Richardson [944]. In some cases it can be shown to reduce to the Jacobi iteration. Let α be a strictly positive real number. The method is defined by

$$x_{k+1} = x_k + \alpha(b - Ax_k). \tag{4.18}$$

This method is known as the stationary Richardson method. As other methods in this chapter, it is based upon a splitting of A,

$$\frac{1}{\alpha}x_{k+1} = \left(\frac{1}{\alpha}I - A\right)x_k + b.$$

Hence, the splitting gives $M = \frac{1}{\alpha}I$ and $N = \frac{1}{\alpha}I - A$. A necessary and sufficient condition for the method (4.18) to converge is $\rho(I - \alpha A) < 1$.

Theorem 4.26. *Let A be a symmetric and positive definite matrix and $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of A. Then, the stationary Richardson method (4.18) converges if and only if $\alpha < \frac{2}{\lambda_n}$.*

Proof. It is obvious that $|1 - \alpha\lambda_i| < 1$ for all i if and only if $\alpha < \frac{2}{\lambda_n}$. □

As with any other method depending upon a parameter, the problem is to find the optimal value of α giving the smallest spectral radius.

Theorem 4.27. *With the hypothesis of Theorem 4.26 the optimal value of α for the stationary Richardson method is $\alpha_{opt} = \frac{2}{\lambda_1+\lambda_n}$.*

Proof. The optimal value is determined by $1 - \alpha\lambda_1 = -(1 - \alpha\lambda_n)$. Hence, $\alpha_{opt} = \frac{2}{\lambda_1+\lambda_n}$ and

$$\rho(I - \alpha_{opt}A) = 1 - \frac{2}{\lambda_1 + \lambda_n}\lambda_1 = \frac{\lambda_n - \lambda_1}{\lambda_1 + \lambda_n} = \frac{\kappa(A) - 1}{\kappa(A) + 1},$$

where $\kappa(A) = \frac{\lambda_n}{\lambda_1}$ is the condition number of A. The closer $\kappa(A)$ is to 1, the faster the convergence. □

Of course, the downside of this method (as with relaxed Jacobi or SOR) is that we have to know the maximum and minimum eigenvalues of A to compute the optimal parameter. A straightforward way of generalizing the stationary Richardson method is to allow the parameter to change at every iteration,

$$x_{k+1} = x_k + \alpha_k(b - Ax_k). \tag{4.19}$$

The problem is again to choose the sequence of parameters α_k's. One choice is to minimize a norm of the residual at each iteration. Let μ be an integer. If A is symmetric and positive definite, $A^{-\mu}$ has the same properties. Let us introduce the norm

$$\|r_k\|_\mu^2 = (r_k, A^{-\mu}r_k),$$

and suppose we want to minimize $\|r_k\|_\mu^2$. Clearly,

$$r_{k+1} = r_k - \alpha_k A r_k.$$

Therefore,

$$(r_{k+1}, A^{-\mu}r_{k+1}) = (r_k, A^{-\mu}r_k) - 2\alpha_k(r_k, A^{1-\mu}r_k) + \alpha_k^2(r_k, A^{2-\mu}r_k).$$

Hence,

$$\alpha_k = \frac{(r_k, A^{1-\mu} r_k)}{(r_k, A^{2-\mu} r_k)}.$$

Of course, we need to be able to compute α_k, and usually one chooses $\mu = 1$. In that case, we obtain,

$$\alpha_k = \frac{(r_k, r_k)}{(r_k, Ar_k)}. \tag{4.20}$$

This method is called the gradient or steepest descent method. Note that it can be used for nonsymmetric problems. The code for this algorithm is quite simple as shown below.

```
function [x,nit,res] = steepest_descent(A,b,x0,nitmax,epss);
res = zeros(nitmax+1,1);
nb = norm(b);
x = x0;
r = b - A * x;
resid = norm(r);
res(1) = resid;
nit = 0;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 rr = r' * r;
 Ar = A * r;
 rar = r' * Ar;
 alpha = rr / rar;
 x = x + alpha * r;
 r = r - alpha * Ar;
 resid = norm(r);
 res(nit+1) = resid;
end % while
res = res(1,1:nit+1);
```

Note that `rr` could have been computed as `sqrt(resid)`.

Theorem 4.28. *Let A be a symmetric positive definite matrix. Then, the steepest descent method (4.19), (4.20) is convergent.*

Proof. We have

$$(r_{k+1}, A^{-1} r_{k+1}) = (r_k, A^{-1} r_k) - \frac{(r_k, r_k)^2}{(r_k, Ar_k)},$$

$$\frac{(r_{k+1}, A^{-1} r_{k+1})}{(r_k, A^{-1} r_k)} = 1 - \frac{(r_k, r_k)^2}{(r_k, Ar_k)(r_k, A^{-1} r_k)}.$$

We then apply the Kantorovich inequality which says that

$$\frac{(r_k, Ar_k)(r_k, A^{-1} r_k)}{(r_k, r_k)^2} \leq \left(\frac{\sqrt{\kappa(A)} + (\sqrt{\kappa A})^{-1}}{2} \right)^2.$$

Therefore,

$$\frac{(r_{k+1}, A^{-1} r_{k+1})}{(r_k, A^{-1} r_k)} \leq \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right)^2 < 1,$$

$$0 < \|r_{k+1}\|_{-1}^2 < \|r_k\|_{-1}^2,$$

and $r_k \to 0$. □

We observe that in this method we have the local orthogonality relation $(r_{k+1}, r_k) = 0$. The reader may wonder if the steepest descent method is optimal. Unfortunately, the answer is negative.

Let $\varepsilon_k = x - x_k$. Then,

$$\varepsilon_{k+1} = (I - \alpha_k A)\varepsilon_k,$$

and

$$\varepsilon_k = \prod_{i=0}^{k-1}(I - \alpha_i A)\varepsilon_0 = P_k(A)\varepsilon_0,$$

where $P_k(t)$ is a polynomial of degree k whose value is 1 at zero. Of course,

$$\|\varepsilon_k\| \leq \|P_k(A)\|\,\|\varepsilon_0\|,$$

and we may try to minimize $\|P_k(A)\|$ to get the smallest value of $\|\varepsilon_k\|$ for a given ε_0. Assume that A has ℓ distinct eigenvalues $\lambda_1, \ldots, \lambda_\ell$. Then, we consider the polynomial

$$\prod_{i=1}^{\ell}(\lambda_i - \lambda).$$

By the Cayley-Hamilton theorem, we have $\prod_{i=1}^{\ell}(\lambda_i I - A) = 0$. If we choose

$$\alpha_i = \frac{1}{\lambda_{i+1}},$$

we have $P_\ell(A) = 0$ and $\varepsilon_\ell = 0$. Of course, as we said before, usually we do not know the eigenvalues of A. But, if A is symmetric, there exists an orthogonal matrix Q and a diagonal matrix Λ such that $A = Q^T \Lambda Q$. Then, $P_k(A) = Q^T P_k(\Lambda) Q$, and

$$\|P_k(A)\| = \|Q^T P_k(\Lambda) Q\| = \|P_k(\Lambda)\| = \max_i |P_k(\lambda_i)|.$$

An upper bound of $\|P_k(A)\|$ can be found by considering that, if for all i, $\lambda_i \in [a, b]$, then

$$\max_i |P_k(\lambda_i)| \leq \max_{\lambda \in [a,b]} |P_k(\lambda)|.$$

Therefore,

$$\|\varepsilon_k\| \leq \max_{\lambda \in [a,b]} |P_k(\lambda)|\,\|\varepsilon_0\|,$$

and the problem amounts to find the polynomial of degree k with $P_k(0) = 1$ minimizing $\max_{\lambda \in [a,b]} |P_k(\lambda)|$. The solution to this problem with Chebyshev polynomials was given in Chapter 1,

$$P_k(\lambda) = \frac{C_k\left(\frac{a+b-2\lambda}{b-a}\right)}{C_k\left(\frac{a+b}{b-a}\right)}.$$

With this choice, we obtain

$$\|\varepsilon_k\| \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k \|\varepsilon_0\|,$$

if

$$\frac{1}{\alpha_i} = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2i-1}{k}\frac{\pi}{2}\right), \quad 1 \le i \le k.$$

The coefficients α_i's depend on k. Usually, one chooses an integer d and uses cyclically the parameters α_k for $k = 1, \ldots, d$. This method is clearly better than the stationary Richardson method regarding theoretical convergence. Unfortunately, it was noticed very early that it is very sensitive to rounding errors. The potential instability is linked to the order that is used for the parameters. R.S. Anderssen and G.H. Golub [36] studied an ordering due to V.I. Lebedev and S.A. Finoguenov [728] for which one can show that the method is stable. The α_k's are used in the order defined by the permutation

$$\chi_k = (i_1, i_2, \ldots, i_k).$$

Suppose $d = 2^p$, then $\chi_1 = 1$. If $\chi_{2^{p-1}} = (j_1, j_2, \ldots, j_{2^{p-1}})$, we take,

$$\chi_{2^p} = (j_1, 2^p + 1 - j_1, j_2, 2^p + 1 - j_2, \ldots, j_{2^{p-1}}, 2^p + 1 - j_{2^{p-1}}).$$

For example,

$$\chi_2 = (1, 2), \quad \chi_4 = (1, 4, 2, 3), \quad \chi_8 = (1, 8, 4, 5, 2, 7, 3, 6),$$

$$\chi_{16} = (1, 16, 8, 9, 4, 13, 5, 12, 2, 15, 7, 10, 3, 14, 6, 11).$$

The problem of finding the optimal polynomial when A is not positive (or negative) definite was solved by C. De Boor and J.R. Rice [322]. G. Opfer and G. Schober [881] studied the problem when A is nonsymmetric.

This kind of Richardson method can be extended in two ways. First, one can introduce a nonsingular matrix M and define

$$Mx_{k+1} = Mx_k - \alpha(b - Ax_k). \tag{4.21}$$

Since we have shown above that the convergence depends on the condition number, it is natural to choose M to minimize $\kappa(M^{-1}A)$. M is called the preconditioning matrix or preconditioner. The second possible generalization is to look for methods using more than one previous iterate.

4.6 ▪ Acceleration techniques

Let A be a symmetric matrix. Consider a convergent splitting method,

$$Mx_{k+1} = Nx_k + b,$$

or

$$x_{k+1} = Bx_k + c,$$

where $B = M^{-1}N$, $c = M^{-1}b$, and $\rho(B) < 1$. We have seen that $\varepsilon_k = B^k\varepsilon_0$. To accelerate the convergence, a common technique is to average the iterates x_k to obtain another sequence which, hopefully, will converge faster. Let

$$y_k = \sum_{\ell=0}^{k} \alpha_{\ell,k} x_\ell.$$

We require that $\sum_{\ell=0}^{k} \alpha_{\ell,k} = 1$, since if $x_\ell = x$ for all ℓ, we need to have $y_k = x$. Let

$$\eta_k = x - y_k = \sum_{\ell=0}^{k} \alpha_{\ell,k}(x - x_\ell) = \sum_{\ell=0}^{k} \alpha_{k,\ell} B^\ell \varepsilon_0 = P_k(B)\varepsilon_0,$$

where P_k is a polynomial of degree k with $P_k(1) = 1$. Once again, we would like to minimize $\|P_k(B)\|$. Assume that B has real eigenvalues belonging to $[-\rho, \rho]$, $\rho > 0$. The solution of this minimization problem is

$$P_k(t) = \frac{C_k(t/\rho)}{C_k(1/\rho)}.$$

When P_k is known, one can compute the coefficients $\alpha_{\ell,k}$, but it is better to use the recurrence relations of the Chebyshev polynomials to avoid storing all the vectors x_ℓ. We know that

$$C_k(t/\rho) = C_k(1/\rho)P_k(t),$$

but,

$$C_{k+1}(x) = 2xC_k(x) - C_{k-1}(x).$$

Therefore,

$$C_{k+1}\left(\frac{1}{\rho}\right)P_{k+1}(t) = \frac{2t}{\rho}C_k\left(\frac{1}{\rho}\right)P_k(t) - C_{k-1}\left(\frac{1}{\rho}\right)P_{k-1}(t).$$

With $\eta_k = P_k(B)\varepsilon_0$ and post multiplying by ε_0,

$$C_{k+1}\left(\frac{1}{\rho}\right)\eta_{k+1} = \frac{2}{\rho}C_k\left(\frac{1}{\rho}\right)B\eta_k - C_{k-1}\left(\frac{1}{\rho}\right)\eta_{k-1},$$

or

$$y_{k+1} - x = \frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})}(By_k - x + c) - \frac{C_{k-1}(\frac{1}{\rho})}{C_{k+1}(\frac{1}{\rho})}(y_{k-1} - x).$$

Noticing that

$$\left[\frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})} - \frac{C_{k-1}(\frac{1}{\rho})}{C_{k+1}(\frac{1}{\rho})}\right]x = x,$$

we obtain,

$$y_{k+1} = \frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})}(By_k + c) - \frac{C_{k-1}(\frac{1}{\rho})}{C_{k+1}(\frac{1}{\rho})}y_{k-1}.$$

Therefore, the vectors y_k can be computed without any knowledge of the x_ℓ's. A further simplification can be achieved since

$$y_{k+1} = \frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})}(By_k + c - y_{k-1}) + \left(\frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})} - \frac{C_{k-1}(\frac{1}{\rho})}{C_{k+1}(\frac{1}{\rho})}\right)y_{k-1}.$$

This can be written in a simpler way as

$$y_{k+1} = \omega_{k+1}(By_k + c - y_{k-1}) + y_{k-1},$$

with

$$\omega_{k+1} = \frac{2C_k(\frac{1}{\rho})}{\rho C_{k+1}(\frac{1}{\rho})}, \quad \omega_1 = 1.$$

But,

$$\omega_{k+1} = \frac{2C_k(\frac{1}{\rho})}{2C_k(\frac{1}{\rho}) - \rho C_{k-1}(\frac{1}{\rho})} = \frac{1}{1 - \frac{\rho C_{k-1}(\frac{1}{\rho})}{2C_k(\frac{1}{\rho})}} = \frac{1}{1 - \frac{\rho^2}{4}\omega_k}.$$

This shows that we do not need to compute the Chebyshev polynomials to obtain the coefficients of the method. This algorithm can be rewritten as

$$y_{k+1} = \omega_{k+1}(z_k + y_k - y_{k-1}) + y_{k-1}, \tag{4.22}$$

with

$$Mz_k = r_k = b - Ay_k.$$

The method is known as the Chebyshev semi-iterative method, see G.H. Golub and R.S. Varga [550, 551]. When the eigenvalues belongs to $[a, b]$, straightforward modifications show that the method becomes

$$y_{k+1} = \frac{\omega_{k+1}}{2-(a+b)}[2z_k + (2-(a+b))(y_k - y_{k-1})] + y_{k-1},$$

$$Mz_k = r_k,$$

with $y_0 = x_0$, $y_1 = x_1$, and

$$\omega_{k+1} = \frac{1}{1 - \frac{\omega_k}{4w^2}}, \quad \omega_2 = \frac{2w^2}{2w^2 - 1}, \quad \omega_1 = 1, \quad w = \frac{2-(a+b)}{b-a}.$$

If ω_{k+1} is kept constant and equal to $2/(\sqrt{1-\rho(B)^2}$, the method is known as the second-order Richardson iteration, see [539].

A Chebyshev code to accelerate the Jacobi iteration is the following.

```
function [x,nit,res] = chebyshev_jacobi(A,b,x0,nitmax,epss)
nb = norm(b);
res = zeros(1,nitmax+1);
M = diag(diag(A));
N = M - A;
eigMN = eig(full(inv(M) * N));
lmin = min(eigMN);
lmax = max(eigMN);
w = coeff(nitmax+1,lmin,lmax);
x = x0;
x_old = x;
r = b - A * x;
res(1) = norm(r);
x = M \ (r + M * x);
r = b - A * x;
resid = norm(r);
res(2) = resid;
ab = 2 - (lmin + lmax);
nit = 2;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 z = M \ r;
 xnew = (w(nit) / ab) * ( 2 * z + ab * (x - x_old)) + x_old;
 x_old = x;
 x = xnew;
 r = b - A * x;
```

```
 resid = norm(r);
 res(nit) = resid;
end % while
res = res(1,1:nit+1);
end % function

function w = coeff(k,lmin,lmax)
% coefficients of the Chebyshev semi iterative method
z = (2 - (lmin + lmax)) / (lmax - lmin);
z2 = z * z;
w = zeros(k,1);
w(1) = 1;
w(2) = 2 * z2 / (2 * z2 - 1);
for i = 3:k
 w(i) = 1 / (1 - w(i-1) / (4 * z2));
end
end % function
```

Other semi-iterative methods and their theory were studied by R.S. Varga and his collaborators, see [394, 398, 399, 400, 1096].

When the matrix A is nonsymmetric the parameters can be estimated dynamically by enclosing the eigenvalues in ellipses (see T.A. Manteuffel [779, 780]), although the Chebyshev polynomials over ellipses are not necessarily optimal.

4.7 ▪ Stability of classical iterative methods

The stability of classical iterative methods has not been completely studied. There are examples of well conditioned matrices for which, because of rounding errors, SOR may diverge or stagnate when it is supposed to converge. For methods derived from a splitting of A, an error analysis was done by N.J. Higham [631, 633]. Let

$$Mx_{k+1} = Nx_k + b - \xi_k,$$

be the computed iterates where the last term stands for the rounding errors. For the methods we have been studied,

$$|\xi_k| \le cu(|M|\,|x_{k+1}| + |N|\,|x_k| + |b|) = \mu_k,$$

where c is a constant and u is the unit roundoff. Then, the error satisfies

$$\varepsilon_{k+1} = (M^{-1}N)^{k+1}\varepsilon_0 + \sum_{i=0}^{k}(M^{-1}N)^i M^{-1}\xi_{k-i}.$$

From this relation, normwise bounds can be derived, see [631, 633]. Let

$$\gamma_x = \sup_k \frac{\|x_k\|}{\|x\|},$$

and $q = \|M^{-1}N\|_\infty < 1$. Then,

$$\|\varepsilon_{k+1}\|_\infty \le \|(M^{-1}N)^{k+1}\varepsilon_0\|_\infty + cu(1+\gamma_x)(\|M\|_\infty + \|N\|_\infty)\|x\|_\infty \frac{\|M^{-1}\|_\infty}{1-q}.$$

Componentwise bounds can also be obtained. Let

$$\theta_x = \sup_k \max_i \frac{(|M| + |N|)|x_k|)_i}{(|M| + |N|)|x|)_i},$$

and $c(A)$ be the smallest ϵ such that

$$\sum_{i=0}^{\infty} |(M^{-1}N)^i M^{-1}| \leq \epsilon |A^{-1}|.$$

Then,

$$|\varepsilon_{k+1}| \leq |(M^{-1}N)^{k+1}\varepsilon_0| + cu(1 + 2\theta_x)c(A)|A^{-1}|(|M| + |N|)|x|.$$

If $M^{-1} \geq 0, N \geq 0$ then, $c(A) = 1$. This is the case if $M - N$ is a regular splitting of A. From this bound, we can deduce, for instance, that if A is an M-matrix, the Jacobi and Gauss-Seidel methods are componentwise forward stable. For a backward error analysis, see N.J. Higham [631, 633]. The stability of the steepest descent method was studied by J.A.M. Bollen [135]. The algorithm is backward stable as long as the condition number is not too large.

The rounding properties of the second-order Richardson method were studied by G.H. Golub [539]; see also [545] with M.L. Overton.

4.8 ▪ Numerical experiments

Let us do numerical experiments with some of the matrices that we have already used in Chapter 2. Their characteristics are described in the Appendix. Table 4.1 shows the spectral radiuses of the iteration matrices for Jacobi, Gauss-Seidel, and SOR with a parameter ω computed as in formula (4.8). However, first in some cases $\rho(J) > 1$ and ω becomes a complex number, and secondly the hypothesis for applying Young's theory may not be satisfied, and even if ω can be computed it may not be the optimal value of the relaxation parameter, see, for instance, the matrix `pde_225` for which SOR is diverging with the computed ω.

In this set of matrices there are many cases for which the spectral radiuses are larger than 1, which means that the methods do not converge. In cases where everything is fine, the spectral radius of SOR is generally smaller than those of Jacobi and Gauss-Seidel. Remember that the spectral radius describes only the asymptotic behavior of the method. Even when the spectral radiuses are smaller than 1, many of them are very close to 1 and the convergence is slow.

Figure 4.6 shows the spectral radius of the SOR iteration matrix as a function of ω for two matrices, one (`fs_680_1c`) for which Young's theory apply and one (`steam2`) for which it does not apply. There is an optimal value of ω for `steam2` but is it not the one computed with formula (4.8).

Table 4.2 displays the number of iterations to obtain $\|r_k\|/\|b\| \leq 10^{-10}$ as well as the absolute ℓ_2 norms of the residual and the error at convergence. The initial vector is zero and the maximum number of iterations is $20,000$. SOR is ran with the value of the parameter given by formula (4.8). On this set of matrices, when both methods converge, the number of iterations of Gauss-Seidel is smaller than for Jacobi, approximately by a factor of two. When the optimal value of ω can be computed the number of iterations of SOR is much smaller. The differences of the residual and error norms can be understood by looking at the norms of the matrix and its inverse; for instance, for `watt1`, $\|A\| = 1$, but $\|A^{-1}\| = 4.3596\ 10^9$.

Table 4.3 displays the relative ℓ_2 norms of the residual and the error at convergence.

Tables 4.4 and 4.5 shows the residual and error norms for the symmetric Gauss-Seidel (SGS), SSOR with ω computed using formula (4.13), and steepest descent. The number of iterations for

Table 4.1. *Spectral radiuses*

matrix	Jacobi	Gauss-Seidel	SOR	ω
pde_225	0.988761	0.977649	3.42333	1.73988
gre_343	-	-	-	-
jpwh_991	0.979722	0.959915	0.746060	1.66616
jagmesh1	5.90560	12.0026	-	-
bfwa782	1.21863	1.01695	1.09245	1.34675
jagmesh2	5.96954	12.6111	-	-
fs_680_1c	0.969742	0.940407	0.617856	1.60755
fs_680_1	0.969742	0.940407	0.617856	1.60755
sherman1	0.999690	0.999380	0.951418	1.95142
nos3	1.63139	0.999840	-	-
olm1000	4.24458	-	-	-
steam2	0.998660	0.996894	5.25310	1.90157
1138bus	0.999996	0.999992	0.995007	1.99430
steam1	1.18377	1.12561	6.10033	1.42723
nos7	1.00000	1.00000	0.999648	1.99965
fs_183_6	0.765285	0.603918	0.944965	1.21677
bcsstk20	2.49830	1.00000	-	-
mcfe	37.9076	$2.91387\ 10^3$	-	-
lnsp	-	-	-	-
bcsstk01	1.10145	0.996914	-	-
pde2961	0.987160	0.974485	1.19027	1.72453
add20	0.999662	0.999323	0.953970	1.94929
watt1	0.998696	0.997395	0.902872	1.90287

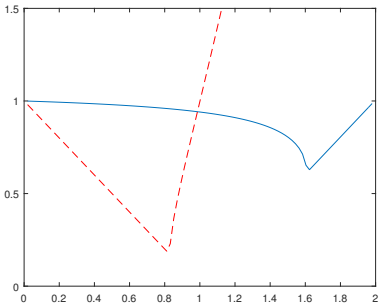


Figure 4.6. $\rho(\mathcal{L}_\omega)$ *as a function of* ω, `fs_680_1c` *(plain) and* `steam2` *(dashed)*

SGS is smaller than for Gauss-Seidel, but remember that the cost of an iteration of SGS is twice that of a Gauss-Seidel iteration. SSOR with this computation of ω does not always give a smaller number of iterations than SOR.

Table 4.2. *Number of iterations, and absolute residual and error norms,* $\epsilon = 10^{-10}$

matrix	Jacobi	Gauss-Seidel	SOR	ω
pde_225	1929	975	-	1.73988
residual	$1.60269\ 10^{-9}$	$1.58510\ 10^{-9}$	NaN	
error	$4.15933\ 10^{-10}$	$3.56241\ 10^{-10}$	NaN	
jpwh_991	1063	536	82	1.66616
	$1.20292\ 10^{-9}$	$1.17811\ 10^{-9}$	$1.04851\ 10^{-9}$	
	$9.57916\ 10^{-9}$	$8.31718\ 10^{-9}$	$2.82079\ 10^{-9}$	
pde2961	1854	954	-	1.72453
	$1.55039\ 10^{-9}$	$1.53695\ 10^{-9}$	NaN	
	$3.83110\ 10^{-8}$	$3.58453\ 10^{-8}$	NaN	
fs_680_1c	826	302	57	1.60755
	$2.28946\ 10^{-9}$	$2.33863\ 10^{-9}$	$41.57680\ 10^{-9}$	
	$1.32638\ 10^{-9}$	$1.33854\ 10^{-7}$	$1.62057\ 10^{-9}$	
add20	20000	20000	535	1.94929
	$8.30792\ 10^{-17}$	$1.49912\ 10^{-20}$	$8.94053\ 10^{-21}$	
	$4.36585\ 10^{-13}$	$1.34988\ 10^{-16}$	$1.50508\ 10^{-18}$	
fs_680_1	739	269	48	1.60755
	$7.93717\ 10^{3}$	$7.57164\ 10^{3}$	$5.03969\ 10^{3}$	
	$1.94921\ 10^{-8}$	$1.02555\ 10^{-6}$	$1.25637\ 10^{-7}$	
sherman1	20000	20000	491	1.95142
	$1.74289\ 10^{-5}$	$4.31042\ 10^{-8}$	$3.78678\ 10^{-10}$	
	$4.43846\ 10^{-2}$	$9.00556\ 10^{-5}$	$1.30482\ 10^{-8}$	
nos3	-	20000	-	$0.751473 - 0.96862i$
	NaN	$1.41374\ 10^{-2}$	-	
	Inf	$7.38218\ 10^{-1}$	-	
watt1	15535	7283	248	1.90287
	$4.23810\ 10^{-9}$	$4.22855\ 10^{-9}$	$3.92756\ 10^{-9}$	
	3.47374	12.2643	$5.40654\ 10^{-1}$	

Table 4.3. *Number of iterations, and relative residual and error norms,* $\epsilon = 10^{-10}$

matrix	Jacobi	Gauss-Seidel	SOR	ω
pde_225	1929	975	-	1.73988
residual	$9.91975\ 10^{-11}$	$9.81090\ 10^{-11}$	NaN	
error	$4.13707\ 10^{-11}$	$3.54335\ 10^{-11}$	NaN	
jpwh_991	1063	536	82	1.66616
	$9.98970\ 10^{-11}$	$9.78364\ 10^{-11}$	$8.70743\ 10^{-11}$	
	$3.04292\ 10^{-11}$	$2.64204\ 10^{-10}$	$8.96053\ 10^{-11}$	
pde2961	1854	954	-	1.72453
	$9.92392\ 10^{-11}$	$9.83790\ 10^{-11}$	NaN	
	$7.04052\ 10^{-10}$	$6.58738\ 10^{-10}$	NaN	
fs_680_1c	826	302	57	1.60755
	$9.69694\ 10^{-11}$	9.90519_10^{-11}	$6.67847\ 10^{-11}$	
	$5.08643\ 10^{-11}$	$5.13307\ 10^{-9}$	$6.21461\ 10^{-11}$	
add20	20000	20000	535	1.94929
	$8.37839\ 10^{-7}$	$1.51184\ 10^{-10}$	$9.01636\ 10^{-11}$	
	3.9970710^{-6}	$1.23586\ 10^{-9}$	$1.37794\ 10^{-11}$	
fs_680_1	739	269	48	$1.60755e$
	$9.86898\ 10^{-11}$	$9.41449\ 10^{-11}$	$6.26629\ 10^{-11}$	
	$7.47487\ 10^{-10}$	$3.93281\ 10^{-8}$	$4.81798\ 10^{-9}$	
sherman1	20000	20000	491	1.95142
	$4.56024\ 10^{-6}$	$1.12782\ 10^{-8}$	$9.90806\ 10^{-11}$	
	$3.61139\ 10^{-4}$	$7.32746\ 10^{-7}$	$1.06168\ 10^{-10}$	
nos3	-	20000	-	$0.751473 - 0.96862i$
	NaN	$4.44156\ 10^{-5}$	-	
	Inf	$2.38259\ 10^{-2}$	-	
watt1	15535	7283	248	1.90287
	$9.99658\ 10^{-11}$	$9.97405\ 10^{-11}$	$9.26411\ 10^{-11}$	
	$1.34398\ 10^{-10}$	$4.74501\ 10^{-9}$	$2.09177\ 10^{-10}$	

Table 4.4. *Number of iterations, and absolute residual and error norms,* $\epsilon = 10^{-10}$

matrix	SGS	SSOR	ω	Steep. desc.
pde_225	37	70	1.65094	712
residual	$1.10993\ 10^{-9}$	$1.37247\ 10^{-9}$		$1.58492\ 10^{-9}$
error	$1.72582\ 10^{-9}$	$1.59195\ 10^{-9}$		$3.30690\ 10^{-10}$
jpwh_991	297	194	1.55842	1485
	$1.19337\ 10^{-9}$	$1.17963\ 10^{-9}$		$1.19124\ 10^{-9}$
	$9.78922\ 10^{-9}$	$4.84431\ 10^{-9}$		$7.05114\ 10^{-9}$
pde2961	468	113	1.63145	2938
	$1.53321\ 10^{-9}$	$1.35070\ 10^{-9}$		$1.55939\ 10^{-9}$
	$3.79086\ 10^{-8}$	$3.01899\ 10^{-8}$		$2.70660\ 10^{-8}$
fs_680_1c	168	77	1.48671	684
	$2.24252\ 10^{-9}$	$2.18622\ 10^{-9}$		$2.30158\ 10^{-9}$
	$8.62051\ 10^{-8}$	$6.53387\ 10^{-8}$		$5.37291\ 10^{-8}$
add20	20000	20000	1.92902	20000
	$1.04423\ 10^{-20}$	$4.11000\ 10^{-13}$		$3.73052\ 10^{-15}$
	$1.34952\ 10^{-16}$	$1.28427\ 10^{-9}$		$3.92475\ 10^{-11}$
fs_680_1	147	68	1.48671	20000
	$7.46767\ 10^{3}$	$7.46587\ 10^{3}$		$1.05230\ 10^{9}$
	$1.02724\ 10^{-6}$	$7.94496\ 10^{-7}$		$1.04683\ 10^{-1}$
sherman1	20000	20000,	1.93198	20000
	$6.81163\ 10^{-10}$	$3.08831\ 10^{-03}$		$3.38688\ 10^{-4}$
	$1.34822\ 10^{-6}$	3.70904		$8.94886\ 10^{-1}$
nos3	20000	20000	0.462655	20000
	$7.73232\ 10^{-4}$	$3.12171\ 10^{2}$		$1.49296\ 10^{-1}$
	$3.90825\ 10^{-2}$	$3.15268\ 10^{1}$		5.98348
watt1	4782	16561	1.86535	20000
	$4.23667\ 10^{-9}$	$4.23853\ 10^{-9}$		$4.40261\ 10^{1}$
	$1.02966\ 10^{1}$	3.66456		$2.58678\ 10^{9}$

Table 4.5. *Number of iterations, and relative residual and error norms,* $\epsilon = 10^{-10}$

matrix	SGS	SSOR	ω	Steep. desc.
pde_225	37	70	1.65094	712
	$6.86986\ 10^{-11}$	$8.49484\ 10^{-11}$		$9.80976\ 10^{-11}$
	$1.71659\ 10^{-10}$	$1.58343\ 10^{-10}$		$3.28921\ 10^{-11}$
jpwh_991	297	194	1.55842	1485
	$9.91042_{1}0^{-11}$	$9.79628\ 10^{-11}$		$9.89269\ 10^{-11}$
	$3.10965\ 10^{-10}$	$1.53885\ 10^{-10}$		$2.23987\ 10^{-10}$
pde2961	468	113	1.63145	2938
	$9.81395\ 10^{-11}$	$8.64574\ 10^{-11}$		$9.98150\ 10^{-11}$
	$6.96656_{1}0^{-10}$	$5.54808\ 10^{-10}$		$4.97399\ 10^{-10}$
fs_680_1c	168	77	1.48671	684
	$9.49813\ 10^{-11}$	$9.25965\ 10^{-11}$		$9.74825\ 10^{-11}$
	$3.30582\ 10^{-9}$	$2.50562\ 10^{-9}$		$2.06042\ 10^{-9}$
add20	20000	20000	1.92902	20000
	$1.05309\ 10^{-10}$	$4.14486\ 10^{-3}$		$3.76216\ 10^{-5}$
	$1.23553_{1}0^{-9}$	$1.17579\ 10^{-2}$		$3.59323\ 10^{-4}$
fs_680_1	147	68	1.48671	20000
	$9.28521\ 10^{-11}$	$9.28298\ 10^{-11}$		$1.30842\ 10^{-5}$
	$3.93928\ 10^{-8}$	$3.04675\ 10^{-8}$		$4.01443\ 10^{-3}$
sherman1	20000	20000	1.93198	20000
	$1.78225\ 10^{-10}$	$8.08051\ 10^{-4}$		$8.86172\ 10^{-5}$
	$1.09699_{1}0^{-8}$	$3.01789\ 10^{-2}$		$7.28133\ 10^{-3}$
nos3	20000	20000	0.462655	20000
	$2.42928\ 10^{-6}$	$9.80754\ 10^{-1}$		$4.69046\ 10^{-4}$
	$1.26138\ 10^{-3}$	1.01752		$1.93116\ 10^{-1}$
watt1	4782	16561	1.86535	20000
	$9.99321\ 10^{-11}$	$9.99760\ 10^{-11}$		1.03846
	$3.98370\ 10^{-9}$	$1.41780\ 10^{-9}$		1.00081

Let us finally illustrate the benefits of Chebyshev acceleration. We solve a symmetric linear system arising from the 5-point discretization of the Poisson equation in the unit square. The characteristics of the matrix `Lap2500` of order 2500 are given in the Appendix. Figure 4.7 shows the residual norms for Jacobi and its Chebyshev acceleration. Figure 4.8 shows the residual norms for SSOR with $\omega = 1.9$ and its acceleration. In both cases the convergence is improved. However, for these two splittings, we have to know the smallest and largest eigenvalues of $M^{-1}N$. For this example, they are known analytically for Jacobi, but not for SSOR.

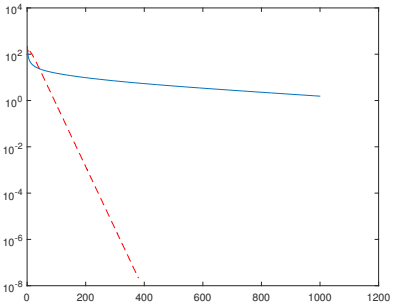


Figure 4.7. *Residual norms for Jacobi (plain) and for its Chebyshev acceleration (dashed)*

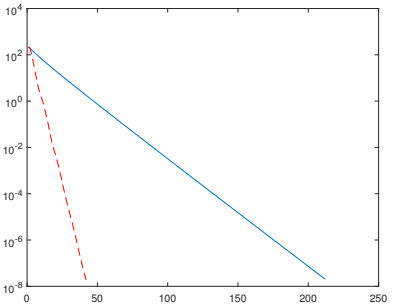


Figure 4.8. *Residual norms for SSOR with $\omega = 1.9$ (plain) and for its Chebyshev acceleration (dashed)*

4.9 ▪ Historical and bibliographical comments

It seems that iterative methods were not used for solving linear systems before Carl Friedrich Gauss (1777-1855), at the beginning of the 1820s. He explained his method in a letter to one of his former students, Christian Ludwig Gerling (1788-1864); see the quote at the beginning of this chapter. Gauss used his method to solve linear systems arising from least squares geodesy problems. Gerling published the method in a book in 1843.

In 1845, Carl Gustav Jacob Jacobi (1804-1851) also used an iterative method for the solution of the normal equations for least squares problems. He considered diagonally dominant symmetric systems, where the off-diagonal entries are small compared to the diagonal ones.

Philipp Ludwig Seidel (1821-1896) published his iterative method in 1874, even though he used it before in 1862. Seidel did not use matrices and did not give many details about the derivation of his method. The first method proposed by Seidel was to change the components $x_1, x_2, x_3, \ldots$ of the approximation cyclically, but he remarked that any ordering of the components can be used.

An analysis of the convergence of Seidel's method was done in 1885 by the Russian mathematician Pavel Alekseevich Nekrasov (1853-1924), who was working mainly on probability. Nekrasov was the first to relate the convergence to the eigenvalues of the iteration matrix and to analyze the modern Gauss-Seidel method. After 1890, in correspondence with Rudolf Mehmke (1857-1944), a German mathematician, he published sufficient convergence conditions. In 1892, Mehmke described what can be considered a block Seidel's method.

Sufficient conditions for convergence of Jacobi and Gauss-Seidel methods were given in 1929 by Richard Edler von Mises (1883-1953) and Hilda Pollaczek-Geiringer (1893-1973).

Relaxation methods started to be developed by Richard Vyne Southwell (1888-1970) in the UK in the second half of the 1930s, at first as a pure engineering method for problems in mechanics. The choices of the unknowns to be relaxed were based on the intelligence and the insight of the engineer doing the computation. Then, Southwell realized that he could apply the same techniques to other engineering problems. He summarized his research in a book published in 1940.

Edgar Reich (1927-2009) was a Research Assistant at MIT when, in 1949, he published a paper in which he proved that if A is a real symmetric matrix with a positive diagonal, Seidel's method is convergent if and only if A is positive definite. In 1949, Geiringer proved that the Gauss-Seidel method is convergent for irreducibly diagonally dominant matrices.

The idea of a symmetric Gauss-Seidel method was introduced by Alexander Craig Aitken (1895-1967) in 1950.

A landmark in the development of classical iterative methods is the Ph.D. thesis of David Monaghan Young (1923-2008) in 1950, done under the supervision of Garrett Birkhoff (1911-1996) at Harvard University. In his thesis, he developed the theory for the choice of the optimal SOR parameter. The SOR method was also proposed independently by Stanley Phillips Frankel (1919-1978) in 1950 under the name extrapolated Liebmann method. Even though the work of Young was more general, since the paper of Frankel was published in 1950 and that of Young only in 1954 in a journal, the SOR method was often named Frankel's method or the extrapolated Liebmann's method in the 1950s.

The symmetric successive overrelaxation (SSOR) method was introduced by John Waldo Sheldon (1923-2015) in 1955. A block version was proposed by Louis William Ehrlich (1927-2007) in 1964.

Regular splittings were introduced by Richard Steven Varga (1928-2022) in 1960. The research on classical iterative methods was summarized in the seminal books by R.S. Varga [1098] in 1962 and D.M. Young [1144] in 1971.

Alternating direction methods were introduced by Donald William Peaceman (1926-2017)

and Henry Herbert Rachford Jr. (1925-2022) in 1955, as well as by Jim Douglas Jr. (1927-2016).

In 1910, Lewis Fry Richardson (1881-1953), an English mathematician, physicist, and meteorologist, published a 53-page paper in which he considered approximate solutions of partial differential equations using finite difference schemes, and an iterative method that will be known later as (first-order) Richardson's method. The problem of convergence and the choice of the parameters in Richardson's method were studied in the second half of the 20th century.

Chebyshev polynomials are named after the famous Russian mathematician Pafnuty Lvovich Chebyshev (1821-1894). He was the founder of the Saint Petersburg school of mathematics. A first semi-iterative method was studied by R.S. Varga in 1957. G.H. Golub (1932-2007) and R.S. Varga wrote a two-part paper that was published in 1961. They considered three methods, Chebyshev, SOR, and second-order Richardson. The orderings for the parameters of the second-order Richardson method were studied by R.S. Anderssen and G.H. Golub in 1972.

For more details on the history of classical iterative methods, see Chapter 5 of [160].

5

The conjugate gradient and related methods

Let A be a symmetric positive definite (SPD) matrix of order n. One of the most efficient iterative method for solving $Ax = b$ is the conjugate gradient (CG) method, particularly in its preconditioned form (PCG). This method was introduced by Magnus R. Hestenes and Eduard Stiefel in 1952 [629]. CG can be derived in several different ways: as a minimization method, from the Lanczos algorithm, as an orthogonalization method, or as an acceleration method of a splitting iterative method. Let us explore some of these derivations.

5.1 ▪ CG as a minimization method

It is well known that the solution x_e of $Ax = b$ gives also the minimum of the functional

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b.$$

The minimum value of ϕ is $-b^T A^{-1} b/2$ for $x = x_e = A^{-1}b$. The negative gradient of ϕ is $-\nabla\phi(x) = b - Ax$, which is the residual vector r. It is interesting to note that the functional ϕ is related to the A-norm of the error $e = x_e - x$ which satisfies $Ae = r$.

Lemma 5.1. *Let $e = x_e - x$. Then,*

$$\|e\|_A^2 = e^T Ae = x_e^T Ax_e + 2\phi(x). \tag{5.1}$$

Proof. We have

$$\begin{aligned} e^T Ae &= x_e^T Ax_e - 2x^T Ax_e + x^T Ax, \\ &= x_e^T Ax_e - 2x^T b + x^T Ax, \\ &= x_e^T Ax_e + 2\phi(x). \end{aligned}$$

□

If we can derive an algorithm to decrease ϕ at each iteration, this will also decrease the A-norm of the error.

Let us look for something which is similar to what is done in the steepest descent algorithm,

$$x_{k+1} = x_k + \gamma_k p_k,$$

where $p_0, p_1, \ldots$ are descent directions to be chosen. The value $\phi(x_{k+1})$ is minimized by choosing

$$\gamma_k = \frac{p_k^T r_k}{p_k^T A p_k}, \tag{5.2}$$

and

$$\phi(x_k + \gamma_k p_k) = \phi(x_k) - \frac{1}{2}\frac{(p_k^T r_k)^2}{p_k^T A p_k}.$$

To decrease ϕ we need to have $p_k^T r_k \neq 0$ and its ratio with $p_k^T A p_k$ as large as possible. In the steepest descent method, the direction p_k is chosen as $r_k = b - Ax_k$. It is a local minimization algorithm since it minimizes ϕ only in the direction of r_k. In CG, we choose the descent directions in a different way. We set $p_0 = r_0$, and we would like to solve

$$\min_{x \in x_0 + \text{span}\{p_0, \ldots, p_k\}} \phi(x),$$

where $p_0, \ldots, p_k$ are linearly independent. If we solve this problem, we have convergence (that is, a zero residual vector) in at most n steps. Let $P_k = (\, p_0 \quad \cdots \quad p_k \,)$. If $x_{k+1} = x_0 + P_{k-1}y + \gamma p_k$,

$$\phi(x_{k+1}) = \phi(x_0 + P_{k-1}y) + \gamma y^T P_{k-1}^T A p_k + \frac{\gamma^2}{2} p_k^T A p_k - \gamma p_k^T r_0.$$

Let us assume that p_k is orthogonal to $\text{span}\{Ap_0, \ldots, Ap_{k-1}\}$, that is, $p_k^T A P_{k-1} = 0$. Then,

$$\min_{x \in x_0 + \text{span}\{p_0, \ldots, p_k\}} \phi(x) = \min_{y,\gamma} \left(\phi(x_0 + P_{k-1}y) + \frac{\gamma^2}{2} p_k^T A p_k - \gamma p_k^T r_0 \right).$$

This is equal to

$$\min_y \phi(x_0 + P_{k-1}y) + \min_\gamma \left(\frac{\gamma^2}{2} p_k^T A p_k - \gamma p_k^T r_0 \right). \tag{5.3}$$

Let y_{k-1} be the solution of the first minimization problem. Then, $x_0 + P_{k-1}y_{k-1} = x_k$. The solution of the second one is

$$\gamma = \frac{p_k^T r_0}{p_k^T A p_k},$$

but

$$\begin{aligned} p_k^T r_k &= p_k^T (b - Ax_k), \\ &= p_k^T (b - A(x_0 + P_{k-1}y_{k-1})) = p_k^T r_0, \end{aligned}$$

since $p_k^T A P_{k-1} = 0$. Therefore, the solution of the second problem is $\gamma = \gamma_k$ defined by (5.2). The splitting of the minimization problem in two parts shows that we can write

$$x_{k+1} = x_k + \gamma_k p_k, \quad r_{k+1} = r_k - \gamma_k A p_k.$$

Let us show by induction that $P_k^T r_{k+1} = 0$. Since $p_0 = r_0$ and $r_1 = r_0 - \gamma_1 A r_0$, we have $p_0^T r_1 = 0$. Assume that $P_{k-1}^T r_k = 0$. Since $r_{k+1} = r_k - \gamma_k A p_k$, and because of the A-orthogonality of the p_j's, we have

$$\begin{aligned} P_k^T r_{k+1} &= P_k^T r_k - \gamma_k P_k^T A p_k, \\ &= \begin{pmatrix} P_{k-1}^T r_k \\ p_k^T r_k \end{pmatrix} - \frac{p_k^T r_k}{p_k^T A p_k} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_k^T A p_k \end{pmatrix}, \end{aligned}$$

$$= \begin{pmatrix} P_{k-1}^T r_k \\ 0 \end{pmatrix},$$
$$= 0.$$

We choose p_{k+1} as $p_{k+1} = r_{k+1} - AP_k z_k$, where z_k is the solution of the least squares problem

$$\min_z \|r_{k+1} - AP_k z_k\|.$$

We observe that, writing the normal equations, we have

$$P_k^T A p_{k+1} = P_k^T A(r_{k+1} - AP_k z_k) = 0.$$

So, the A-orthogonality (also called conjugacy) condition is satisfied. Moreover, p_{k+1} is the orthogonal projection of r_{k+1} onto the range of $(AP_k)^\perp$, and the closest vector to r_{k+1} in that space. From the definition of the residuals, we have

$$\{Ap_0, \ldots, Ap_k\} \subseteq \{r_0, \ldots, r_{k+1}\},$$

and, by the definition of p_{k+1}, we have

$$\text{span}\{p_0, \ldots, p_{k+1}\} = \text{span}\{r_0, \ldots, r_{k+1}\}.$$

Since $P_k^T r_{k+1} = 0$, r_{k+1} is orthogonal to any vector in the range of P_k which contains $r_0, \ldots, r_k$. Therefore, r_{k+1} is orthogonal to all the previous residual vectors.

Let us show that p_{k+1} defined as above is a linear combination of r_{k+1} and p_k. We have to relate p_{k+1} to p_k by writing

$$p_{k+1} = r_{k+1} - AP_k \begin{pmatrix} w \\ \omega \end{pmatrix} = r_{k+1} - AP_{k-1} w - \omega A p_k.$$

Using $Ap_k = -(r_{k+1} - r_k)/\gamma_k$, it yields

$$p_{k+1} = \left(1 + \frac{\omega}{\gamma_k}\right) r_{k+1} - s_k, \quad s_k = -\left(\frac{\omega}{\gamma_k} r_k + AP_k w\right).$$

Clearly, s_k is in $\text{span}\{r_k, Ap_0, \ldots, Ap_k\}$ which is contained in $\text{span}\{r_0, \ldots, r_k\}$. Hence, s_k is orthogonal to r_{k+1}. The minimization problem giving p_{k+1} is equivalent to finding ω and w minimizing

$$\left(1 + \frac{\omega}{\gamma_k}\right)^2 \|r_{k+1}\|^2 + \|s_k\|^2.$$

From the definition of s_k, we see that the minimization of its norm implies that s_k is a multiple of p_k. Therefore, we can write

$$p_{k+1} = r_{k+1} + \delta_{k+1} p_k.$$

To satisfy the orthogonality condition $p_k^T A p_{k+1} = 0$, we must have

$$\delta_{k+1} = -\frac{p_k^T A r_{k+1}}{p_k^T A p_k}.$$

This expression for δ_{k+1} can be simplified since, because of the orthogonality of the residuals, $r_{k+1}^T r_{k+1} = -\gamma_k r_{k+1}^T A p_k$. Moreover, $p_k^T A p_k = p_k^T r_k/\gamma_k = r_k^T r_k/\gamma_k$. Finally, we have the simpler expression,

$$\delta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k},$$

and γ_k can be written as

$$\gamma_k = \frac{r_k^T r_k}{p_k^T A p_k}.$$

We observe that the CG iterates are in $x_0 + \text{span}\{p_0, \ldots, p_k\}$. This is the same as $x_0 + \mathcal{K}_{k+1}(A, r_0)$ where

$$\mathcal{K}_{k+1}(A, v) = \text{span}\{v, Av, A^2 v, \ldots, A^k v\},$$

which is a so-called *Krylov subspace*.

Coding CG is particularly simple as one can see in the following.

```
function [x,nit,res] = cg(A,b,x0,nitmax,epss)
res = zeros(1,nitmax+1);
nb = norm(b);
x = x0;
r = b - A * x;
p = r;
rtr = r' * r;
resid = sqrt(rtr);
res(1) = resid;
nit = 0;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 Ap = A * p;
 gamma = rtr / (p' * Ap);
 x = x + gamma * p;
 r = r - gamma * Ap;
 rk = r' * r;
 resid = sqrt(rk);
 res(nit+1) = resid;
 delta = rk / rtr;
 rtr = rk;
 p = r + delta * p;
end % while
```

There is one matrix-vector product, two dot products, and three axpy operations per iteration. The number of operations per iteration is approximately $10n$ plus what is needed for the matrix-vector product.

Let us summarize the properties of the CG algorithm obtained from the previous discussion.

Theorem 5.2. *The vectors generated by the CG algorithm are such that*

1) $p_i^T A p_j = 0$, $i \neq j$,

2) $r_i^T r_j = 0$, $i \neq j$,

3) $p_i^T r_j = 0$, $i < j$.

Moreover, CG minimizes the A-norm of the error at each iteration.

Iterating the recurrence relation of p_k, we have

$$p_k = r_k + \frac{\|r_k\|^2}{\|r_{k-1}\|^2} r_{k-1} + \cdots + \frac{\|r_k\|^2}{\|r_0\|^2} r_0. \tag{5.4}$$

and, using the orthogonality of the residual vectors,

$$\frac{\|p_k\|^2}{\|r_k\|^2} = 1 + \sum_{j=0}^{k-1} \frac{\|r_k\|^2}{\|r_j\|^2}\,. \tag{5.5}$$

However, the relation (5.5) can also be derived assuming local orthogonality $r_j^T p_{j-1} = 0$ only. When CG converges fast enough, the descent direction p_k and the residual r_k tend to be in the same direction. One can also prove [816, Lemma 2.31] that $p_k^T A p_k \leq r_k^T A r_k$. From this, and assuming only local orthogonality of the residuals, it can be shown that CG converges at least as fast as the steepest descent algorithm which, unfortunately, is known to be quite slow.

CG is almost always used in its precondtioned form to speed up convergence. Let M be an SPD matrix which is known as the *precondtioner*, see Chapter 7. The preconditioned CG (PCG) is obtained by applying CG to the preconditioned linear system

$$\widehat{A}\,\widehat{x} = \widehat{b}, \qquad \widehat{A} = M^{-1/2} A M^{-1/2}, \quad \widehat{b} = M^{-1/2} b,$$

and then, going back to the original variables x, r, p. Note that this is done to preserve symmetry. Denoting the corresponding CG coefficients and vectors with a hat and defining

$$x_k \equiv M^{-1/2}\widehat{x}_k, \;\; r_k \equiv M^{1/2}\,\widehat{r}_k, \;\; p_k \equiv M^{-1/2}\widehat{p}_k, \;\; z_k \equiv M^{-1} r_k,$$

where x_k and r_k represent the approximate solution and residual for the original problem $Ax = b$, we obtain the standard version of the preconditioned CG (PCG) method which involves only M. It is straightforward to see that

$$\|\widehat{x} - \widehat{x}_k\|_{M^{-1/2} A M^{-1/2}} = \|x - x_k\|_A. \tag{5.6}$$

Initially, $r_0 = b - Ax_0$, $Mz_0 = r_0$, and $p_0 = z_0$. One iteration of PCG is the following,

$$\begin{aligned}
\gamma_k &= \frac{r_k^T z_k}{p_k^T A p_k}, \\
x_{k+1} &= x_k + \gamma_k p_k, \\
r_{k+1} &= r_k - \gamma_k A p_k, \\
M z_{k+1} &= r_{k+1}, \\
\delta_{k+1} &= \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}, \\
p_{k+1} &= z_{k+1} + \delta_{k+1} p_k.
\end{aligned}$$

The residual vectors in PCG satisfy the orthogonality relation

$$(r_i, z_j) = r_i^T z_j = r_i^T M^{-1} r_j = 0, \quad i \neq j.$$

The matrix M is usually chosen (heuristically) to improve the convergence of the algorithm. We observe that we have to solve a linear system with the matrix M at each iteration. Hence, M must also be chosen such that this is not too expensive.

In CG and PCG, the computation of x_{k+1} and r_{k+1} are decoupled. In finite precision arithmetic, this may lead to r_k being different from $b - Ax_k$. That difference was called the *residual gap* in [563].

5.2 ▪ CG from the Lanczos algorithm

Let A be a real nonsingular symmetric matrix of order n, not necessarily positive definite. Given a starting vector v, the Lanczos algorithm computes an orthonormal basis $v_1, \ldots, v_{k+1}$ of the Krylov subspace $\mathcal{K}_{k+1}(A, v)$ defined above. The dimension of these subspaces for $k = 0, 1, 2, \ldots$ increases up to an index $m \leq n$, called the *grade of* v *with respect to* A, at which the maximal dimension is attained, and $\mathcal{K}_m(A, v)$ is invariant under multiplication with A. The orthonormality condition for the basis vectors means that they are of unit norm and that $v_i^T v_j = 0$, $i \neq j$. We use a particular form of the Gram-Schmidt algorithm. Because the matrix A is symmetric, it is enough to orthogonalize against the two previous vectors v_k and v_{k-1} to construct the next basis vector v_{k+1}. A code implementing the Lanczos algorithm is the following.

```
function [V,T] = Lanczos(A,u,nitmax)
v = u / norm(u);
T = sparse(nitmax+1,nitmax+1);
V = zeros(n,nitmax+2);
V(:,1) = v;
v1 = v;
u = A * v;
alpha = v' * u;
T(1,1) = alpha;
r = u - alpha * v;
beta = norm(r);
v = r / beta;
V(:,2) = v;

for k = 2:nitmax+1
 T(k,k-1) = beta;
 T(k-1,k) = beta;
 u = A * v - beta * v1;
 alpha = v' * u;
 T(k,k) = alpha;
 r = u - alpha * v;
 beta = norm(r);
 v1 = v;
 v = r / beta;
 V(:,k+1) = v;
end % for k
```

In this code we have used the so-called modified Gram-Schmidt (MGS) form of the algorithm for the orthogonalization. This algorithm generates basis vectors which are the columns of V and a tridiagonal matrix T. The basis vectors v_j satisfy the relation

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T \tag{5.7}$$

where e_k is the last column of the identity matrix of order k, $V_k = [v_1, \cdots, v_k]$ and T_k is the $k \times k$ symmetric tridiagonal matrix constructed with the recurrence coefficients computed in the algorithm,

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \end{pmatrix}.$$

The coefficients β_j being positive, T_k is often called a *Jacobi matrix*. When A is positive definite, the matrix T_k is positive definite as well.

The Lanczos algorithm is usually used to compute approximations of the eigenvalues of A. The approximations at iteration k (the so-called *Ritz values*) are the eigenvalues of the matrix T_k. The basis vectors can also be used to solve linear systems. Given an initial vector x_0, the approximate solution of a linear system $Ax = b$ is sought as

$$x_k = x_0 + V_k y_k, \quad k = 1, 2, \ldots, \tag{5.8}$$

by choosing $u = r_0$ and asking the residual vector

$$r_k = b - Ax_k = b - A(x_0 + V_k y_k) = r_0 - AV_k y_k$$

to be orthogonal to V_k. The orthogonality property of the basis vectors leads to

$$T_k\, y_k = \|r_0\| e_1\,, \tag{5.9}$$

that is, the coordinate vector y_k is the solution of a linear system with the tridiagonal matrix T_k. Mathematically, this gives $r_{m+1} = 0$.

When A is positive definite, CG is obtained by using a Cholesky-like factorization of the tridiagonal matrix

$$T_k = L_k D_k L_k^T \tag{5.10}$$

with

$$L_k \equiv \begin{pmatrix} 1 & & & \\ \sqrt{\delta_1} & \ddots & & \\ & \ddots & \ddots & \\ & & \sqrt{\delta_{k-1}} & 1 \end{pmatrix}, \qquad D_k \equiv \begin{pmatrix} \gamma_0^{-1} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \gamma_{k-1}^{-1} \end{pmatrix}, \tag{5.11}$$

where the δ_j's and γ_j's are the coefficients computed in the standard form of CG derived in the previous section. A detailed derivation is given in [816].

The relation of CG with the Lanczos algorithm shows that the recurrence coefficients of both methods are linked through

$$\beta_{k+1} = \frac{\sqrt{\delta_k}}{\gamma_{k-1}}, \quad \alpha_k = \frac{1}{\gamma_{k-1}} + \frac{\delta_{k-1}}{\gamma_{k-2}}, \quad \delta_0 = 0, \quad \gamma_{-1} = 1. \tag{5.12}$$

Moreover, the CG residual vectors r_k are proportional to the Lanczos basis vectors v_{k+1} and hence mutually orthogonal,

$$v_{j+1} = (-1)^j \frac{r_j}{\|r_j\|}\,, \qquad j = 0, \ldots, k.$$

As we already know, the residual vectors r_j yield an orthogonal basis of the Krylov subspaces $\mathcal{K}_{k+1}(A, r_0)$.

5.3 ▪ CG as an orthogonalization and acceleration algorithm

In Chapter 4 we study how to accelerate a basic linear iterative method. Let us start from a slight generalization of Richardson's method

$$Mx_{k+1} = Mx_k + \alpha(b - Ax_k),$$

where M is a nonsingular preconditioning matrix. The standard Richardson's method corresponds to $M = I$. A formal generalization of the acceleration method is to allow the parameters to vary with the iteration number,

$$\begin{aligned} x_{k+1} &= \omega_{k+1}(\alpha_k z_k + x_k - x_{k-1}) + x_{k-1}, \\ Mz_k &= r_k. \end{aligned} \tag{5.13}$$

Let A and M be SPD. We would like to choose ω_{k+1} and α_k to obtain a converging sequence. Let us first compute ω_{k+1}, α_k, $k = 0, 1, \ldots$ such that the generalized residuals z_k are mutually orthogonal for the dot product defined by the SPD matrix M.

Lemma 5.3. *Let M be symmetric positive definite and z_k, $k = 0, 1, \ldots, n$ be a sequence in $\mathbb{R}^n$ such that*

$$z_i^T M z_j = 0, \quad i \neq j. \tag{5.14}$$

Then, $z_n = 0$.

Proof. The proof is straightforward. Since the vectors z_k, $k = 0, 1, \ldots, n-1$ are orthogonal in the dot product defined by M, they give a basis of the whole space. Therefore,

$$z_n = \sum_{j=0}^{n-1} \eta_j z_j, \quad z_n^T M z_n = \sum_{j=0}^{n-1} \eta_j z_j^T M z_n = 0.$$

Since M is SPD, this implies that $z_n = 0$. □

Let us now show by induction that we can construct ω_k and α_k to satisfy (5.14). Assume

$$z_i^T M z_j = 0, \quad i \neq j, \quad 0 \leq i, j \leq k.$$

From (5.13) it is easy to see that

$$r_{k+1} = r_{k-1} - \omega_{k+1}(\alpha_k A z_k - r_k + r_{k-1}). \tag{5.15}$$

This can be written as

$$Mz_{k+1} = Mz_{k-1} - \omega_{k+1}(\alpha_k A z_k - Mz_k + Mz_{k-1}). \tag{5.16}$$

Lemma 5.4. *If α_k is chosen as*

$$\alpha_k = \frac{z_k^T M z_k}{z_k^T A z_k},$$

then, $z_k^T M z_{k+1} = 0$.

Proof. Multiplying relation (5.16) by z_k, we have,

$$z_k^T M z_{k+1} = z_k^T M z_{k-1} - \omega_{k+1}[\alpha_k \, z_k^T A z_k - z_k^T M z_k + z_k^T M z_{k-1}].$$

But, by the induction hypothesis $z_k^T M z_{k-1} = 0$, and

$$z_k^T M z_{k+1} = -\omega_{k+1}[\alpha_k \, z_k^T A z_k - z_k^T M z_k],$$

which gives the result if $\omega_{k+1} \neq 0$. □

Lemma 5.5. *If ω_{k+1} is chosen as*

$$\omega_{k+1} = \frac{1}{1 + \alpha_k \frac{z_{k-1}^T A z_k}{z_{k-1}^T M z_{k-1}}}, \tag{5.17}$$

then, $z_{k-1}^T M z_{k+1} = 0$.

Proof. We multiply relation (5.16) by z_{k-1}. Then

$$z_{k-1}^T M z_{k+1} = z_{k-1}^T M z_{k-1} - \omega_{k+1} \left[\alpha_k \, z_{k-1}^T A z_k - z_{k-1}^T M z_k + z_{k-1}^T M z_{k-1} \right].$$

Therefore,

$$\omega_{k+1} = \frac{z_{k-1}^T M z_{k-1}}{\alpha_k z_{k-1}^T A z_k + z_{k-1}^T M z_{k-1}}.$$

□

Note that if $z_k \neq 0$, then $z_k^T A z_k \neq 0$, so α_k can always be computed. For ω_{k+1} defined by (5.17), this is not so obvious yet. So far we have not used the hypothesis that A and M are symmetric to compute the coefficients. Let us now give a cheaper computational expression for ω_{k+1}. We split A as $A = M - N$, and

$$\begin{aligned} z_{k-1}^T A z_k &= z_{k-1}^T (M - N) z_k, \\ &= -z_{k-1}^T N z_k. \end{aligned}$$

By writing relation (5.16) at iteration k,

$$M z_k = M z_{k-2} - \omega_k (\alpha_{k-1}(M - N) z_{k-1} - M z_{k-1} + M z_{k-2}).$$

Multiplying by z_k and using the induction hypothesis, we obtain

$$z_k^T M z_k = \omega_k \alpha_{k-1} \, z_k^T N z_{k-1}.$$

But, N is a symmetric matrix so,

$$z_{k-1}^T N z_k = \frac{z_k^T M z_k}{\omega_k \alpha_{k-1}}.$$

Therefore,

$$\omega_{k+1} = \frac{1}{1 - \frac{\alpha_k \, z_k^T M z_k}{\omega_k \alpha_{k-1} \, z_{k-1}^T M z_{k-1}}}. \tag{5.18}$$

Formula (5.18) is more efficient than (5.17) since we do not have to compute the additional dot product $z_{k-1}^T A z_k$. Since the method (5.13) involves two levels of iterations, we need x_0 and x_{-1} to start with. This is overcome by taking $\omega_1 = 1$, then

$$x_1 = \alpha_0 z_0 + x_0,$$

$$M z_0 = r_0,$$

and we need only to define x_0. The first step is only a steepest descent iteration. We must now show that the induction hypothesis about orthogonality holds at level $k+1$, that is, the new vector is orthogonal not only to the last two, but to all the previous ones.

Theorem 5.6. *The induction hypothesis holds at iteration $k+1$,*

$$z_{k+1}^T M z_j = 0, \quad 0 \le j < k-1.$$

Proof. Multiplying relation (5.16) by z_j, $0 \le j < k-1$, we have,

$$z_j^T M z_{k+1} = z_j^T M z_{k-1} - \omega_{k+1}[\alpha_k\, z_j^T (M-N) z_k - z_j^T M z_k + z_j^T M z_{k-1}].$$

But, since $j < k-1$,

$$z_j^T M z_{k+1} = \omega_{k+1}\, \alpha_k\, z_j^T N z_k.$$

Writing relation (5.16) at iteration $j+1$, we obtain

$$M z_{j+1} = M z_{j-1} - \omega_{j+1}\,(\alpha_j (M-N) z_j - M z_j + M z_{j-1}).$$

Multiplying by z_k and observing that $j+1 < k$,

$$\omega_{j+1}\, \alpha_j z_k^T N z_j = 0.$$

This is where we need the hypothesis that N is symmetric to imply $(N z_k)^T z_j = 0$. It shows that $z_j^T M z_{k+1} = 0$ for all j such that $j < k-1$. □

This method is the three-term recurrence variant of the PCG method. The computational steps in PCG are,

$$\begin{aligned}
M z_k &= r_k \ (= b - A x_k), \\
\alpha_k &= \frac{z_k^T M z_k}{z_k^T A z_k}, \\
\omega_{k+1} &= \frac{1}{1 - \frac{\alpha_k}{\omega_k \alpha_{k-1}} \frac{z_k^T M z_k}{z_{k-1}^T M z_{k-1}}}, \quad \omega_1 = 1, \\
x_{k+1} &= x_{k-1} + \omega_{k+1}\,(\alpha_k z_k + x_k - x_{k-1}).
\end{aligned} \tag{5.19}$$

The residual r_{k+1} is usually computed by relation (5.15) and not as $b - A x_k$ to save a matrix-vector product.

The two variants of PCG are mathematically equivalent. This is obvious for $k = 0$. For $k > 0$ we can eliminate p_k as

$$p_k = \frac{1}{\gamma_k}(x_{k+1} - x_k).$$

Therefore,

$$\frac{1}{\gamma_k}(x_{k+1} - x_k) = z_k + \frac{\delta_k}{\gamma_{k-1}}(x_k - x_{k-1}).$$

A short manipulation gives

$$x_{k+1} = x_{k-1} + \gamma_k z_k + \left(1 + \frac{\gamma_k \delta_k}{\gamma_{k-1}}\right)(x_k - x_{k-1}).$$

We can identify the two expressions for x_{k+1},

$$\begin{aligned}
\omega_{k+1} &= 1 + \frac{\gamma_k \delta_k}{\gamma_{k-1}}, \\
\alpha_k &= \frac{\gamma_k}{1 + \frac{\gamma_k \delta_k}{\gamma_{k-1}}}.
\end{aligned}$$

Since $\gamma_k > 0$ and $\delta_k > 0$, this expression for ω_{k+1} shows that $\omega_{k+1} > 1$ and it is always well defined.

5.4 ▪ The convergence of CG

So far, we have just seen that the two variants of (P)CG must give $r_j = 0$ for some $j \leq n$. However, the A-norm of the error is decreasing and we would like to use CG as an iterative method. It is straightforward to see that in both variants of PCG we have

$$x_{k+1} = x_0 + P_k(K)z_0, \qquad K = M^{-1}A, \tag{5.20}$$

where P_k is a polynomial of degree k satisfying a three-term recurrence relation. Moreover,

$$z_{k+1} = [I - KP_k(K)]z_0.$$

The matrix K is nonsymmetric, but it is similar to a symmetric matrix $\bar{K} = M^{\frac{1}{2}}KM^{-\frac{1}{2}}$, and its eigenvalues λ_i are real and positive. The matrix $\bar{K}$ has a spectral factorization $\bar{K} = U\Lambda U^T$ with $U^TU = I$ and Λ diagonal.

Let the error be $\epsilon_k = x_e - x_k$. Since $K\epsilon_k = z_k$, we have

$$\epsilon_k = [I - KP_k(K)]\epsilon_0.$$

We denote the square of the A-norm of the error as $\varepsilon_k = (x_e - x_k)^T A(x_e - x_k)$.

Theorem 5.7.

$$\varepsilon_k = \sum_{i=1}^{n}(1 - \lambda_i P_k(\lambda_i))[\bar{\epsilon}_0]_i^2,$$

where $\bar{\epsilon}_j = \Lambda^{\frac{1}{2}}U^T M^{\frac{1}{2}}\epsilon_j$.

Proof. Clearly,

$$P_k(K) = M^{-\frac{1}{2}}P_k(\bar{K})M^{\frac{1}{2}}.$$

Therefore,

$$\epsilon_{k+1} = M^{-\frac{1}{2}}[I - \bar{K}P_k(\bar{K})]M^{\frac{1}{2}}\epsilon_0.$$

Since $P_k(\bar{K}) = UP_k(\Lambda)U^T$,

$$\epsilon_{k+1} = M^{-\frac{1}{2}}U\Lambda^{-\frac{1}{2}}[I - \Lambda P_k(\Lambda)]\Lambda^{\frac{1}{2}}U^T M^{\frac{1}{2}}\epsilon_0.$$

With our notation, this is simply

$$\bar{\epsilon}_{k+1} = [I - \Lambda P_k(\Lambda)]\bar{\epsilon}_0.$$

Let us compute $\bar{\epsilon}_{k+1}^T\bar{\epsilon}^{k+1}$. By definition

$$\begin{aligned}\bar{\epsilon}_{k+1}^T\bar{\epsilon}_{k+1} &= [\Lambda^{\frac{1}{2}}U^T M^{\frac{1}{2}}\epsilon_{k+1}]^T\Lambda^{\frac{1}{2}}U^T M^{\frac{1}{2}}\epsilon_{k+1},\\ &= [M^{\frac{1}{2}}\bar{K}M^{\frac{1}{2}}\epsilon_{k+1}]^T\epsilon_{k+1},\\ &= \epsilon_{k+1}^T A\epsilon_{k+1}.\end{aligned}$$

Therefore,

$$\varepsilon_{k+1} = [(I - \Lambda P_k(\Lambda))\bar{\epsilon}_0]^T(I - \Lambda P_k(\Lambda))\bar{\epsilon}_0).$$

□

From Theorem 5.7 we learn that the A-norm of the error depends on the distribution of the eigenvalues of $K = M^{-1}A$. We can be more precise by using the spectral factorization of

$\bar{K} = U\Lambda U^T$, where Λ is the diagonal matrix of the eigenvalues λ_i of $\bar{K}$ (which are the same as those of K). Let

$$\omega_i = \frac{(r_0, u_i)^2}{\|r_0\|^2}, \qquad i = 1, \ldots, n. \tag{5.21}$$

For any set $\mathcal{I}_k$ of k ordered indices $1 \leq i_1 < \cdots < i_k \leq n$, we define the quantities

$$\Theta_\Lambda(\mathcal{I}_1) \equiv 1, \qquad \Theta_\Lambda(\mathcal{I}_k) \equiv \prod_{\substack{i_\ell < i_j \\ i_\ell, i_j \in \mathcal{I}_k}} |\lambda_{i_j} - \lambda_{i_\ell}|^2, \quad \text{for} \quad k > 1. \tag{5.22}$$

The following theorem shows that ε_k can be expressed explicitly in terms of the matrix eigenvalues and scaled and squared projections ω_i.

Theorem 5.8.

$$\varepsilon_k = \|r_0\|^2 \frac{\sum_{\mathcal{I}_{k+1}} \left[\prod_{j=1}^{k+1} \frac{\omega_{i_j}}{\lambda_{i_j}}\right] \Theta_\Lambda(\mathcal{I}_{k+1})}{\sum_{\mathcal{I}_k} \left[\prod_{j=1}^{k} \omega_{i_j}\lambda_{i_j}\right] \Theta_\Lambda(\mathcal{I}_k)}, \quad k = 1, \ldots, n-1, \tag{5.23}$$

where $\mathcal{I}_k$ is a set of k ordered indices $1 \leq i_1 < \cdots < i_k \leq n$, ω_i is defined by (5.21), and $\Theta_\Lambda(\mathcal{I}_k)$ is defined by (5.22).

Proof. The proof uses the relationship of the A-norm of the CG error and MINRES residual norm that we will study in a following section. We see that ε_k also depends on the mutual distances of the eigenvalues. □

From Theorem 5.7 we can also obtain bounds for the A-norm of the error.

Theorem 5.9.

$$\varepsilon_{k+1} \leq \max_{1 \leq i \leq n} (R_{k+1}(\lambda_i))^2 \varepsilon_0, \tag{5.24}$$

for all polynomials R_{k+1} of degree $k+1$ such that $R_{k+1}(0) = 1$

Proof. The PCG polynomial P_k minimizes ε_{k+1}. Therefore, replacing the polynomial P_k by any other kth degree polynomial, we obtain a larger value. This can be written as

$$\varepsilon_{k+1} \leq \sum_{i=1}^{n} (R_{k+1}(\lambda_i))^2 [\bar{\epsilon}_0]_i^2,$$

for all polynomials R_{k+1} of degree $k+1$, such that $R_{k+1}(0) = 1$, equality holding only if $R_{k+1}(\lambda) = 1 - \lambda P_k(\lambda)$. Therefore,

$$\varepsilon_{k+1} \leq \max_{1 \leq i \leq n} (R_{k+1}(\lambda_i))^2 \sum_{i=1}^{n} [\bar{\epsilon}_0]_i^2.$$

But, we have $\|\bar{\epsilon}_0\|^2 = \|\epsilon_0\|_A^2$, which proves the result. □

Several mathematical results can be obtained by choosing different polynomials in inequality (5.24).

Theorem 5.10.

1) $\varepsilon_n = 0$,

2) If K *has only* p *distinct eigenvalues,* $\varepsilon_p = 0$,

3) $\varepsilon_k \leq 4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2k}\varepsilon_0$, *where* κ *is the condition number of* $K = M^{-1}A$.

Proof. The first item is the finite termination property that we have already seen. In fact, it happens for some $m \leq n$. To prove it again, we choose

$$R_k(\lambda) = \prod_{i=1}^{k}\left(1 - \frac{\lambda}{\lambda_i}\right).$$

Hence, $R_n(\lambda_i) = 0,\ \forall i,\ 1 \leq i \leq n,$ and $\varepsilon_n = 0$. To prove assertion 2), we simply take into account the distinct eigenvalues in R_k and the result follows.

To prove assertion 3), we first observe that $\max_{1\leq i\leq n}(R_k(\lambda_i))^2$ is bounded by

$$\max_{\lambda_{\min}\leq\lambda\leq\lambda_{\max}}(R_k(\lambda))^2.$$

For R_k we choose the kth degree polynomial such that $R_k(0) = 1$, which minimizes that bound. The solution of this problem was given in Chapter 1,

$$R_k(\lambda) = \frac{C_k(\frac{\lambda_{\min}+\lambda_{\max}-2\lambda}{\lambda_{\max}-\lambda_{\min}})}{C_k(\frac{\lambda_{\min}+\lambda_{\max}}{\lambda_{\max}-\lambda_{\min}})}.$$

where C_k are the Chebyshev polynomials. Then,

$$\max_{\lambda_{\min}\leq\lambda\leq\lambda_{\max}}|R_k(\lambda)| \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k.$$

This proves the result. □

Assertion 3) is the most well known bound on the A-norm of the error. This upper bound is generally a large overestimate of the A-norm of the error and its only interest is to show that, mathematically, the A-norm is bounded above by a sequence decreasing to zero when k goes to infinity. Note that if $\sqrt{\kappa(K)}$ is large, the ratio within parenthesis is close to 1, and the bound is useless.

Particular distributions of eigenvalues have been studied to improve the upper bound in 3). The following result was proved by O. Axelsson [57, 69].

Proposition 5.11. *Assume that*

$$\alpha \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-m} \leq \beta \leq \lambda_{n-m+1} \leq \cdots \leq \lambda_n,$$

and let $k \geq m$. *Then,*

$$\varepsilon_k \leq 4\left(\frac{\sqrt{\frac{\beta}{\alpha}}-1}{\sqrt{\frac{\beta}{\alpha}}+1}\right)^{2(k-m)}\varepsilon_0.$$

Proof. In (5.24), we choose

$$R_k(\lambda) = \prod_{i=1}^{m}\left(1 - \frac{\lambda}{\lambda_{n-i+1}}\right)\frac{C_{k-m}\left(\frac{\alpha+\beta-2\lambda}{\beta-\alpha}\right))}{C_{k-m}\left(\frac{\alpha+\beta}{\beta-\alpha}\right)}.$$

Because of the first term on the right, $R_k(\lambda_i) = 0$ for all λ_i, $i = n, n-1, \ldots, n-m+1$. Hence,

$$\max_i (R_k(\lambda_i))^2 = \max_{i=1,\ldots,n-m} (R_k(\lambda_i))^2 \leq \max_{\alpha \leq \lambda \leq \beta} (R_k(\lambda))^2.$$

When $\lambda \in [\alpha, \beta]$,

$$\left| 1 - \frac{\lambda}{\lambda_{n-i+1}} \right| < 1, \quad \forall i = 1, \ldots m.$$

Therefore,

$$\max_{\alpha \leq \lambda \leq \beta} (R_k(\lambda))^2 \leq \max_{\alpha \leq \lambda \leq \beta} \left| \frac{C_{k-m}\left(\frac{\alpha+\beta-2\lambda}{\beta-\alpha}\right)}{C_{k-m}\left(\frac{\alpha+\beta}{\beta-\alpha}\right)} \right|,$$

and the result follows. □

This result is of interest when a few of the largest eigenvalues are well separated from the others. In this case, m is small, and we do not lose too much in the exponent. Moreover, $\frac{\beta}{\alpha}$ can be much less than $\kappa = \frac{\lambda_n}{\lambda_1}$. The ratio $\frac{\beta}{\alpha}$ can be seen as an effective condition number.

H. van der Vorst and A. van der Sluis [1085] studied the case of the smallest eigenvalues. They proved the following result. Let us first assume that we have only one isolated eigenvalue.

Theorem 5.12. *Assume that* $0 < \lambda_1 < \alpha \leq \lambda_2 \leq \cdots \leq \lambda_n \leq \beta$, *and let* ℓ *be an integer* $\ell \leq k$ *which will be chosen later on. Then,*

$$\varepsilon_k \leq 4 \left[\frac{1}{C_\ell(\frac{\chi}{\xi})} \right]^2 \left(\frac{\sqrt{\frac{\beta}{\alpha}} - 1}{\sqrt{\frac{\beta}{\alpha}} + 1} \right)^{2(k-\ell)} \varepsilon_0,$$

with

$$\chi = \beta - \xi, \qquad \xi = \frac{\beta - \lambda_1}{1 + \cos(\frac{\pi}{2\ell})}.$$

Proof. We look for a polynomial Q_ℓ of degree ℓ with λ_1 as a root, with $Q_\ell(0) = 1$, and whose maximum over $[\lambda_1, \beta]$ is small. The solution is given by

$$Q_\ell(\lambda) = \frac{C_\ell(\frac{\chi - \lambda}{\xi})}{C_\ell(\frac{\chi}{\xi})},$$

with $\chi = \beta - \xi$ and $\xi = \frac{\beta - \lambda_1}{1 + \cos(\frac{\pi}{2\ell})}$. Since

$$\chi = \frac{\beta \cos(\frac{\pi}{2\ell}) + \lambda_1}{1 + \cos(\frac{\pi}{2\ell})}, \quad \frac{\chi - \lambda}{\xi} = \frac{\cos(\frac{\pi}{2\ell})(\beta - \lambda) + \lambda_1 - \lambda}{\beta - \lambda_1},$$

we have

$$Q_\ell(\lambda_1) = \frac{C_\ell\left(\cos(\frac{\pi}{2\ell})\right)}{C_\ell(\frac{\chi}{\xi})} = 0.$$

The choice for R_k is

$$R_k(\lambda) = Q_\ell(\lambda) \frac{C_{k-\ell}\left(\frac{\alpha+\beta-2\lambda}{\beta-\alpha}\right)}{C_{k-\ell}\left(\frac{\alpha+\beta}{\beta-\alpha}\right)}.$$

We are interested in $\max_i (R_k(\lambda_i))^2$. Clearly $R_k(\lambda_1) = 0$, and

$$\max_i (R_k(\lambda_i))^2 \leq \max_{\lambda \in [\lambda_1, \beta]} (Q_\ell(\lambda))^2 \left(\frac{\sqrt{\frac{\beta}{\alpha}} - 1}{\sqrt{\frac{\beta}{\alpha}} + 1} \right)^{2(k-\ell)}.$$

But,

$$\max_{\lambda \in [\lambda_1, \beta]} |Q_\ell(\lambda)| \leq \frac{1}{C_\ell(\frac{\beta}{\alpha})}.$$

□

To see if there is an improvement, we need a bound on $\frac{1}{C_\ell(\frac{\chi}{\xi})}$. Let us assume that λ_1 is much smaller than β ($\alpha \ll \beta$).

Proposition 5.13. *When $\lambda_1 \ll \beta$, $\frac{1}{C_\ell(\frac{\chi}{\xi})}$ is of the order of $(\frac{\beta}{\lambda_1})^{\frac{1}{\ell}}$.*

Proof. Let $\rho = \frac{\chi}{\xi} = \frac{\cos\theta + \frac{\lambda_1}{\beta}}{1 - \frac{\lambda_1}{\beta}}$ where $\theta = \frac{\pi}{2\ell}$. Then, with our hypothesis,

$$\rho = \frac{\lambda_1}{\beta} + \left(1 + \frac{\lambda_1}{\beta}\right) \cos\theta + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right) = \cos\theta + O\left(\frac{\lambda_1}{\beta}\right).$$

It yields,

$$C_\ell(\rho) = C_\ell(\cos\theta) + (\rho - \cos\theta) C'_\ell(\cos\theta) + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right).$$

But,

$$\rho - \cos\theta = \frac{\lambda_1}{\beta}(1 + \cos\theta) + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right),$$

and $C_\ell(\cos\theta) = 0$. From the properties of the Chebyshev polynomials, one can check that,

$$(1 - \cos^2\theta) C'_\ell(\cos\theta) = \ell C_{\ell-1}(\cos\theta),$$

$$C'_\ell(\cos\theta) = \frac{\ell C_{\ell-1}(\cos\theta)}{\sin^2\theta},$$

and

$$C_\ell(\rho) = \frac{\lambda_1}{\beta} \frac{1 + \cos\theta}{\sin^2\theta} \ell C_{\ell-1}(\cos\theta) + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right).$$

But,

$$\begin{aligned} C_{\ell-1}(\cos\theta) &= \cos((\ell-1)\arccos(\cos\theta) = \cos((\ell-1)\theta), \\ &= \cos\left(\frac{\ell-1}{\ell}\frac{\pi}{2}\right) = \sin\left(\frac{\pi}{2\ell}\right) = \sin\theta. \end{aligned}$$

Hence,

$$C_\ell(\rho) = \frac{\lambda_1}{\beta} \frac{1 + \cos\theta}{\sin\theta} \ell + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right).$$

□

Since $0 \leq \frac{\sin\theta}{1+\cos\theta} \leq 1$, we have the bound

$$\varepsilon_k \leq 4\left[\left(\frac{\beta}{\lambda_1}\right)^2 \frac{1}{\ell^2} + O\left(\left(\frac{\lambda_1}{\beta}\right)^2\right)\right]\left(\frac{\sqrt{\frac{\beta}{\alpha}}-1}{\sqrt{\frac{\beta}{\alpha}}+1}\right)^{2(k-\ell)} \varepsilon_0.$$

This gives us an idea of what the bound looks like when the condition number $\frac{\beta}{\lambda_1}$ is large. If we allow ℓ to be a real number, the value of ℓ that minimizes the upper bound is $\frac{1}{2}\sqrt{\frac{\beta}{\alpha}}$ when $\beta \gg \alpha$. This suggests we can take ℓ as the closest integer to $\frac{1}{2}\sqrt{\frac{\beta}{\alpha}}$. With this value, the upper bound is almost

$$16\,\frac{\alpha\beta}{\lambda_1^2}\left(\frac{\sqrt{\frac{\beta}{\alpha}}-1}{\sqrt{\frac{\beta}{\alpha}}+1}\right)^{2(k-\ell)} \varepsilon_0.$$

The ratio $\frac{\alpha}{\lambda_1}$ is not necessarily very large. The important thing to observe is that the convergence rate depends on the separation of the two first eigenvalues. This result can be generalized to the p smallest eigenvalues. The following theorem is due to H. van der Vorst.

Theorem 5.14. *Let*

$$0 < \lambda_1 < \lambda_2 < \cdots < \lambda_p \leq \alpha \leq \lambda_{p+1} \leq \cdots \leq \lambda_n \leq \beta$$

and let ℓ_j be integers such that $\ell = \sum_{j=1}^{p} \ell_j \leq k$. Then,

$$\varepsilon_k \leq 4\left(\frac{1}{\prod_{j=1}^{p} C_{\ell_j}(\rho_j)}\right)^2 \left(\frac{\sqrt{\frac{\beta}{\alpha}}-1}{\sqrt{\frac{\beta}{\alpha}}+1}\right)^{2(k-\ell)} \varepsilon_0,$$

where,

$$\rho_j = \frac{\beta\cos\frac{\pi}{2\ell_j} + \lambda_j}{\beta - \lambda_j}.$$

□

A detailed study of the case of isolated eigenvalues has also been done by O. Axelsson and G. Lindskog [69]. However, what we have described above are mathematical results. All these papers assumed exact arithmetic and it was not obvious that the bounds they obtained are still valid in finite precision arithmetic. These issues were discussed in a paper by T. Gergelits and Z. Strakoš [520] in 2014. Their statement is

> *We conclude and numerically demonstrate that in the presence of large outlying eigenvalues such bounds have, apart from simple exceptions, little in common with the finite precision behaviour of the CG method.*

We will see how to compute accurate error bounds during the CG iterations in Section 5.6.

5.5 ▪ CG in finite precision arithmetic

In this section we assume $M = I$, but the results also apply to PCG. It has been known almost since the introduction of the Lanczos and CG algorithms in the 1950s that, in finite precision

arithmetic, these algorithms do not satisfy all of their mathematical properties. In particular, there are many cases in which the Lanczos basis vectors and the CG residual vectors lose orthogonality and even linear independence.

The first significant results for explaining the behavior of the Lanczos algorithm for computing eigenvalues in finite precision arithmetic were obtained by C.C. Paige in his Ph.D. thesis [889] in 1971. He improved and extended his results in subsequent papers in journals [890, 891, 892, 894, 895]. He proved the important result that loss of orthogonality has a close relationship with convergence of Ritz values to the eigenvalues of A, and that the Lanczos method can be used efficiently, even in finite precision arithmetic, eventually doing a number of iterations larger than the order of the matrix. For a summary of some of these results, see [816, 825]. The finite precision Lanczos algorithm was also studied by H.D. Simon [1006, 1008, 1007].

Since CG is a nonlinear algorithm (because of the dot products), it is not easy to analyze its finite precision behaviour. Numerically, the CG iterates satisfy

$$\begin{aligned} x_{k+1} &= x_k + \gamma_k p_k + g_k^x, \\ r_{k+1} &= r_k - \gamma_k A p_k + g_k^r, \\ p_{k+1} &= r_{k+1} + \delta_{k+1} p_k + g_k^p, \end{aligned} \tag{5.25}$$

where x_k, r_k, and p_k are the computed vectors and the perturbation terms g_k^x, g_k^r, and g_k^p account for local rounding errors. In finite precision arithmetic, the coefficients γ_k and δ_{k+1} are not computed exactly. One can either consider rounding errors in computing the coefficients like in [1052], or, to incorporate the corresponding rounding errors into the perturbation terms, and assume that the coefficients are computed exactly, see [816]. Let us consider the local orthogonality properties. The local error terms can be bounded as

$$\begin{aligned} \|g_k^r\| &\leq u\|r_k\| \left[1 + \bar{\kappa}(A)\frac{\|r_k\|}{\|p_k\|}\{m+2+(m+n)\bar{\kappa}(A)+n+1\}\right] + O(u^2), \\ \|g_k^p\| &\leq u\|r_{k+1}\| \left[1 + (3+2n)\frac{\|p_k\|}{\|r_k\|^2}\|r_{k+1}\|\right] + O(u^2), \end{aligned}$$

with $\bar{\kappa}(A) = \| \, |A| \, \|/\lambda_1$, λ_1 being the smallest eigenvalue of A, n is the order of A, m is the maximum number of nonzero entries in rows of A and u is the unit roundoff. These inequalities can be written as

$$\begin{aligned} \|g_k^r\| &\leq u\|r_k\| \, (1 + C_k^r) + O(u^2), \\ \|g_k^p\| &\leq u\|r_{k+1}\| \, (1 + C_k^p) + O(u^2). \end{aligned}$$

The positive coefficients C_k^r and C_k^p are bounded because the ratios $\|r_k\|/\|p_k\|$, $\|p_k\|/\|r_k\|$, and $\|r_{k+1}\|/\|r_k\|$ are bounded. It can be shown that, even though r_k may not be exactly orthogonal to r_{k-1}, we have

$$\begin{aligned} |(r_k, r_{k-1})| \leq \kappa(A) \, \frac{\|r_{k-1}\|}{\|p_{k-1}\|} \left[C_{k-1}^A u \frac{\|r_{k-1}\|}{\|p_{k-1}\|} \, \frac{\|r_{k-1}\|^2}{\|r_{k-2}\|^2} + \|r_{k-1}\| \, \|g_{k-2}^p\|\right] \\ + \|g_{k-1}^r\| \, \|r_{k-1}\|, \end{aligned} \tag{5.26}$$

where $\kappa(A)$ is the condition number of A and C_{k-1}^A is a constant involved in the bound

$$|(Ap_{k-2}, p_{k-1})| \leq \lambda_n C_{k-1}^A u,$$

where λ_n is the largest eigenvalue of A.

The two terms within brackets and the last one in the right-hand side of inequality (5.26) are small provided that the ratios are bounded and $\kappa(A)$ is not too large. Local orthogonality is, in general, well satisfied, but global orthogonality can be lost, sometimes even after a small number of iterations. This depends on the distribution of the eigenvalues of A and on the convergence of the Ritz values.

On the foundations provided by C.C. Paige for the Lanczos method for computing eigenvalues, an explanation of the behavior of Lanczos and CG algorithms in finite precision arithmetic was given by A. Greenbaum [560, 561, 562, 564], starting in 1981, see also A. Greenbaum and Z. Strakoš [571]. She showed that the tridiagonal matrix T_k generated at a given iteration k of a perturbed Lanczos recurrence is the same as the one generated by an exact Lanczos recurrence applied to a matrix of larger dimension than A, whose eigenvalues all lie within tiny intervals about the eigenvalues of A, and some special starting vector. These matrix and vector depend on the iteration number k. An analogous statement is valid, within a small inaccuracy, also for the behaviour of finite precision CG. This can be considered as a backward error analysis in a sense different from the usual one, but this is too technical to be reported here.

A. Greenbaum and Z. Strakoš [571, 1051] demonstrated numerically that the behaviour of CG in finite precision arithmetic is very similar to that of the exact algorithm applied to any matrix, say, $\tilde{A}$, which has many eigenvalues spread in tiny intervals about the eigenvalues of A. Note that this is slightly different from Greenbaum's previous results where the matrix depended on the iteration number k. Thus, in practice, CG convergence can be understood by considering the behavior of the exact algorithm applied to such matrices. The size of the intervals is a modest multiple of the unit roundoff. However, as far as we know, this result has not been theoretically proved.

It is well known that, in finite precision arithmetic, the true residual $b - Ax_k$ may differ from the recursively updated residual r_k,

$$r_k = b - Ax_k - f_k.$$

The residual gap f_k was studied by A. Greenbaum in [564] where bounds can be found. In practice, the two residuals are quite close until the maximum attainable accuracy is reached. Then, the true residual norm stagnates whence the norm of the residual computed with the recurrence continues to decrease.

Using results from Greenbaum, Y. Notay investigated rounding error effects on the convergence of CG [859] in 1993. He analysed theoretically and experimentally how finite precision arithmetic affects known bounds on iteration numbers when the spectrum of the system matrix has small or large isolated eigenvalues.

In [893] and in [900] with W. Wülling, C.C. Paige studied the vectors obtained from an orthonormalization process in finite precision arithmetic. Let v_j, $j = 1, \ldots, k+1$ be vectors of unit norm and $V_j = (\, v_1, \ldots, v_j \,)$. Paige introduced a strictly upper triangular matrix S_k,

$$S_k = (I + U_k)^{-1} U_k = U_k (I + U_k)^{-1},$$

where U_k is the strictly upper triangular part of $V_k^* V_k = I + U_k + U_k^*$. This matrix S_k has interesting properties,

$$U_k S_k = S_k U_k, \quad U_k = (I - S_k)^{-1} S_k = S_k (I - S_k)^{-1}, \quad (I - S_k)^{-1} = I + U_k.$$

Moreover, $\|S_k\| \leq 1$, and $V_k^* V_k = I$ if and only if $\|S_k\| = 0$. The matrix $V_k^* V_k$ is singular if and only if $\|S_k\| = 1$. So, S_k is an indicator of the loss of orthogonality. Moreover, S_k is the unique strictly upper triangular matrix of order k such that

$$Q^{(k)} = \begin{pmatrix} Q_{1,1}^{(k)} & Q_{1,2}^{(k)} \\ Q_{2,1}^{(k)} & Q_{2,2}^{(k)} \end{pmatrix} = \begin{pmatrix} S_k & (I - S_k) V_k^* \\ V_k (I - S_k) & I - V_k (I - S_k) V_k^* \end{pmatrix}$$

is a unitary matrix of order $k+n$. Note that in the definition of $Q^{(k)}$ the identity matrices are not all the same order. Then, in [894, 895], Paige applied this result to the suppposedly orthogonal basis vectors computed in the Lanczos algorithm. He showed that the tridiagonal matrix T_k computed by the Lanczos process in finite precision arithmetic can be viewed as the result of a unitary similarity transformation applied to a slightly perturbed, higher dimensional matrix $\mathcal{A}_k$. If the columns of V_{k+1} are the computed Lanczos basis vectors normalized to have unit norm, then

$$[Q^{(k)}]^* \mathcal{A}_k Q_k^{(k)} = \begin{pmatrix} T_k & \beta_k e_k v_{k+1}^* \\ \beta_k v_{k+1} e_k^T & A_k \end{pmatrix}$$

where $Q_k^{(k)}$ was defined above, $A_k = A - \beta_k v_{k+1} v_k^* - \beta_k v_k v_{k+1}^*$, and

$$\mathcal{A}_k = \begin{pmatrix} T_k & 0 \\ 0 & A \end{pmatrix} + H^{(k)}, \quad \|H^{(k)}\| \leq O(u)\|A\|,$$

u being the unit roundoff. From [895] we also have

$$\left[\begin{pmatrix} T_k & 0 \\ 0 & A \end{pmatrix} + H^{(k)} \right] \begin{pmatrix} S_k \\ V_k(I - S_k) \end{pmatrix} = \begin{pmatrix} S_k \\ V_k(I - S_k) \end{pmatrix} T_k + \beta_k \begin{pmatrix} s_{k+1} \\ v_{k+1} - V_k s_{k+1} \end{pmatrix} e_k^T,$$

where $s_{k+1} = (I - S_k)V_k^* v_{k+1}$. This relation appears as to be the exact result of k iterations of an exact Lanczos process with exact orthogonality arising from the Hermitian matrix $\mathcal{A}_k$. This can be seen as a strange backward stability result. It is different from Greenbaum's results, but we observe that in both approaches the extended matrix to be considered at iteration k depends on k. Paige's conclusion for the eigenproblem [895] was that

> *every converged eigenvalue of T_k is essentially an eigenvalue of A, and eigenvalues of the developing T_k never lose their level of convergence. Because this is true for all k, it shows that the Lanczos process is always on track for the eigenproblem, the accuracy of approximation to eigenvalues of A is only limited by the slowly growing size of the backward error $H^{(k)}$.*

In practice, it turns out that the Frobenius norm of $Q_{2,2}^{(k)}$ is decreasing as a function of k and converging to zero (sometimes very slowly). Using the Lanczos basis vectors to solve a linear system was analyzed in [895] when $Q_{2,2}^{(k)} = 0$.

5.6 ▪ CG error estimates

In this section we show how one can compute estimates or even bounds of the A-norm of the error during the CG iterations. We first consider obtaining a lower bound of the error A-norm. The formula in the following theorem appeared in [304] and was proved in [546].

Theorem 5.15. *The square of the A-norm of the error in CG is given by*

$$\varepsilon_k \;=\; \|r_0\|^2 \, \left([T_m^{-1}]_{1,1} - [T_k^{-1}]_{1,1} \right), \tag{5.27}$$

$k = 0, 1, \ldots, m-1$, where m is the grade of r_0 with respect to A, and T_k is the tridiagonal matrix of the Lanczos algorithm.

Proof. Using

$$x - x_k = \|r_0\| \, V_m \left(T_m^{-1} \underline{e}_1 - \begin{pmatrix} T_k^{-1} e_1 \\ 0 \end{pmatrix} \right),$$

$AV_m = V_m T_m$, and the global orthogonality of the Lanczos vectors we obtain

$$\begin{aligned}
\varepsilon_k &= \|r_0\|^2 \left(T_m^{-1}\underline{e}_1 - \begin{pmatrix} T_k^{-1}e_1 \\ 0 \end{pmatrix} \right)^T T_m \left(T_m^{-1}\underline{e}_1 - \begin{pmatrix} T_k^{-1}e_1 \\ 0 \end{pmatrix} \right), \\
&= \|r_0\|^2 \left[\underline{e}_1^T T_m^{-1}\underline{e}_1 - 2\underline{e}_1^T \begin{pmatrix} T_k^{-1}e_1 \\ 0 \end{pmatrix} + \begin{pmatrix} T_k^{-1}e_1 \\ 0 \end{pmatrix}^T T_m \begin{pmatrix} T_k^{-1}e_1 \\ 0 \end{pmatrix} \right] \\
&= \|r_0\|^2 \left[\underline{e}_1^T T_m^{-1}\underline{e}_1 - 2e_1^T T_k^{-1}e_1 + e_1^T T_k^{-1}e_1 \right],
\end{aligned}$$

where $\underline{e}_1$ is the first unit vector of length m and e_1 is of length k. □

Theorem 5.16. *For $k \leq m$, the squared A-norm of the error satisfies*

$$\varepsilon_{k-1} - \varepsilon_k \;=\; \Delta_{k-1}, \qquad \Delta_{k-1} = \gamma_{k-1}\|r_{k-1}\|^2. \tag{5.28}$$

Proof. From the definition of the A-norm, we obtain

$$\varepsilon_{k-1} \;=\; \|x_k - x_{k-1}\|_A^2 + 2(x - x_k)^T A(x_k - x_{k-1}) + \varepsilon_k.$$

The error and residual vectors are linked through $A(x - x_k) = r_k$, and using $x_k - x_{k-1} = \gamma_{k-1}p_{k-1}$ we get

$$\varepsilon_{k-1} - \varepsilon_k \;=\; \gamma_{k-1}^2 p_{k-1}^T A p_{k-1} + 2\gamma_{k-1} r_k^T p_{k-1} = \gamma_{k-1}\|r_{k-1}\|^2,$$

where, in the last equation, we have used the definition of γ_{k-1} and the local orthogonality between two consecutive vectors r_k and p_{k-1}. □

From Theorem 5.16, and for $0 \leq \ell < k \leq m$ it holds that

$$\varepsilon_\ell - \varepsilon_k = \Delta_{\ell:k-1}, \qquad \Delta_{\ell:k-1} = \sum_{j=\ell}^{k-1} \gamma_j \|r_j\|^2 . \tag{5.29}$$

At CG iteration k, the sum $\Delta_{\ell:k-1}$ on the right-hand side of (5.29) is a lower bound on ε_ℓ, that is, the square of the A-norm of the error at iteration ℓ. This bound is sharp if $\varepsilon_k \ll \varepsilon_\ell$, that is, if CG converges fast enough. At iteration k the lower bound improves when we increase the delay $d = k - \ell$, but the iteration number $\ell = k - d$ at which we obtain the bound decreases.

Since, mathematically, $\varepsilon_m = 0$ we obtain an expression for ε_k as a function of the residual norms,

$$\varepsilon_k = \Delta_{k:m-1} = \sum_{j=k}^{m-1} \gamma_j \|r_j\|^2 . \tag{5.30}$$

Unfortunately, at iteration k, most of these residual norms are not known yet. This is why we have to look backwards.

In [629], M.R. Hestenes and E. Stiefel studied the relations of CG with the theory of orthogonal polynomials and continued fractions. They showed that CG implicitly computes orthogonal polynomials corresponding to a dot product defined by a Riemann-Stieltjes integral for a piecewise constant measure. Let the spectral factorization of the matrix A be $A = U\Lambda U^T$. For simplicity, we assume that the eigenvalues of A are distinct and ordered as $\lambda_1 < \lambda_2 < \ldots < \lambda_n$. Let w be a given vector of unit norm. We define ω_i as

$$\omega_i \equiv (w, u_i)^2 \qquad \text{so that} \qquad \sum_{i=1}^{n} \omega_i = 1 , \tag{5.31}$$

and the (nondecreasing) distribution function $\omega(\lambda)$ with a finite number of points of increase $\lambda_1, \lambda_2, \ldots, \lambda_n$ as

$$\omega(\lambda) \equiv \begin{cases} 0 & \text{for} \quad \lambda < \lambda_1 \,, \\ \sum_{j=1}^{i} \omega_j & \text{for} \quad \lambda_i \le \lambda < \lambda_{i+1} \,, \quad 1 \le i \le n-1 \,, \\ 1 & \text{for} \quad \lambda_n \le \lambda \,. \end{cases} \tag{5.32}$$

Then, ω_i is the jump at the point of increase λ_i, see Figure 5.1. Let us assume that $\omega_i \neq 0$,

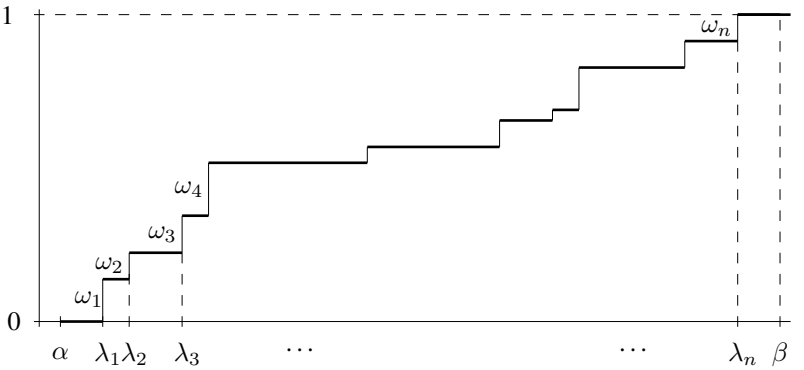


Figure 5.1. *Distribution function* $\omega(\lambda)$

$i = 1, \ldots, n$. With the distribution function $\omega(\lambda)$ and an interval $[\alpha, \beta]$ such that $\alpha < \lambda_1 < \lambda_2 < \ldots < \lambda_n \le \beta$, we can define the Riemann-Stieltjes integral,

$$\int_{\alpha}^{\beta} f(\lambda)\, d\omega(\lambda), \tag{5.33}$$

for any continuous function f. Since $\omega(\lambda)$ is a piecewise constant function, the integral (5.33) can be written as a finite sum and

$$\int_{\alpha}^{\beta} f(\lambda)\, d\omega(\lambda) = \sum_{i=1}^{n} \omega_i f(\lambda_i) \;=\; w^T f(A) w. \tag{5.34}$$

Choosing $f(\lambda) = 1/\lambda$, the quadratic form $w^T A^{-1} w$ can be written as a Riemann-Stieltjes integral. If $w = v_1 = r_0/\|r_0\|$, the jumps ω_i defining the distribution function $\omega(\lambda)$ which corresponds to the Lanczos and CG algorithms are given by

$$\omega_i = (v_1, u_i)^2 = \frac{(r_0, u_i)^2}{\|r_0\|^2} \,, \qquad i = 1, \ldots, n.$$

The Lanczos basis vectors can be written as polynomials in A applied to the initial vector v_1. These polynomials P_k satisfy a three-term recurrence and are orthogonal with respect to the dot product defined by the distribution function $\omega(\lambda)$,

$$\int_{\alpha}^{\beta} P_i(\lambda) P_j(\lambda)\, d\omega(\lambda) = 0, \quad i \neq j.$$

Since the CG residual vectors are proportional to the Lanczos basis vectors, we obtain a similar result for the residuals.

There is also a relation to Gauss quadrature. Using the Jacobi matrices T_k generated by the Lanczos process (and implicitly by CG), a Gauss quadrature rule can be written in the form

$$[f(T_n)]_{1,1} = [f(T_k)]_{1,1} + \mathcal{R}_k^{(G)}[f] \tag{5.35}$$

where the Riemann-Stieltjes integral is given by

$$\int_\alpha^\beta f(\lambda)\, d\omega(\lambda) = [f(T_n)]_{1,1}\,, \tag{5.36}$$

the quantity $[f(T_k)]_{1,1}$ is the Gauss quadrature approximation to the integral, and $\mathcal{R}_k^{(G)}[f]$ is the reminder, see [543]. Using the relation (5.27) we obtain

$$\left[T_n^{-1}\right]_{1,1} = \left[T_k^{-1}\right]_{1,1} + \frac{\varepsilon_k}{\|r_0\|^2}\,. \tag{5.37}$$

which corresponds to a Gauss quadrature rule for $f(\lambda) = \lambda^{-1}$ with the remainder

$$\mathcal{R}_k^{(G)}\left[\lambda^{-1}\right] = \frac{\varepsilon_k}{\|r_0\|^2}.$$

The reminder is nothing other than the scaled and squared A-norm of the error at iteration k. The Gauss quadrature rule (5.37), multiplied by $\|r_0\|^2$, can be written using CG quantities, and the Gauss quadrature approximation is given by

$$\left(T_k^{-1}\right)_{1,1} = \frac{\Delta_{0:k-1}}{\|r_0\|^2}\,.$$

The nodes of the Gauss quadrature rule are the eigenvalues of T_k and the weights are given by the squared first components of the normalized eigenvectors of T_k, see [543].

It was shown in [1052] that the lower bound given by the Gauss rule can also be used safely in finite precision arithmetic since the relation (5.27) is satisfied with only a small inaccuracy.

Upper bounds for the A-norm of the error can be obtained by using a Gauss-Radau quadrature rule with a prescribed node μ on the left of the integration interval. This was mainly developed by G.H. Golub and his collaborators and summarized in the book [543]. To obtain the nodes and weights we have to modify the tridiagonal matrices such that they have a prescribed eigenvalue μ. The modified quadrature rule for the function $f(\lambda) = \lambda^{-1}$ can be written as

$$\varepsilon_0 = \|r_0\|^2 \left[\left(T_{k+1}^{(\mu)}\right)^{-1}\right]_{1,1} + \mathcal{R}_k^{(\mu)}[\lambda^{-1}] \tag{5.38}$$

with

$$T_{k+1}^{(\mu)} = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \ddots & \ddots & & \\ & \ddots & \ddots & \beta_{k-1} & \\ & & \beta_k & \alpha_k & \beta_{k+1} \\ & & & \beta_{k+1} & \alpha_{k+1}^{(\mu)} \end{pmatrix}, \tag{5.39}$$

where the α_j's and β_j's are the Lanczos coefficients, and $\alpha_{k+1}^{(\mu)}$ is such that μ is an eigenvalue of the extended tridiagonal matrix (5.39). Subtracting (5.38) and

$$\varepsilon_0 = \|r_0\|^2 \left[T_\ell^{-1}\right]_{1,1} + \varepsilon_\ell$$

with $\ell = k - d$, we obtain

$$\varepsilon_\ell = \underbrace{\|r_0\|^2 \left(\left[\left(T_{k+1}^{(\mu)} \right)^{-1} \right]_{1,1} - \left[T_\ell^{-1} \right]_{1,1} \right)}_{\Delta_{\ell:k}^{(\mu)}} + \mathcal{R}_k^{(\mu)}[\lambda^{-1}].$$

By adding and subtracting $\left[T_k^{-1} \right]_{1,1}$ and using (5.29), we get

$$\Delta_{\ell:k}^{(\mu)} = \Delta_{\ell:k-1} + \Delta_k^{(\mu)},$$

where

$$\Delta_k^{(\mu)} = \|r_0\|^2 \left(\left[\left(T_{k+1}^{(\mu)} \right)^{-1} \right]_{1,1} - (T_k^{-1})_{1,1} \right). \tag{5.40}$$

The quantity $\Delta_{\ell:k}^{(\mu)}$ is an upper bound on ε_ℓ. In particular, for $k = \ell$ and $k < n$ we obtain

$$\varepsilon_k < \Delta_k^{(\mu)}.$$

How to compute $\alpha_{k+1}^{(\mu)}$ and $\Delta_k^{(\mu)}$ was described in [542, 810, 812, 543]; see also the forthcoming book [830]. It leads to the CGQL algorithm where the Lanczos coefficients needed in the formulas where computed at each iteration from the CG coefficients.

Later on, it was realized by G.M. and P. Tichý [826, 827, 828] that the Gauss-Radau upper bound can be computed directly from the CG coefficients by using a Cholesky factorization of $T_{k+1}^{(\mu)}$. We have

$$\Delta_k^{(\mu)} = \|r_0\|^2 \left(\left[\left(T_{k+1}^{(\mu)} \right)^{-1} \right]_{1,1} - \left(T_k^{-1} \right)_{1,1} \right) = \gamma_k^{(\mu)} \|r_k\|^2,$$

where $\gamma_k^{(\mu)}$ is computed with the recurrence

$$\gamma_0^{(\mu)} = \frac{1}{\mu}, \qquad \gamma_k^{(\mu)} = \frac{\gamma_{k-1}^{(\mu)} - \gamma_{k-1}}{\mu \left(\gamma_{k-1}^{(\mu)} - \gamma_{k-1} \right) + \delta_k},$$

where γ_{k-1} and δ_k are the CG coefficients. Incorporating the computation of lower and upper bounds of the A-norm of the error in CG is particularly simple as it is shown in the following code.

```
function [x,nit,res,estG,estGRu] = cgq(A,b,x0,nitmax,epss,mu,
    delay)
res = zeros(1,nitmax+1);
g = zeros(1,nitmax+1);
g1 = zeros(1,nitmax+1);
estG = zeros(1,nitmax+1); % Gauss lower bound
estGRu = zeros(1,nitmax+1); % Gauss-Radau upper bound
nb = norm(b);
x = x0;
r = b - A * x;
p = r;
```

```
rtr = r' * r;
g1(1) = rtr / mu;
resid = sqrt(rtr);
res(1) = resid;
nit = 0;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 Ap = A * p;
 gamma = rtr / (p' * Ap);
 x = x + gamma * p;
 r = r - gamma * Ap;
 rk = r' * r;
 resid = sqrt(rk);
 res(nit+1) = resid;
 delta = rk / rtr;
 rtro = rtr;
 rtr = rk;
 p = r + delta * p;
 g(nit) = gamma * rtro;
 d1 = g1(nit) - g(nit);
 g1(nit+1) = rtr * d1 / (mu * d1 + rtr);
 t = sum(g(nit-delay+1:nit));
 if nit > delay
  estG(nit-delay) = sqrt(t);
  estGRu(nit-delay) = sqrt(t + g1(nit+1));
 end % if
end % while
res = res(1:nit);
estG = estG(1:nit);
estGRu = estGRu(1:nit);
```

Heuristic algorithms to adaptively compute an almost optimal value of the delay d were proposed in [824] for the Gauss lower bound and in [829] for the Gauss-Radau upper bound. It must be noted that these quadrature-based bounds also hold for PCG with only slight modifications.

How to compute estimates or bounds for the ℓ_2 norm of the error was considered in [815, 830].

There are other ways to obtain bounds or estimates of the error norms. Anti-Gauss quadrature rules were used by L. Reichel and his co-authors, see [199, 197, 15, 16]. However, these rules do not always provide bounds, but just estimates. Their interest is that they do not need approximations of the extreme eigenvalues as in Gauss-Radau rules.

Estimates derived from extrapolation techniques were obtained by C. Brezinski [159]. On that topic, see also [453, 454].

5.7 ▪ Parallel variants of CG

The standard version of CG with three two-term recurrences (HS-CG) is not well suited for parallel computing. On distributed memory computers there must be a synchronization of all the processors after almost every step. The only operations that can be done in parallel are the

updates of the iterate and the residual and maybe some parts of the matrix-vector product. The computation of the two dot products can be a bottleneck since they require a global reduction involving communications between all the processing elements.

Some early attempts of introducing more parallelism in CG were [685, 686, 1091, 260, 259, 318]. To reduce the CG data dependencies we may use the following result.

Proposition 5.17. *In preconditioned CG, we have*

$$(z_{k+1}, r_{k+1}) = \gamma_k^2 (M^{-1} A p_k, A p_k) - (r_k, z_k). \tag{5.41}$$

Proof. We have

$$\begin{aligned} z_{k+1} &= z_k - \gamma_k M^{-1} A p_k, \\ r_{k+1} &= r_k - \gamma_k A p_k. \end{aligned}$$

Multiplying these two equalities, we obtain

$$\gamma_k^2 (M^{-1} A p_k, A p_k) = (z_{k+1} - z_k, r_{k+1} - r_k).$$

But, we have the orthogonality relation $(r_i, z_j) = 0,\ i \neq j$. Therefore,

$$\gamma_k^2 (M^{-1} A p_k, A p_k) = (z_{k+1}, r_{k+1}) + (r_k, z_k).$$

□

Proposition 5.17 shows that, mathematically, we can compute (z_{k+1}, r_{k+1}) before computing r_{k+1} and z_{k+1}. First, we can compute t_k by solving $M t_k = A p_k$, and compute in parallel the two dot products $(t_k, A p_k)$, $(A p_k, p_k)$. Then, we compute γ_k and use relation (5.41) to obtain δ_{k+1}. Finally, we can compute in parallel x_{k+1}, r_{k+1}, z_{k+1}, and p_{k+1}.

However, this algorithm is unstable since using repeatedly relation (5.41) leads to a difference with the true value of (z_{k+1}, r_{k+1}). It was proposed in [802] to use relation (5.41) as a predictor for the value of (z_{k+1}, r_{k+1}), and to compute the true value of this dot product at the beginning of the next iteration. Therefore, at the beginning of iteration k, we have to compute in parallel three dot products, $(t_k, A p_k)$, $(A p_k, p_k)$, and (z_k, r_k). The price to pay for stability is one more dot product per iteration. This method was named predictor-corrector CG in [802].

It was put back to life 35 years later in [247] where a slightly different relation is used for the predictor, and renamed predict-and-recompute PR-CG. In that paper, a rounding error analysis explained why this method is working nicely. A "pipelined" variant was also proposed to be able to compute the matrix-vector product, the action of the preconditioner and dot products simultaneously. This is done by first writing a recurrence for $A p_k$,

$$\begin{aligned} s_k = A p_k &= A z_k + \delta_{k-1} A p_{k-1}, \\ &= w_k + \delta_{k-1} s_{k-1}. \end{aligned}$$

Then, we need a recurrence for w_k,

$$\begin{aligned} w_k = A z_k &= A z_{k-1} - \gamma_{k-1} A \tilde{s}_{k-1}, \\ &= w_{k-1} - \gamma_{k-1} u_{k-1}, \end{aligned}$$

where $\tilde{s}_{k-1} = M^{-1} s_k$, $u_{k-1} = A \tilde{s}_{k-1}$, and M is the preconditioner. Then, we can write

$$\tilde{s}_k = M^{-1} s_k = \tilde{w}_k + \delta_{k-1} \tilde{s}_{k-1},$$

with $\tilde{w}_k = M^{-1}w_k$ which can be computed as

$$\tilde{w}_k = \tilde{w}_{k-1} - \gamma_{k-1}\tilde{u}_{k-1},$$

with $\tilde{u}_k = M^{-1}u_k$.

At iteration k we can first compute in parallel x_k, r_k, z_k, w_k, and $\tilde{w}_k$. Then, we compute the coefficient δ_{k-1} and, in parallel, p_k, s_k, $\tilde{s}_k$, u_k, and $\tilde{u}_k$ from which we can compute the three dot products (p_k, s_k), $(\tilde{s}_k, s_k)$, and (z_k, r_k) which yields the coefficient γ_k. To obtain stability, we have to recompute $w_k = Az_k$ and $\tilde{w}_k = M^{-1}w_k$ for the next iteration.

Recent proposals for parallel CG are [522, 281, 282, 278, 202, 277]. Variants with deep pipelines were proposed in [278]. Although mathematically equivalent to HS-CG, these algorithms do not recompute any dot products or vectors, and the auxiliary recurrences they use can cause rounding errors to be amplified. The use of such pipelined CG variants is potentially dangerous because the maximum attainable accuracy may be much worse than for HS-CG, or they may require a larger number of iterations such that there is no benefit in terms of computing time, see [207]. On the convergence of variants of CG in finite precision arithmetic, see also [568].

5.8 ▪ The conjugate residual method

The conjugate residual method (CR) is constructed by requiring the residual vectors to be A-orthogonal, that is, $(r_i, Ar_j) = 0, i \neq j$. The corresponding iterates x_k minimize the Euclidean norm (ℓ_2-norm) of the residual

$$\|b - Ax_k\| = \min\{\|b - Ay\|, y \in x_0 + \mathcal{K}_k(A, r_0)\}.$$

Here we minimize the ℓ_2-norm of the residual when CG minimizes the A-norm of the error, that is, the A^{-1}-norm of the residual. For the derivation of the algorithm, see R. Chandra [227], G.H. Golub and C.F. Van Loan [547] or B. Fischer [455]. Without preconditioning, the algorithm is the following,

```
function [x,nit,res] = cr(A,b,x0,epss,nitmax)
n = size(A,1);
nb = norm(b);
x = x0;
r = b -A * x;
resid = norm(r);
res = zeros(1,nitmax+1);
res(1) = resid;
w = zeros(n,1);
Aw = w;
rar = 1;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 rar_old = rar;
 Ar = A * r;
 rar = r' * Ar;
 nu = rar / rar_old;
 w = nu * w - r;
 Aw = nu * Aw - Ar;
```

```
 eta = -rar /(Aw' * Aw);
 x = x + eta * w;
 r = r - eta * Aw;
 resid = norm(r);
 res(nit+1) = resid;
end % while
res = res(1:nit);
```

The CR algorithm is well defined if A is SPD. Otherwise, there can be problems computing the denominator of one of the coefficients. In the case where A is indefinite, we will see a stable implementation in the next section. The conjugate residual method was generalized by minimizing $\|r_k\|_{A^{\mu-1}}$, see R. Chandra [227].

5.9 ▪ SYMMLQ and MINRES

We have seen that, for an SPD matrix A, CG can be derived from the Lanczos algorithm by using a Cholesky factorization of the tridiagonal matrix T_k constructed during the Lanczos iterations. This may not be possible when A is symmetric but indefinite, that is, with positive and negative eigenvalues. One way to obtain a stable factorization of T_k would be to eventually use 2×2 pivots. This gives rise to a method known as SYMMBK. C.C Paige and M.A. Saunders [898, 899] proposed instead using an LQ factorization of T_k where L is lower triangular and Q is orthogonal or unitary. Such a factorization always exists. They named their algorithm SYMMLQ. Remember that, when using $x_0 = 0$, the CG approximate solution is obtained (when it exists) by

$$T_k y_k = \|r_0\| e_1, \quad x_k^{CG} = V_k y_k.$$

Let us write the LQ factorization of T_k as

$$T_k = L_k Q_k, \quad Q_k^T Q_k = I,$$

The matrix Q_k is not constructed explicitly, but as the product of matrices of plane rotations. We have

$$T_k = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{pmatrix},$$

Let us consider the first steps of the reduction of T_k to triangular form. We first consider the case $k = 4$ since it is enough to see how the factorization is working. To zero the entry in position $(1, 2)$, we multiply from the right by a rotation matrix denoted by $Q_{1,2}$,

$$Q_{1,2} = \begin{pmatrix} c_1 & s_1 & & \\ s_1 & c_2 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}.$$

To annihilate the $(1, 2)$ entry, we must have $s_1\alpha_1 = c_1\beta_2$. Let $\zeta_1 = \sqrt{\alpha_1^2 + \beta_2^2}$, we take $s_1 = \beta_2/\zeta_1$, $c_1 = \alpha_1/\zeta_1$ and then

$$T_4 Q_{1,2} = \begin{pmatrix} \zeta_1 & & & \\ \omega_2 & \bar{\zeta}_2 & \beta_3 & \\ \pi_3 & \bar{\omega}_3 & \alpha_3 & \\ & & \beta_4 & \alpha_4 \end{pmatrix},$$

with $\bar{\zeta}_2 = (\beta_2\alpha_1\alpha_2)/\zeta_1$, $\pi_3 = s_1\beta_3$, $\bar{\omega}_3 = -c_1\beta_3$. We observe that we have created a fill-in in position $(3,1)$. In the next step, we multiply by

$$Q_{2,3} = \begin{pmatrix} 1 & & & \\ & c_2 & s_2 & \\ & s_2 & -c_2 & \\ & & & 1 \end{pmatrix}.$$

As a result, we obtain

$$T_4 Q_{1,2} Q_{2,3} = \begin{pmatrix} \zeta_1 & & & \\ \omega_2 & \zeta_2 & & \\ \pi_3 & \omega_3 & \bar{\zeta}_3 & \\ & \pi_4 & \bar{\omega}_4 & \alpha_4 \end{pmatrix},$$

$\zeta_2 = \sqrt{\bar{\zeta}_2^2 + \beta_3^2}$, $s_2 = \beta_3/\zeta_2$, $c_2 = \bar{\zeta}_2/\zeta_2$, $\bar{\zeta}_3 = (\beta_3\bar{\omega}_3 - \bar{\zeta}_2\alpha_3)/\zeta_2$, $\bar{\omega}_4 = -c_2\beta_4$, $\pi_4 = s_2\beta_4$.

Now, the general process is clear, $Q_{j,j+1}$ differs from the identity matrix only in the elements $q_{j,j} = -q_{j+1,j+1} = c_j = \cos\theta_j$, $q_{j,j+1} = q_{j+1,j} = s_j = \sin\theta_j$ and

$$T_k Q_{1,2} \cdots Q_{k-1,k} = T_k Q_k^T = L_k = \begin{pmatrix} \zeta_1 & & & & \\ \omega_2 & \zeta_2 & & & \\ \pi_3 & \omega_3 & \zeta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \pi_k & \omega_4 & \bar{\zeta}_k \end{pmatrix}.$$

The last rotation is defined by

$$\zeta_k = \sqrt{\bar{\zeta}_k^2 + \beta_{k+1}^2}, \quad s_k = \frac{\beta_{k+1}}{\zeta_k}, \quad c_k = \frac{\bar{\zeta}_k}{\zeta_k}.$$

Moreover $\bar{\zeta}_{k+2} = -c_k\beta_{k+2}$, $\omega_{k+1} = \bar{\omega}_{k+1}c_k + s_k\alpha_{k+1}$, $\pi_{k+2} = s_k\beta_{k+2}$. Let us define $\tilde{L}_k$ as being identical to L_k except for the (k,k) entry which is replaced by ζ_k. The principal minor of order $k-1$ of L_k is $\tilde{L}_{k-1}$. Following Paige and Saunders [898], we denote

$$\overline{W}_k = [w_1 \cdots w_{k-1}\ \bar{w}_k] = [W_{k-1}\ \bar{w}_k] = V_k Q_k^T,$$

$$\bar{q}_k = (\chi_1, \ldots, \chi_{k-1}, \bar{\chi}_k)^T = ((q_{k-1})^T, \bar{\chi}_k)^T = Q_k y_k.$$

With this notation,

$$L_k\bar{q}_k = \|r_0\|e_1, \quad x_k^{CG} = \overline{W}_k\bar{q}_k.$$

Since $L_k\bar{q}_k = \|r_0\|e_1$ and $\tilde{L}_k q_k = \|r_0\|e_1$ we obtain

$$\chi_k = \bar{\chi}_k\frac{\bar{\zeta}_k}{\zeta_k} = \bar{\chi}_k c_k.$$

By looking at the last two columns of the matrix equality $\overline{W}_{k+1} = V_{k+1}Q_{k+1}^T$, one can see that, since $\overline{W}_{k+1} = [W_k\ \bar{w}_{k+1}]$,

$$[\bar{w}_k\ v_{k+1}]\begin{pmatrix} c_k & s_k \\ s_k & -c_k \end{pmatrix} = [w_k\ \bar{w}_{k+1}], \quad \bar{w}_1 = v_1.$$

It is not necessary to compute x_k^{CG} at each iteration since L_k can be singular. However, $\tilde{L}_k$ is nonsingular as long as $\beta_{k+1} \neq 0$. Hence, Q_k is always well defined. Paige and Saunders chose to compute and update $x_k^S = W_k z_k$ for which we have the update formula

$$x_k^S = x_{k-1}^S + \chi_k w_k,$$

w_k is obtained by applying the plane rotation to $\bar{w}_k$ and v_{k+1}. The CG iterate can be obtained (when it exists) as

$$x_{k+1}^{CG} = x_k^S + \bar{\chi}_{k+1}\bar{w}_{k+1}.$$

Paige and Saunders [898] proved that the residual $r_k^{CG} = b - Ax_k^{CG}$ is given by

$$r_k^{CG} = -\frac{\beta_1 s_1 \cdots s_k}{c_k} v_{k+1}.$$

Therefore, the norm of the residual is available to stop the iteration and we can compute x_{k+1}^{CG} from x_k^S when it is small enough. A code for SYMMLQ is following. To obtain a more robust code one can test for the denominators to be tiny. Clearly, the code is more complicated than for CG since we have to compute the rotations to reduce T_k to a lower triangular matrix.

```
function [x,nit,xcg,res,rescg] = symmlq(A,b,x0,nitmax,epss)
n = size(A,1);
nb = norm(b);
x = x0;
r = b - A * x;
normr = norm(r);
res = zeros(1,nitmax+1);
res(1) = normr;
rescg = zeros(1,nitmax+2);
rescg(1) = normr;
vold = r;
u = vold;
v = u;
beta1 = vold' * v;
beta1 = sqrt(beta1);
vv = v / beta1;
wbar = vv;
v = A * vv;
alpha = vv' * v;
v = v - (alpha/beta1) * vold;
volder = vold;
vold = v;
betaold = beta1;
beta = vold' * v;
beta = sqrt(beta);
zetab = alpha;
deltab = beta;
zeta = sqrt(zetab * zetab + beta * beta);
cs = zetab / zeta;
sn = beta / zeta;
chi = beta1 / zeta;
epsz = 0;
normcgcs = abs(beta1 * sn);
normcg = normcgcs / abs(cs);
rescg(2) = normcg;
resid = normr;
nit = 0;
```

```
while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 vv = v / beta;
 w = cs * wbar + sn * vv;
 x = x + chi * w; % SYMMLQ iterate
 wbar = sn * wbar - cs * vv;
 v = A * vv;
 v = v - (beta / betaold) * volder;
 alpha = vv' * v;
 v = v - (alpha / beta) * vold;
 volder = vold;
 vold = v;
 betaold = beta;
 beta = vold' * v;
 beta = sqrt(beta);
 delta = cs * deltab + sn * alpha;
 deltazeta = - delta * chi;
 zetab = sn * deltab - cs * alpha;
 epsilon = sn * beta;
 deltab = - cs * beta;
 zeta = sqrt(zetab * zetab + beta * beta);
 cs = zetab / zeta;
 sn = beta / zeta;
 epsdz = epsz + deltazeta;
 epsz = -epsilon * chi;
 chi = epsdz / zeta;
 normr = sqrt(epsdz * epsdz + epsz * epsz);
 normcgcs = normcgcs * abs(sn);
 normcg = normcgcs / abs(cs);
 res(nit+1) = normr;
 resid = norm(r);
 rescg(nit+2) = normcg;
 xcg = x + (epsdz/zetab) * wbar; % CG iterate
end % while
res = res(1:nit+1);
rescg = rescg(1:nit+2);
```

B. Fischer [455] described a similar algorithm using a QR factorization instead of the LQ factorization of the Paige and Saunders method. Both methods are mathematically equivalent. Even though the CG iterates can be computed from SYMMLQ, even when the matrix is positive definite, the convergence can be slightly different from CG convergence since the rounding errors are not the same.

The Lanczos basis vectors can be used in a different way. Let us consider iterates written as $x_0 + V_k y$ for some $y \in \mathbb{R}^k$. The residual vector can be written as

$$\begin{aligned} b - A(x_0 + V_k y) &= r_0 - AV_k y, \\ &= \|r_0\| V_k e_1 - V_k T_k y - \beta_{k+1}[y]_k v_{k+1}, \\ &= V_{k+1}(\|r_0\| e_1 - \underline{T}_k y), \end{aligned}$$

where $\underline{T}_k$ is the tridiagonal matrix T_k appended with a row $\beta_{k+1}e_k^T$ at the bottom. One possibility for solving the linear system $Ax = b$ is to require that the vector y minimizes the norm of the residual, that is,

$$y_k^M = \text{argmin}_y \| \|r_0\|e_1 - \underline{T}_k y\|. \tag{5.42}$$

Let us denote the corresponding iterate and residual vector by x_k^M and r_k^M,

$$x_k^M = x_0 + V_k y_k^M, \quad r_k^M = b - Ax_k^M.$$

By construction the residual vector r_k^M is orthogonaal to $A\mathcal{K}_k(A, r_0)$.

This method is called MINRES. It was devised by C.C. Paige and M.A. Saunders [899]. Of course, for the method to be practical, one has to show how to solve the least squares problem (5.42). This can be done in several ways by using QR or LQ factorizations. A code implementing MINRES with Givens rotations is following.

```
function [x,nit,res] = minres(A,b,nitmax,epss)
% we use x_0 = 0
n = size(A,1);
x = zeros(n,1);
y = b;
r1 = b;
beta1 = norm(b);
oldb = 0;
beta = beta1;
dbar = 0;
epsln = 0;
phibar = beta1;
cs = -1;
sn = 0;
w = zeros(n,1);
w2 = zeros(n,1);
r2 = r1;
nr = beta1;
nb = betz1;
res = zeros(1,nitmax);
res(1) = beta;
resid = beta;
nit = 0;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 s = 1 / beta;
 v = s * y;
 y = A * v;
 if nit >= 2
   y = y - (beta / oldb) * r1;
 end % if
 alfa = v' * y;
 y = y - (alfa / beta) * r2 ;
 r1 = r2;
 r2  = y;
```

```
 oldb = beta;
 beta   = norm(y);
 oldeps = epsln;
 delta  = cs * dbar + sn * alfa;
 gbar   = sn * dbar - cs * alfa;
 epsln  = sn * beta;
 dbar   = - cs * beta;
 % compute the next plane rotation
 gamma = norm([gbar beta]);
 gamma = max([gamma eps]);
 cs = gbar / gamma;
 sn = beta / gamma;
 phi = cs * phibar ;
 phibar = sn * phibar ;
 denom = 1 / gamma;
 w1 = w2;
 w2 = w;
 w = (v - oldeps * w1 - delta * w2) * denom;
 x = x + phi * w;
 % estimate of the residual norm
 resid = abs(phibar);
 res(nit+1) = resid;
end % while
res = res(1:nit);
```

Mathematically, MINRES is equivalent to CR. However, since MINRES uses orthogonal transformations, it is more stable and can be used for indefinite symmetric matrices.

It is interesting to consider the relations between the CG and MINRES residual norms. It is a special case of relations between the (quasi-)residual norms of quasi-orthogonal (Q-OR) and quasi-minimum (Q-MR) methods. This is described in chapters 3 and 5 of [823]. We have the following result from [823], Theorem 3.3.

Theorem 5.18. *Let $r_0^M = r_0$. Then,*

$$\frac{1}{\|r_k\|^2} = \frac{1}{\|r_k^M\|^2} - \frac{1}{\|r_{k-1}^M\|^2}. \tag{5.43}$$

By construction the residual norms $\|r_k^M\|$ are decreasing, but, from (5.43), we see that, if they almost stagnate, we can have peaks in the CG residual norm curve for indefinite matrices. On the contrary, if MINRES converges fast, there should not be much difference in the ℓ_2 residual norms of CG and MINRES. The MINRES residual norms can be written as functions of the eigenvalues and eigenvectors of A. From [823] Theorem 5.17, we have the following expressions.

Theorem 5.19. *Let the spectral factorization of A be $U\Lambda U^T$ with distinct eigenvalues λ_i, $= 1, \ldots, n$ and $c = U^T r_0^M$ with no zero component. The MINRES residual norms are given by*

$$\|r_k^M\|^2 = \frac{\sum_{\mathcal{I}_{k+1}} \left[\prod_{j=1}^{k+1} c_{i_j}^2\right] \Theta_\Lambda(\mathcal{I}_{k+1})}{\sum_{\mathcal{I}_k} \left[\prod_{j=1}^{k} c_{i_j}^2 \lambda_{i_j}^2\right] \Theta_\Lambda(\mathcal{I}_k)}, \quad k \geq 1, \tag{5.44}$$

where $\mathcal{I}_k$ is a set of k ordered indices $1 \leq i_1 < \cdots < i_k \leq n$, and $\Theta_\Lambda(\mathcal{I}_k)$ is defined by (5.22).

Using (5.43) and (5.44), we can obtain a (complicated) expression for the CG residual norm as a function of the eigenvalues of A and the projection of the initial residual on the eigenvectors. Once again, it shows that the dependence on the eigenvectors of A is only through $U^T r_0$ and what is important is the eigenvalue distribution through the mutual distances of the eigenvalues.

5.10 ▪ Numerical experiments

Let us start with examples showing the influence of the eigenvalue distribution on CG convergence without preconditioning. For the first example of order 30, we consider four eigenvalue distributions shown in Figure 5.2. The top one (a) has eigenvalues regularly distributed in $[0.1, 100]$, (b) has one isolated simple eigenvalue equal to 0.1 and 100 has multiplicity 5, (c) has one isolated simple eigenvalue equal to 100 and 0.1 has multiplicity 5, and (d) has isolated smallest and largest eigenvalues with multiplicities larger than 1. The matrix is diagonal and the right-hand side is a unit norm vector with equal components $1/\sqrt{30}$.

Figure 5.3 displays the true residual norms. The A-norms of the error are given in Figure 5.4.

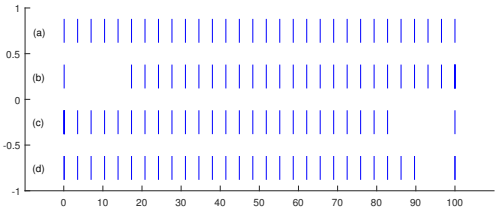


Figure 5.2. *Eigenvalue distributions*

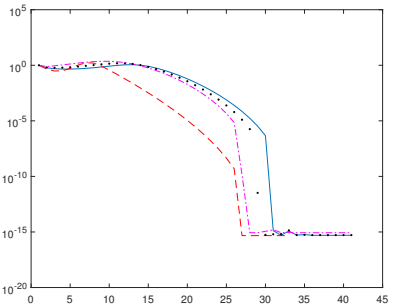


Figure 5.3. *Residual norms, (a) solid, (b) dashed, (c) dot-dashed, (d) dotted*

The worst convergence is for case (a) with a regular distribution. The fastest convergence is obtained for case (b) with one isolated smallest eigenvalue. However, for all cases we obtain the maximum attainable accuracy at or before iteration 30. With such a regular distribution for most of the eigenvalues there is not much growth of the rounding errors, and almost no convergence delay.

The second example with two matrices is more discriminating. The first matrix of order 30

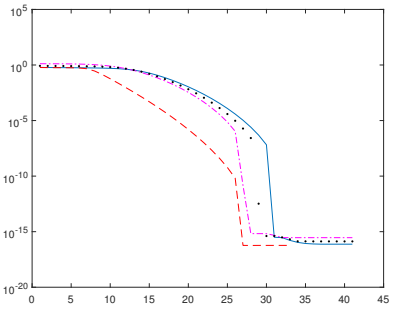


Figure 5.4. *Error A-norms, (a) solid, (b) dashed, (c) dot-dashed, (d) dotted*

is diagonal with eigenvalues

$$\lambda_i = \lambda_{\min} + \frac{i-1}{n-1}\left(\lambda_{\max} - \lambda_{\min}\right)\rho^{n-i},\ i = 1, \ldots, n, \tag{5.45}$$

with $\lambda_{\min} = 0.1$, $\lambda_{\max} = 100$, and $\rho = 0.9$; see [1050]. For the second diagonal matrix, we reverse the distribution and $\hat{\lambda}_i = \lambda_{\min} + \lambda_{\max} - \lambda_i$. The right-hand sides are the same as for the first example and $x_0 = 0$. Figure 5.5 shows the eigenvalue distributions. The first one (a) has well isolated large simple eigenvalues and an accumulation of the smallest eigenvalues towards $\lambda_{\min}$, and for (b) it is the opposite. Figures 5.6 and 5.7 show the residual norms and A-norms of the error.

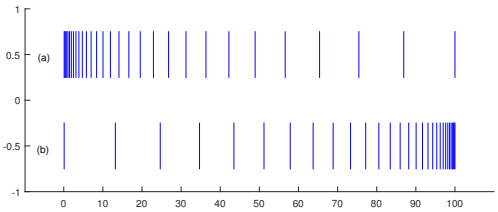


Figure 5.5. *Distribution (5.45), eigenvalue distributions*

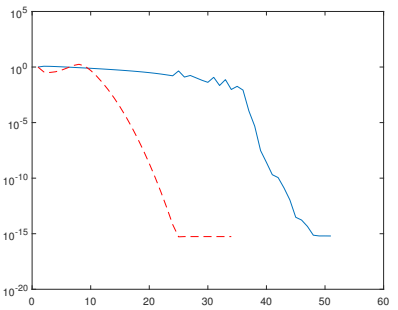


Figure 5.6. *Distribution (5.45), residual norms, (a) solid and (b) dashed*

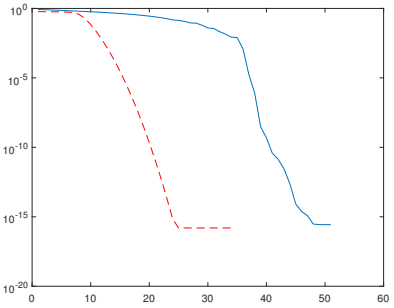


Figure 5.7. *Distribution (5.45), error norms, (a) solid and (b) dashed*

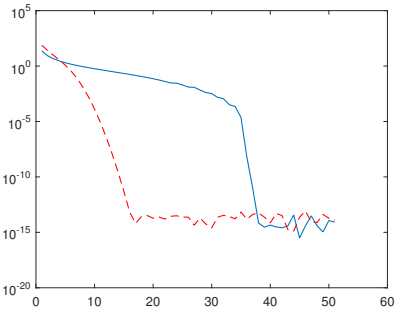


Figure 5.8. *Distribution (5.45), distance of the smallest Ritz value to* $\lambda_{\min}$*, (a) solid and (b) dashed*

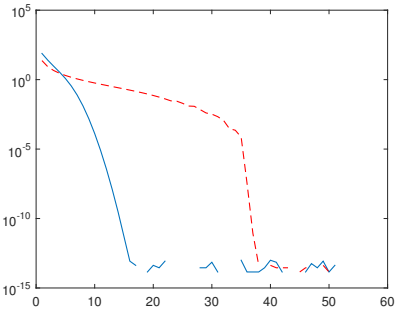


Figure 5.9. *Distribution (5.45), distance of the largest Ritz value to* $\lambda_{\max}$*, (a) solid and (b) dashed*

Figures 5.8 and 5.9 show respectively the distance of the smallest (resp. largest) Ritz value to the smallest (resp. largest) eigenvalue for the two eigenvalue distributions.

With distribution (a), the convergence is hampered by the smallest eigenvalues. Moreover, the largest Ritz value converges quickly to the isolated largest eigenvalue leading to an increase of the rounding errors and a delay of convergence. With that distribution we do not have an acceptable approximate solution in 30 iterations. For distribution (b), the smallest Ritz value converges fast and the largest converges slowly. As shown from figures 5.6 and 5.7 the convergence is much faster with distribution (b) and a solution is obtained in less than 30 iterations.

Tables 5.1 and 5.2 shows results for a set of symmetric positive definite matrices whose characteristics are described in the Appendix. Table 5.1 shows the maximum attainable accuracies for the true residual norm $\|b - Ax_k\|$ and the A-norm of the error. We do a large enough number of iterations to reach these maximum accuracies with $x_0 = 0$. Upper bounds for the residual maximum attainable accuracy were given in [563], but it is difficult to directly relate the maximum attainable accuracy to the norm of the matrix.

Table 5.1. *Maximum attainable accuracies*

Name	n	nb. it.	residual	error
Lap2500	2500	250	$1.7116\ 10^{-13}$	$8.3150\ 10^{-14}$
ash292	292	35	$5.9899\ 10^{-15}$	$2.7221\ 10^{-15}$
bcsstk01	48	200	$2.4502\ 10^{-13}$	$3.7144\ 10^{-16}$
bcsstk01I	48	250	$1.1293\ 10^{-19}$	$4.6561\ 10^{-17}$
bcsstk09	1083	400	$1.4953\ 10^{-13}$	$9.3888\ 10^{-17}$
Pb26	6400	3000	$1.6650\ 10^{-12}$	$3.5749\ 10^{-13}$
1138_bus	1138	4000	$5.5196\ 10^{-10}$	$1.5103\ 10^{-11}$
nos3	960	350	$5.4605\ 10^{-12}$	$1.1298\ 10^{-12}$

Table 5.2 shows the numbers of iterations to obtain $\|r_k\|/\|b\| \leq 10^{-10}$, the residual norms and the A-norms of the error at convergence. One can see that for some matrices we have to do a number of iterations much larger than the order of the matrix to satisfy the stopping criterion. However, these results were obtained without preconditioning to show the behavior of CG.

Table 5.2. *Number of iterations, residual norms, and A-norms of the error, $\epsilon = 10^{-10}$*

Name	n	nb. it.	residual	error	$\|b\|$
Lap2500	2500	164	$1.9303\ 10^{-8}$	$2.0109\ 10^{-8}$	$2.2149\ 10^{2}$
ash292	292	18	$1.0653\ 10^{-9}$	$4.4305\ 10^{-10}$	17.911
bcsstk01	48	148	$6.1769\ 10^{-11}$	$7.3103\ 10^{-14}$	1
bcsstk01I	48	120	$5.8919\ 10^{-15}$	$1.9056\ 10^{-10}$	$9.5294\ 10^{-5}$
bcsstk09	1083	332	$6.9967\ 10^{-11}$	$4.0408\ 10^{-14}$	1
Pb26	6400	1692	$6.2719\ 10^{-8}$	$6.0924\ 10^{-10}$	$7.2839\ 10^{2}$
1138_bus	1138	3430	$3.4700\ 10^{-9}$	$6.0924\ 10^{-10}$	34.548
nos3	960	283	$2.8382\ 10^{-8}$	$3.3501\ 10^{-9}$	$3.1830\ 10^{2}$

In some of these computations the residual norms are oscillating. Figure 5.10 shows the residual norm which is widely oscillating and the A-norm of the error which is monotonely decreasing by construction for the matrix bcsstk01. The two norms are quite different because the norm of the matrix is large. Note that the matrix is of order 48 and we have to do more than 160 iterations to reach the maximum attainable accuracy. Moreover, there is not much convergence before iteration 130. Figure 5.11 shows the same quantities for the matrix Pb26. Here also the residual norms are oscillating. The residual norms and the A-norms of the error are not much different because the norm of the matrix is small.

Figure 5.12 shows the residual norms for the Hestenes and Stiefel (HS-CG) using three two-term recurrences and the three-term variant for bcsstk01. With this matrix there is a large delay of convergence due to rounding errors. We see that residual norms are the same at the beginning of the iterations, but in the end the three-term recurrence variant is worse than HS-CG. Moreover,

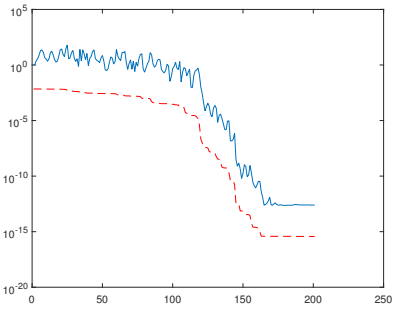


Figure 5.10. *bcsstk01, residual norm (solid) and A-norm of the error (dashed)*

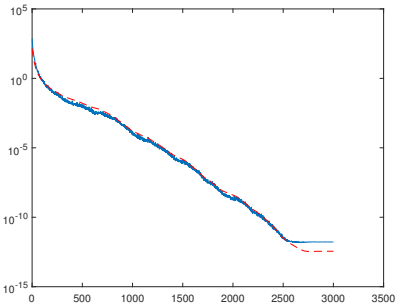


Figure 5.11. *Pb26, residual norm (solid) and A-norm of the error (dashed)*

its maximum attainable accuracy for the residual norm is larger by several orders of magnitude. The interest of the three-term variant is that it is more suited to parallel computing.

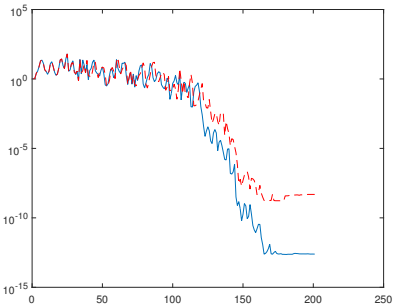


Figure 5.12. *bcsstk01, residual norm HS-CG (solid) and three-term recurrences (dashed)*

Let us do some numerical experiments with the quadrature-based bounds for the A-norm of the error in CG. We first use the matrix Lap2500. The Gauss and Gauss-Radau bounds are shown in Figure 5.13. We used a delay $d = 1$ and $\mu = (1 - 10^{-8})\,\lambda_1$ for Gauss-Radau. One can see that we obtain very good bounds that can be used in a stopping criterion.

Figure 5.14 shows the bounds for bcsstk01, $d = 1$, and $\mu = (1 - 10^{-8})\,\lambda_1$. We see that

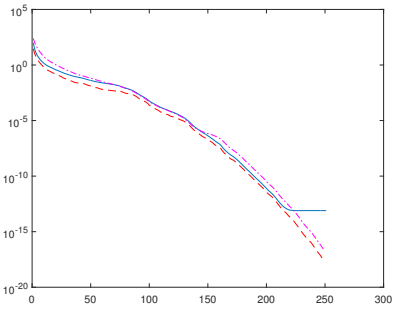


Figure 5.13. *Lap2500, A-norm of the error (solid), Gauss lower bound (dashed), Gauss-Radau upper bound (dot-dashed),* $d = 1$

the Gauss lower is widely oscillating. This is because of the oscillations of the CG residual norm. The Gauss-Radau upper bound is sharp in the beginning of the iterations and its accuracy deteriorates in the fast convergence phase. This phenomenon is analyzed and explained in [829]. It is linked to the convergence of the smallest Ritz value to the smallest eigenvalue of A.

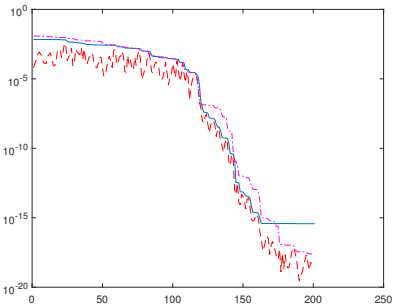


Figure 5.14. *bcsttk01, A-norm of the error (solid), Gauss lower bound (dashed), Gauss-Radau upper bound (dot-dashed),* $d = 1$

As one can see in Figure 5.15, increasing the delay to $d = 5$ improves the results. There are less oscillations in the lower bound and the upper bound is more accurate in the last iterations.

We now consider sequential versions of some of the parallel CG variants. First we use the problem with eigenvalues defined by (5.45) with $\lambda_{\min} = 0.1$, $\lambda_{\max} = 100$, and $\rho = 0.9$. Figure 5.16 shows the residual norms for HS-CG, GV-CG [522], and CV-CG [280]. The convergence of the three methods is almost the same but the parallel variants have a much worse attainable accuracy by several orders of magnitude.

Figure 5.17 displays the residual norms for HS-CG, M-CG [802], and CG-CG [260, 259]. The maximum attainable accuracies are much better than what we have in the methods without recomputations.

The residual norms for pipelined variants Pipe-M-CG and Pipe-PR-CG described in [247] are shown in Figure 5.18. The maximum attainable accuracies are a little bit worse, but still satisfactory.

The differences in accuracy depend on the example. For instance, with Lap2500 all the methods used here give more or less the same residual norms as HS-CG. There are more differences

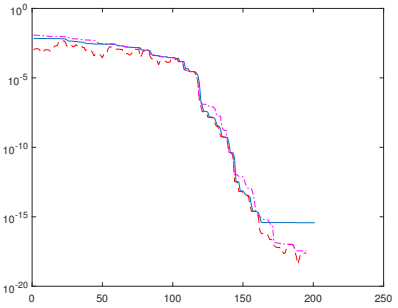


Figure 5.15. *bcsttk01, A-norm of the error (solid), Gauss lower bound (dashed), Gauss-Radau upper bound (dot-dashed),* $d = 5$

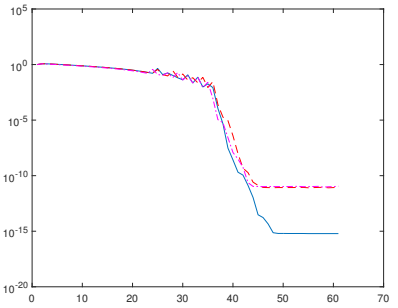


Figure 5.16. *Distribution (5.45), residual norms, HS-CG (solid), GV-CG (dashed), CV-CG (dot-dashed)*

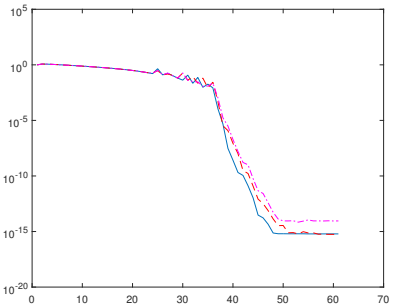


Figure 5.17. *Distribution (5.45), residual norms, HS-CG (solid), M-CG (dashed), CG-CG (dot-dashed)*

with bcsstk01 as one can see in Figure 5.19 with convergence delays for GV-CG and CV-CG and much worse attainable accuracies than for HS-CG. Convergence is much better with M-CG and CG-CG, see Figure 5.20. Figure 5.21 displays the residual norms for the pipelined variants.

Let us now consider other methods derived from the Lanczos algorithm. Let us choose a problem easy to solve with the SPD matrix Lap2500. Figure 5.22 shows the true residual norms for CG, MINRES, SYMMLQ, and the CG iterates obtained from SYMMLQ. The MINRES residual norms are monotonely decreasing by construction and not much different from the CG

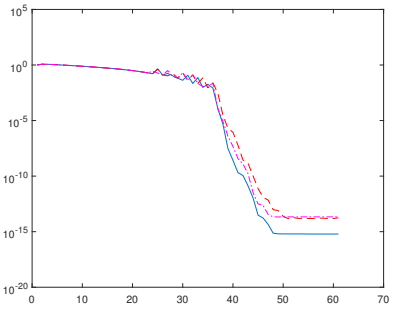


Figure 5.18. *Distribution (5.45), residual norms, HS-CG (solid), Pipe-M-CG (dashed), Pipe-PR-CG (dot-dashed)*

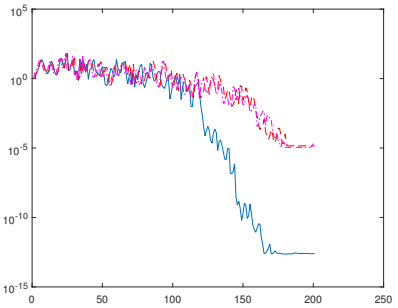


Figure 5.19. *bcsstk01, residual norms, HS-CG (solid), GV-CG (dashed), CV-CG (dot-dashed)*

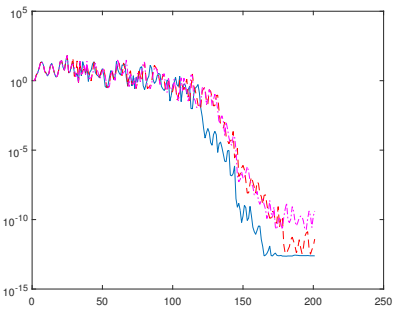


Figure 5.20. *bcsstk01, residual norms, HS-CG (solid), M-CG (dashed), CG-CG (dot-dashed)*

residual norms. The SYMMLQ norms are larger but the CG norms obtained from SYMMLQ are almost on top of the CG residual norms. All the methods converge similarly.

To obtain a problem with an indefinite matrix we negatively shift the matrix A, $\hat{A} = A - 0.1I$. The shifted matrix $\hat{A}$ has 17 negative eigenvalues. From Figure 5.23 we see that CG is converging (even though the matrix is indefinite), but with many oscillations. Moreover, the convergence is much slower than for the SPD matrix A with almost no progress for the first 200 iterations. We show the residual norms because when the matrix is indefinite it does not define a

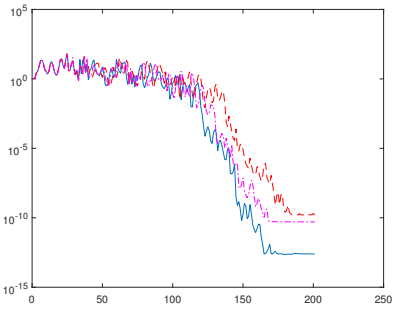


Figure 5.21. *bcsstk01, residual norms, HS-CG (solid), Pipe-M-CG (dashed), Pipe-PR-CG (dot-dashed)*

norm. MINRES converges monotonely, but not much faster than CG, except in the last iterations. SYMMLQ residual norms are a little larger. The CG norms obtained from SYMMLQ match the genuine CG norms at the beginning, but they became different in the fast convergence phase.

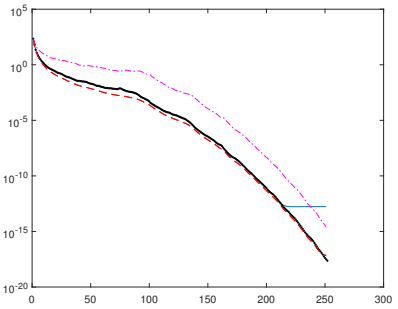


Figure 5.22. *Lap2500, residual norms, CG (solid), MINRES (dashed), SYMMLQ (dot-dashed), CG-SYMMLQ (dotted)*

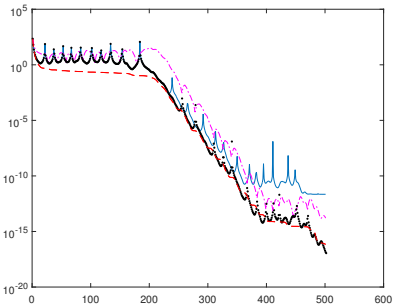


Figure 5.23. *Lap2500, shift=*-0.1*, residual norms, CG (solid), MINRES (dashed), SYMMLQ (dot-dashed), CG-SYMMLQ (dotted)*

Figure 5.24 shows the residual norms for a larger negative shift with $\hat{A} = A - I$, this nonsingular matrix having 205 negative eigenvalues. We have a stagnation for almost 800 iterations and

a lot of oscillations for CG. The other methods converge similarly, but much more slowly than for the two previous examples. In the fast convergence phase, the residual norms of CG-SYMMLQ are quite different from the CG norms. Clearly, CG is not reliable for indefinite matrices, but it was not derived for that. There can even be divisions by zero or very tiny quantities.

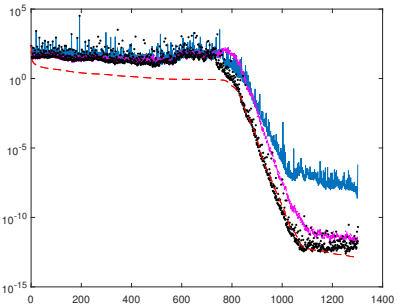


Figure 5.24. *Lap2500, shift=−1, residual norms, CG (solid), MINRES (dashed), SYMMLQ (dot-dashed), CG-SYMMLQ (dotted)*

5.11 ▪ Historical and bibliographical comments

CG was developed independently by M.R Hestenes in the USA and E. Stiefel in Switzerland at the beginning of the 1950s. After meeting at the Institute of Numerical Analysis (INA) of the National Bureau of Standards in 1951, they realized that their algorithms were similar and they wrote a famous and seminal joint paper [629] which was published in the December 1952 issue of the Journal of the National Bureau of Standards. In that paper one can read

> *The method of Conjugate Gradients was developed independently by E. Stiefel of the Institute of Applied Mathematics at Zürich and by M.R. Hestenes with the co-operation of J.B. Rosser, G. Forsythe and L. Paige of the Institute for Numerical Analysis, National Bureau of Standards. The present account was prepared jointly by M.R. Hestenes and E. Stiefel during the latter's stay at the National Bureau of Standards. The first papers on this method were given by E. Stiefel [1952] and by M.R. Hestenes [1951]. Reports on this method were given by E. Stiefel and J.B. Rosser at a symposium on August 23-25, 1951. Recently, C. Lanczos [1952] developed a closely related routine based on his earlier paper on eigenvalue problems [1950]. Examples and numerical tests of the method have been by R. Hayes, U. Hoschstrasser and M. Stein.*

Many of the ideas that were developed later, including preconditioning, are already described in [629]; see also Hestenes [626] and Hestenes' book [628].

At the beginning of the 1950s C. Lanczos was also working at the INA. In [721] published in 1950, he proposed constructing an orthogonal basis of what we now call a Krylov subspace. He used the algorithm on practical problems: the lateral vibration of a bar, the vibrations of a membrane, and a string. In the paper [722] dated September 1951 but published in 1952, he discussed the solution of linear systems.

In the paper [876] in 1996, D.P. O'Leary reported a personal communication from Hestenes who said

I believe it was done in the following order. 1. Stiefel because he had carried out some large experiments which surely took place more than a month before he came to UCLA. I invented it within a month of his arrival. 2. Hestenes. 3. Lanczos. He is third because he would have been talking about it prior to my invention of the routine. I am sure that when he saw my paper he said to himself, "I knew it all along". The remarkable thing is that it took two years of study of iterative methods at INA before the cg-algorithm was devised.

A booklet [424] was published in 1959 by M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel from the ETH Zürich. For the numerical experiments they considered two problems in elasticity analysis: a plate problem with 70 unknowns and a condition number of 1600, and a bar problem with 44 unknowns and a condition number of 7 10^5. These problems where solved on the ERMETH, a computer built at ETH. The plate problem was also solved on an IBM 704 using a refined grid with 270 unknowns and a condition number of 22,000. For the small plate problem on the ERMETH, the computing time with CG was around 150 mn to obtain a reduction of 10^7 of the ℓ_2 norm of the residual.

In the 1960s, people started to solve larger linear systems and CG began to acquire a mixed reputation because of the effect of rounding errors. Moreover, the 1950s and 1960s were the golden age of the relaxation methods. For instance, CG is not cited in the seminal book about iterative methods [1098] by R.S. Varga in 1962.

Things started to change at the beginning of the 1970s. We should give credit to J.K. Reid [938] for having shown that the method could be interesting for iteratively solving "large" sparse linear systems. What makes the success of CG in the 1970s was its coupling to efficient preconditioners which gave favorable eigenvalue distributions for fast convergence. Another seminal paper that renewed the interest in CG was [272] by P. Concus, G.H. Golub, and D.P. O'Leary published in 1976 in the proceedings of the Symposium on Sparse Matrix Computations held at Argonne National Laboratory in September 1975.

In 1980, D.P. O'Leary developed the block conjugate gradient algorithm [874] whose main goal is to solve systems $AX = B$, where B and X are $n \times s$ matrices.

After CG was recognized as a method of interest in the beginning of the 1970s, research was resumed to try to analyze and explain CG convergence. The first most significant results for explaining the behavior of the Lanczos algorithm for computing eigenvalues in finite precision arithmetic were obtained by C.C. Paige in his Ph.D. thesis in 1971 [889] and improved and extended in his subsequent papers in journals [890, 891, 892].

In 1975, G.W. Stewart studied the case with all the eigenvalues of A are contained in an interval $[\alpha, \beta]$ except for one outlier λ_1; see [1041]. In 1976, O. Axelsson gave bounds for the A-norm of the error when there are large outliers in the eigenvalues [57]. The influence of the eigenvalue spectrum on the convergence rate of CG was also studied by A. Jennings [677] in 1977. H.A. van der Vorst and A. van der Sluis studied the case of separated smallest eigenvalues [1085].

On the foundations provided by C.C. Paige for the Lanczos method for computing eigenvalues, an explanation of the behavior of Lanczos and CG algorithms in finite precision arithmetic was given by A. Greenbaum [560, 561, 562, 564], starting in 1981. A parametrized class of matrices for which the rate of convergence of CG varies greatly with the parameter choices was constructed by Z. Strakoš [1050] in 1991 with a distribution of eigenvalues defined as in (5.45). For some choices, the orthogonality of the residual vectors is lost very rapidly. A. Greenbaum and Z. Strakoš [571, 1051] demonstrated numerically that the behaviour of CG in finite precision arithmetic is very similar to that of the exact algorithm applied to any matrix, say, $\tilde{A}$, which has many eigenvalues spread in tiny intervals about the eigenvalues of A. Using results from Greenbaum, Y. Notay investigated rounding error effects on the convergence of CG [859] in

1993. The maximum attainable accuracy of recursively computed residual methods was studied by A. Greenbaum [563].

In 2000, M.H. Gutknecht and Z. Strakoš studied the accuracy of two three-term and three two-term recurrences for Krylov subspace solvers [595].

The main results about the Lanczos and CG algorithms in finite precision arithmetic were summarized and explained in G.M.'s book [816] and in the joint paper with Z. Strakoš [825] in 2006.

An interesting line of research about CG was started by G.H. Golub in the 1970s. His goal was to find upper and lower bounds for the norm of the error; see the joint papers with G. Dahlquist and S.C. Eisenstat [303] in 1972, and with G. Dahlquist and S.G. Nash [304] in 1978. This technique was further developed for CG by G.H. Golub in papers with B. Fischer [457] in 1993, with Z. Strakoš [546] in 194, and G.M. [541, 542]; see also [810, 812, 815] by G.M., and [1052, 1053] by Z. Strakoš and P. Tichý.

The work of G.H. Golub and G.M. was summarized in the book [543] published in 2010. Improvements of the computation of the error bounds were obtained by G.M. and P. Tichý [826, 827, 828, 829], and [824] with Jan Papež.

Computation of bounds for matrix function entries and/or for norms of the error in CG was also considered by D. Calvetti, G.H. Golub, and L. Reichel [196] in 1999 and D. Calvetti, S. Morigi, L. Reichel, and F. Sgallari [197, 198] in 2000-2001. These publications used anti-Gauss quadrature rules that were introduced by D.P. Laurie [724] in 1996; see also [199].

Other papers concerned with computing bounds for the norm of the error in methods related to CG are by C.Brezinski [159] in 1999, by A. Frommer, K. Kahl, T. Lippert, and H. Rittich [479] in 2013, R. Estrin, D. Orban, and M.A. Saunders [429, 428] in 2019, and E. Hallman [611] in 2020.

Probably the first proposal to introduce parallelism in CG was made by J. Van Rosendale in a report for the Institute for Computer Applications in Science and Engineering (ICASE) of NASA [1091] in 1983; see also the papers by L. Johnson [685, 686] in 1983-1984. The predictor-corrector method was proposed by G.M. [800] in 1984, using a formula derived by Y. Saad [973]. This technique was also used in the report [801] in 1985; see also [239, 807]. A paper [802] reporting numerical experiments was published in 1987. The algorithm was also described in the book [811].

Another method was described by A.T. Chronopoulos and C.W. Gear [260, 259] in 1989. It is called the s-step conjugate gradient method. This work was an outcome of Chronopoulos' Ph.D. thesis [258] where several s-step methods were proposed.

In 2014, P. Ghysels and W. Vanroose proposed a preconditioned pipelined CG method [522] that has a single non-blocking reduction per iteration. The numerical results showed that this algorithm gave a maximum attainable accuracy which can be much worse than with the standard HS-CG. This was pointed out by E.C. Carson, M. Rozložník, Z. Strakoš, P. Tichý, and M. Tůma [207] in 2018. This was followed by papers by S. Cools, E.F. Yetkin, E. Agullo, L. Giraud, and W. Vanroose [282], as well as [283] by J. Cornelis, S. Cools, and W. Vanroose, and [276] with P. Ghysels.

G.M.'s 1984 predictor-corrector idea [800] was put back to life more than 35 years later and extended by E.C. Carson and T. Chen [247] in 2020. They renamed the technique as predict-and-recompute.

The methods for indefinite matrices were mainly studied by C.C. Paige and M.A. Saunders [898, 899], see also R. Chandra [227] and the nice book by B. Fischer [455].

For the history of CG up to 1976, see the nice paper by G.H. Golub and D.P. O'Leary [544] published in 1989. For more details on the history of iterative methods, see Chapter 8 of the book [160] by C. Brezinski, G.M., and M. Redivo-Zaglia in 2022.

6

Krylov methods for nonsymmetric linear systems

In this chapter we study some Krylov methods for solving linear systems $Ax = b$ where the matrix A is nonsingular and nonsymmetric. Since the 1970s many such methods have been proposed. We only consider those which are the most used. In the paper [396] by M. Eiermann and O.G. Ernst it is stated that almost any Krylov method can be seen as a (quasi-) orthogonal residual (Q-OR) or (quasi-) minimal residual (Q-OR) method. This is why we start by considering this general abstract class of methods. The existing Q-OR/Q-MR methods mainly differ in the type of basis used for the Krylov subspace.

To motivate what we are doing in this chapter, consider a matrix B whose columns give a basis of $\mathbb{R}^n$. We can write the solution of $Ax = b$ as $x = By$. Let us write $AB = BZ$. Then, we have $BZy = b$ and the vector y is given by solving $Zy = B^{-1}b$. For this process to work we need that $B^{-1}b$ is easy to compute and that $Zy = B^{-1}b$ is easy to solve. The simplest form for Z would be a diagonal matrix, but B would be the matrix of the eigenvectors of A provided A is diagonalizable. Generally, we do not know the eigenvectors of A. The next possibility would be to have Z as an upper triangular matrix. This would be a Schur decomposition. However, the first column of B would have to be an eigenvector of A, and this, again, is not feasible. What we will do in the next sections is to have Z as an upper Hessenberg matrix. We will see that the basis can be computed column by column with only matrix-vector multiplications.

6.1 ▪ Q-OR and Q-MR methods

Let $r_0 = b - Ax_0$ be the initial residual vector. We assume that we have an ascending basis of the nested Krylov subspaces $\mathcal{K}_k(A, r_0)$. It means that, if $v_1, \ldots, v_k$ are the basis vectors, then $v_1, \ldots, v_k, v_{k+1}$ are the basis vectors for $\mathcal{K}_{k+1}(A, r_0)$ as long as $k + 1 \leq m$ where m is the grade of r_0 with respect to A. For such indices, the vector Av_k is in $\mathcal{K}_{k+1}(A, r_0)$ and we can write it in the basis $v_1, \ldots, v_k, v_{k+1}$ as

$$Av_k = \sum_{j=1}^{k+1} h_{j,k} v_j. \tag{6.1}$$

We assume that the basis vectors are of unit norm and then write

$$h_{k+1,k} v_{k+1} = Av_k - \sum_{j=1}^{k} h_{j,k} v_j,$$

with $h_{k+1,k} = \|Av_k - \sum_{j=1}^{k} h_{j,k}v_j\|$ real and positive. By induction the vector v_{k+1} can be expressed as a polynomial of degree k in A applied to v_1 or r_0. Because of relation (6.1) the basis vectors satisfy

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T = V_{k+1} \underline{H}_k, \tag{6.2}$$

where H_k is an upper Hessenberg matrix with entries $h_{i,j}$, the columns of V_k are the basis vectors $v_1, \ldots, v_k$, and e_k is the last column of the identity matrix of order k. The matrix $\underline{H}_k$ is H_k appended with the k first entries of the $(k+1)$st row of H_{k+1}. The matrix H_k is the leading principal submatrix of order k of a larger upper Hessenberg matrix H in the decomposition

$$AV = VH \tag{6.3}$$

valid when $k = m$ in (6.2). Most of the time we will assume that $m = n$ in which case H is a square matrix.

The iterates x_k, $k \geq 1$ in Q-OR and Q-MR methods are of the form

$$x_k = x_0 + V_k y_k, \tag{6.4}$$

for some unique vector $y_k \in \mathbb{R}^k$ or $\mathbb{C}^k$ if the data is complex. It means that we look for x_k in $x_0 + \mathcal{K}_k(A, r_0)$. Since one chooses $v_1 = r_0/\|r_0\|$, the residual vector r_k, defined as $r_k = b - Ax_k$, can be written as

$$\begin{aligned} r_k &= b - Ax_k = b - Ax_0 - AV_k y_k \\ &= \|r_0\| V_k e_1 - AV_k y_k \\ &= V_k (\|r_0\| e_1 - H_k y_k) - h_{k+1,k}[y_k]_k v_{k+1}. \end{aligned} \tag{6.5}$$

In a Q-OR method, the kth iterate x_k^O is defined (provided that H_k is nonsingular) by computing y_k in (6.4) as the solution of the (small) linear system

$$H_k y_k = \|r_0\| e_1. \tag{6.6}$$

This annihilates the term within parenthesis in the rightmost expression of (6.5). We denote the solution of equation (6.6) by y_k^O. The iterates of the Q-OR method are $x_k^O = x_0 + \|r_0\| V_k H_k^{-1} e_1$, the residual vector, which we denote as r_k^O, is proportional to v_{k+1}, and

$$\|r_k^O\| = h_{k+1,k} \left| [y_k^O]_k \right|.$$

In case H_k is singular and x_k^O is not defined, we define the residual norm as being infinite, $\|r_k^O\| = \infty$.

The residual vector in (6.5) can also be written as

$$r_k = V_{k+1}(\|r_0\| e_1 - \underline{H}_k y_k). \tag{6.7}$$

Instead of annihilating the term within parenthesis in the rightmost expression of (6.5), it could be nice to minimize the norm of the residual itself. But this is costly if the columns of the matrix V_{k+1} are not orthonormal. However, we can bound the residual norm,

$$\|r_k\| \leq \|V_{k+1}\| \, \| \|r_0\| e_1 - \underline{H}_k y_k\|.$$

In a Q-MR method where the basis may not be orthogonal, the vector y_k^M is computed as the solution of the least squares problem

$$\min_y \| \|r_0\| e_1 - \underline{H}_k y\|. \tag{6.8}$$

The solution y_k^M is usually computed, just like the solution of (6.6), using Givens rotations to zero the subdiagonal entries of $\underline{H}_k$ and then solving an upper triangular system, see Section 1.18. Note that if $\|V_{k+1}\| \neq 1$, y_k^M does not minimize the norm of the residual but the norm of what is called the quasi-residual defined as

$$z_k^M = \|r_0\| e_1 - \underline{H}_k y_k^M. \tag{6.9}$$

However, since the basis vectors are of unit norm, we have

$$\|V_{k+1}\| \leq \|V_{k+1}\|_F = \sqrt{k+1},$$

where $\|\cdot\|_F$ is the Frobenius norm. The Q-MR iterates are always defined contrary to the Q-OR iterates when H_k is singular.

Q-OR and Q-MR methods are studied in detail in the book [823]. Let us cite some of the main results without proofs.

Theorem 6.1. *Let C be the companion matrix corresponding to the eigenvalues of the non-derogatory matrix A, let $r_0 = \|r_0\| Ve_1$ for a nonsingular matrix V and let the Krylov matrix $K = (\, r_0 \quad Ar_0 \quad \cdots \quad A^{n-1}r_0\,)$ be nonsingular. K can be decomposed as*

$$K = \|r_0\| VU, \tag{6.10}$$

for a nonsingular upper triangular matrix U if and only if $Ue_1 = e_1$ and

$$AV = VH, \tag{6.11}$$

for an unreduced upper Hessenberg matrix H satisfying $H = UCU^{-1}$.

The matrix U in the decomposition of H represents the change of basis from V to the Krylov matrix K. There is a relationship between the Q-OR and Q-MR iterates.

From relation (6.5), if there is an index k for which $h_{k+1,k} = 0$ the Q-OR residual vector is $r_k^O = 0$ and x_k^O is the solution of the linear system. This is also what is obtained with the Q-MR method since the last row of the matrix $\underline{H}_k$ is then zero. This index cannot be smaller than $d(A, r_0)$ (the grade of r_0 with respect to A) since, otherwise, we would obtain an annihilating polynomial $p(A)r_0 = 0$ of degree smaller than $d(A, r_0)$. It means that, mathematically, Q-OR/Q-MR methods are direct methods providing the exact solution after a finite number of steps, even though they are used as iterative methods.

Theorem 6.2. *Assume that $h_{k+1,k} \neq 0$. The vectors of coefficients y_k^O and y_k^M in a Q-OR and a corresponding Q-MR method satisfy*

$$y_k^M = y_k^O - \gamma\, [y_k^O]_k\, H_k^{-1} H_k^{-*} e_k,$$

with

$$\gamma = \frac{h_{k+1,k}^2}{1 + h_{k+1,k}^2 \|H_k^{-*} e_k\|^2}.$$

For the iterates and the error vectors we have

$$x_k^M = x_k^O - \gamma\, [y_k^O]_k\, V_k\, H_k^{-1} H_k^{-*} e_k \tag{6.12}$$

and

$$\epsilon_k^M = \epsilon_k^O - \gamma\, [y_k^O]_k\, V_k\, H_k^{-1} H_k^{-*} e_k.$$

For the quasi-residual vector we have

$$z_k^M = \gamma\,[y_k^O]_k \begin{pmatrix} H_k^{-*} e_k \\ -\frac{1}{h_{k+1,k}} \end{pmatrix}. \tag{6.13}$$

From Theorem 6.2 we obtain a relation between the norm of the Q-OR residual and the norm of the Q-MR quasi-residual.

Proposition 6.3. *The norm of the Q-MR quasi-residual z_k^M is linked to the norm of the Q-OR residual by*

$$\|z_k^M\|^2 = \frac{1}{1 + h_{k+1,k}^2 \|H_k^{-*} e_k\|^2} \|r_k^O\|^2. \tag{6.14}$$

The quasi-residual norms can also be related to the sines and cosines of the Givens rotations used to bring H_k to upper triangular form.

Theorem 6.4. *Let s_j and c_j be the sines and cosines in the Givens rotations used to bring H_k to upper triangular form. Then the norms of the Q-MR quasi-residuals are*

$$\|z_k^M\| = \|r_0\|\,|s_1 s_2 \cdots s_k|. \tag{6.15}$$

We have the following relations between the residual norms of the Q-OR method and the quasi-residual norms of the associated Q-MR method,

$$\|z_k^M\| = |c_k|\,\|r_k^O\|, \tag{6.16}$$

$$\frac{1}{\|r_k^O\|^2} = \frac{1}{\|z_k^M\|^2} - \frac{1}{\|z_{k-1}^M\|^2}. \tag{6.17}$$

Relation (6.17) is responsible for the well-known peak-plateau phenomenon, see [174, 297]. If the quasi-residual norms of the Q-MR method stagnate at iterations $k-1$ and k, the iterate of the corresponding Q-OR method is not defined, that is, H_k is singular. More generally, if we have near stagnation of the Q-MR method we have peaks for the Q-OR residual norm.

From Theorem 6.4 we have

$$1 + h_{k+1,k}^2 \|H_k^{-*} e_k\|^2 = \frac{1}{|c_k|^2} = \frac{\|r_k^O\|^2}{\|z_k^M\|^2}. \tag{6.18}$$

Hence the coefficient γ in Theorem 6.2 can also be expressed as $\gamma = h_{k+1,k}^2\,|c_k|^2$. It is shown in [823] that

$$r_k^O = -h_{k+1,k}[y_k^O]_k v_{k+1} = -\|r_0\| h_{k+1,k} (e_k^T H_k^{-1} e_1) v_{k+1},$$

and

$$e_k^T H_k^{-1} e_1 = -\frac{1}{h_{k+1,k}\vartheta_{1,k+1}}, \quad [y_k^O]_k = -\frac{\|r_0\|}{h_{k+1,k}\vartheta_{1,k+1}},$$

where the $\vartheta_{1,j}$'s are the entries of the first row of U_{k+1}^{-1} in the factorization $H_{k+1} = U_{k+1} C^{(k+1)} U_{k+1}^{-1}$. Remember that

$$U_{k+1} = (\, e_1 \quad H_{k+1} e_1 \quad \cdots \quad H_{k+1}^k \,).$$

The Q-OR residual vector is

$$r_k^O = \frac{\|r_0\|}{\vartheta_{1,k+1}} v_{k+1}.$$

Since v_{k+1} is of unit norm, it yields that

$$|\vartheta_{1,k+1}| = \frac{\|r_0\|}{\|r_k^O\|}.$$

The residual vectors r_k^O of Q-OR and r_k^M of Q-MR can be expressed as polynomials of degree k in A applied to the initial residual r_0,

$$r_k^O = p_k^O(A) r_0, \quad r_k^M = p_k^M(A) r_0.$$

These polynomials have the value 1 at 0 by definition of the residual vector. It can be shown that the residual polynomial p_k^O of the Q-OR method is proportional to the characteristic polynomial of H_k.

Let us now consider the Q-MR residual polynomial p_k^M. If we assume that H_k is nonsingular, we can define

$$\hat{H}_k = H_k + h_{k+1,k}^2 H_k^{-*} e_k e_k^T. \tag{6.19}$$

The second term in the right-hand side is only nonzero in the last column. Hence, the matrix $\hat{H}_k$ is upper Hessenberg and the solution y_k^M of the least squares problem can be obtained by solving $\hat{H}_k y_k^M = \|r_0\| e_1$. It is shown in [823] that the residual polynomial p_k^M of the Q-MR method is proportional to the characteristic polynomial of $\hat{H}_k$.

We have already seen that $\|r_k^O\|/\|r_0\| = 1/|\vartheta_{1,k+1}|$, but we also have a characterization of the quasi-residual norm,

$$\frac{\|z_k^M\|}{\|r_0\|} = \frac{1}{\left[\sum_{j=1}^{k+1} |\vartheta_{1,j}|^2\right]^{\frac{1}{2}}}. \tag{6.20}$$

This can be proved from the peak-plateau relations,

$$\begin{aligned}
\frac{|\vartheta_{1,k+1}|^2}{\|r_0\|^2} &= \frac{1}{\|z_k^M\|^2} - \frac{1}{\|z_{k-1}^M\|^2}, \\
\frac{|\vartheta_{1,k}|^2}{\|r_0\|^2} &= \frac{1}{\|z_{k-1}^M\|^2} - \frac{1}{\|z_{k-2}^M\|^2}, \\
&\vdots = \vdots \\
\frac{|\vartheta_{1,2}|^2}{\|r_0\|^2} &= \frac{1}{\|z_1^M\|^2} - \frac{1}{\|z_0^M\|^2}.
\end{aligned}$$

Summing up all these relations we obtain

$$\frac{1}{\|z_k^M\|^2} - \frac{1}{\|z_0^M\|^2} = \frac{1}{\|r_0\|^2} \left[|\vartheta_{1,k+1}|^2 + \cdots + |\vartheta_{1,2}|^2\right].$$

But noticing that $\|z_0^M\| = \|r_0\|$ and $\vartheta_{1,1} = 1$ gives the result.

If $\vartheta_{1,k+1} = 0$, H_k is singular, and the Q-MR quasi-residual norm stagnates. This result shows that without stagnation the quasi-residual norm is strictly decreasing.

Regarding convergence, it can be shown that any convergence curve is possible for the (relative) residual norms of Q-OR methods and any non-increasing convergence curve is possible

for the (relative) quasi-residual norms of Q-MR methods. Moreover, as long as A does not have to satisfy other constraints (like being symmetric), the eigenvalues of the matrix A can be prescribed for any convergence curve as well as the eigenvalues of the matrices H_k and $\hat{H}_k$ in all iteration numbers k, see [823]. However, a prescribed convergence curve is also linked to a particular right-hand side b.

Many papers have been written providing upper bounds depending on the eigenvalues of A for the residual norms of particular Krylov methods. For simplicity, we consider the case where the Q-MR algorithm does not stop before iteration n.

Theorem 6.5. *Assume the Q-MR algorithm with unit norm basis vectors terminates at iteration n. Let H, such that $AV = VH$, which is assumed to be diagonalizable as $H = X_H \Lambda X_H^{-1}$. Then, the Q-MR residual norm at iteration k is bounded as*

$$\|r_k^M\| \leq \|r_0\| \, \|X_H\| \, \|X_H^{-1}\| \, \sqrt{k+1} \min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)|, \tag{6.21}$$

where π_k is the set of polynomials of degree k.

We observe that, if the matrix V of the basis vectors is unitary, we can replace X_H in the bound (6.21) by the matrix X of the eigenvectors of A. Moreover, if the matrix A is normal then the term $\kappa(X) = \|X\| \, \|X^{-1}\|$ can be replaced by 1. Using Theorem 6.5 and the relations between the Q-MR and Q-OR methods we obtain an upper bound for the norm of the Q-OR residual.

Corollary 6.6. *Using the same notation as in Theorem 6.5, the Q-OR residual norm is bounded as*

$$\|r_k^O\| \leq \frac{\|r_0\|}{|c_k|} \, \|X_H\| \, \|X_H^{-1}\| \, \sqrt{k+1} \min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)|, \tag{6.22}$$

where c_k is the cosine of the kth Givens rotation used to reduce to triangular form the upper Hessenberg matrices.

More importantly, the residual norms can be written as functions of the eigenvalues and eigenvectors.

Theorem 6.7. *Let A be a diagonalizable matrix with a spectral factorization $X\Lambda X^{-1}$ where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ contains the distinct eigenvalues, let b be the right-hand side and $r_0 = b - Ax_0$ such that $c = X^{-1} r_0$ has no zero entries and let $Z = V^{-1}X$ with V the matrix of the basis vectors. Then for $0 < k < n$, let*

$$\mu_1^N = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} Z_{i,j} \, c_j \, \lambda_j \right|^2,$$

and for $k \geq 2$

$$\mu_k^N = \sum_{\mathcal{I}_k} \left| \sum_{\mathcal{J}_k} \det(Z_{\mathcal{I}_k, \mathcal{J}_k}) \, c_{j_1} \cdots c_{j_k} \, \lambda_{j_1} \cdots \lambda_{j_k} \prod_{j_\ell < j_p \in \mathcal{J}_k} (\lambda_{j_p} - \lambda_{j_\ell}) \right|^2,$$

and for $k \geq 1$

$$\mu_{k+1}^D = \sum_{\mathcal{I}_{k+1}} \left| \sum_{\mathcal{J}_{k+1}} \det(Z_{\mathcal{I}_{k+1},\mathcal{J}_{k+1}})\, c_{j_1} \cdots c_{j_{k+1}} \prod_{j_\ell < j_p \in \mathcal{J}_{k+1}} (\lambda_{j_p} - \lambda_{j_\ell}) \right|^2 .$$

The summations are over all sets of indices $\mathcal{I}_{k+1}, \mathcal{J}_{k+1}, \mathcal{I}_k, \mathcal{J}_k$ *defined as* $\mathcal{I}_\ell$ *(or* $\mathcal{J}_\ell$*) to be a set of* ℓ *indices* $(\, i_1, \quad i_2, \quad \ldots, i_\ell\,)$ *such that* $1 \leq i_1 < \cdots < i_\ell \leq n$ *and* $Z_{\mathcal{I}_\ell,\mathcal{J}_\ell}$ *is the submatrix of* Z *whose row and column indices of entries are defined, respectively, by* $\mathcal{I}_\ell$ *and* $\mathcal{J}_\ell$*.*

Then,

$$\|z_k^M\|^2 = \frac{\mu_{k+1}^D}{\mu_k^N}, \qquad \|r_k^O\|^2 = \left(\frac{\mu_k^N}{\mu_{k+1}^D} - \frac{\mu_{k-1}^N}{\mu_k^D} \right)^{-1},$$

where μ_0^N *is defined as* $\mu_0^N = \mu_1^D$*.*

The last theorem shows that the (quasi)-residual norms in Q-OR/Q-MR methods depend upon three types of objects: Eigenvalues, components of the right-hand side in the eigenvector basis and determinants of submatrices of Z, that is, of the eigenvector matrix multiplied with the inverse of the generated basis for the Krylov subspaces. Even if the matrix A is normal, that is, if $X^*X = I$, it does not imply that the determinants in μ_k^N, μ_{k+1}^D can be simplified. However, if V is unitary, and if the matrix A is normal, we have $X^*X = I$ and the sums over $\mathcal{J}_k$ and $\mathcal{J}_{k+1}$ reduce to only one term ($\mathcal{J}_k = \mathcal{I}_k$, respectively, $\mathcal{J}_{k+1} = \mathcal{I}_{k+1}$).

The formulas in Theorem 6.7 are quite complicated expressions but they describe exactly the dependence of (quasi)-residual norms on eigenvalues and eigenvectors. However, we can obtain simpler bounds by using the following result which was proved in [108].

Lemma 6.8. *Let* E *and* F *be two matrices of sizes* $n \times (k+1)$ *and* $n \times n$ *respectively,* $k \leq n-1$*. If the matrix* E *is of full rank*

$$\frac{\sigma_{min}(F)^2}{e_1^T (E^*E)^{-1} e_1} \leq \frac{1}{e_1^T (E^*(F^*F)E)^{-1} e_1} \leq \frac{\sigma_{max}(F)^2}{e_1^T (E^*E)^{-1} e_1}. \tag{6.23}$$

It yields lower and upper bounds for the norm of the quasi-residual of the Q-MR method.

Theorem 6.9. *Let* A *be a diagonalizable matrix,* $A = X\Lambda X^{-1}$*,* $c = X^{-1} r_0$ *and*

$$\mu_k = \frac{\sum_{1 \leq i_1 < \ldots < i_{k+1} \leq n} \left(\prod_{j=1}^{k+1} |c_{i_j}|^2 \right) \prod_{i_1 \leq i_\ell < i_p \leq i_{k+1}} |\lambda_{i_p} - \lambda_{i_\ell}|^2}{\sum_{1 \leq i_1 < \ldots < i_k \leq n} \left(\prod_{j=1}^{k} |c_{i_j}|^2 \, |\lambda_{i_j}|^2 \right) \prod_{i_1 \leq i_\ell < i_p \leq i_k} |\lambda_{i_p} - \lambda_{i_\ell}|^2}.$$

Then,

$$\mu_k \, [\sigma_{min}(V^{-1}X)]^2 \leq \|z_k^M\|^2 \leq \mu_k \, \|V^{-1}X\|^2.$$

6.2 ▪ The Arnoldi process

The basis which is the best conditioned one is an orthonormal basis, since all the singular values of V_k are equal to 1. In theory it can be obtained by computing the QR factorization of the Krylov matrix K_k. However, numerically, we do not want to compute matrices $A^j r_0$ since for large j's

these vectors tend to be in the same direction and K_k may not be of full rank. A more stable method to compute an orthonormal basis is to use a variant of the Gram-Schmidt algorithm. When it is tailored to the columns of the Krylov matrix, it is called the Arnoldi process.

Let $v_1 = r_0/\|r_0\|$. Let us assume that we have already computed orthonormal basis vectors $v_2, \ldots, v_j$ of unit norms. To obtain the next basis vector which has to be orthogonal to all the previous ones and has to belong to $\mathcal{K}_{j+1}(A, r_0)$, we orthogonalize Av_j against the previous basis vectors. Let

$$\tilde{v}_j = Av_j - \sum_{i=1}^{j} (Av_j, v_i)v_i. \tag{6.24}$$

Clearly,

$$(\tilde{v}_j, v_\ell) = (Av_j, v_\ell) - \sum_{i=1}^{j} (Av_j, v_i)(v_i, v_\ell) = 0, \ \ell = 1, \ldots, j,$$

since in the sum only the term with $i = \ell$ is nonzero. The vector $\tilde{v}_j$ is orthogonal to the previous basis vectors. To obtain the next basis vector v_{j+1} we just have to normalize it as $v_{j+1} = \tilde{v}_j/\|\tilde{v}_j\|$. Denoting,

$$h_{i,j} = (Av_j, v_i), \ i = 1, \ldots, j, \quad h_{j+1,j} = \|\tilde{v}_j\|,$$

using $j = 1, \ldots, k$, we have what is known as the Arnoldi relation,

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^T, \tag{6.25}$$

where V_k is an orthonormal matrix with columns v_j, $j = 1, \ldots, k$, and H_k is upper Hessenberg with real positive entries on the first subdiagonal. This can be continued as long as $h_{k+1,k} \neq 0$ that is, when k is less than the grade of r_0 with respect to A. The orthogonality of the basis vectors implies

$$V_k^T A V_k = H_k.$$

We observe that if the matrix A is symmetric, then H_k is also symmetric. A symmetric upper Hessenberg matrix is tridiagonal and the Arnoldi process reduces to the Lanczos algorithm. The symmetry allows to orthogonalize only against the last two vectors. When the matrix is nonsymmetric we need to explicitly orthogonalize against all the previous vectors. Thus, unfortunately, in the Arnoldi process we have to keep these vectors, and the storage grows linearly with k.

The algorithm described above is the classical Gram-Schmidt (CGS) variant of the Arnoldi process. From relation (6.24), in matrix form we have

$$v_{j+1} = \frac{1}{h_{j+1,j}} (I - V_j V_j^T) A v_j.$$

The matrix $I - V_j V_j^T$ represents the projection on the orthogonal complement of the subspace $\mathcal{K}_j(A, r_0)$. In the CGS algorithm all the dot products (Av_j, v_i) are independent of each other and can be computed in parallel. However, for reasons of stability the algorithm which is the most used in practice is the modified Gram-Schmidt (MGS) variant of the Arnoldi process. Mathematically the two algorithms are equivalent, but their behaviors in finite precision arithmetic are different and MGS is more stable than CGS. In MGS, instead of computing all the dot products with Av_j, we compute the dot products with the last known vector which is the result of the subtraction of the components of Av_j. A code implementing the Arnoldi process with MGS is following.

```
function [V,H] = Arnoldi_MGS(A,u,nitmax)
n = size(A,1);
V = zeros(n,nitmax);
H = zeros(nitmax,nitmax);
u = u / norm(u);
V(:,1) = u;
for j = 1:nitmax
 Av = A * V(:,k); % matrix vector product
 w = Av;
 for i=1:j
  vAv = V(:,i)' * w;
  H(i,j) = vAv;
  w = w - vAv * V(:,i);
 end % for i
 nw = norm(w);
 if nw == 0;
  return;
 end; % if
 if j < n
  H(j+1,j) = nw;
  V(:,j+1) = w / nw;  % next basis vector
 end % if j
end % for j
```

In MGS we have

$$v_{j+1} = \frac{1}{h_{j+1,j}}(I - v_j v_j^T)\cdots(I - v_1 v_1^T)Av_j.$$

Another way of computing an orthonormal basis, proposed in [1106, 1107], is to use Householder reflections, which are usually exploited in the QR algorithm; see, for instance, [548]. In fact, one orthogonalizes against each other the vectors $v_1, Av_1, \ldots, Av_j$ where $v_1, v_2, \ldots$ are the consecutively generated basis vectors, similarly in spirit as in the Arnoldi process. But the orthogonalization is based on QR factorization instead of the Gram-Schmidt process. The Householder variant is more costly than CGS or MGS.

6.3 ▪ FOM, GMRES and related methods

FOM (Full Orthogonalization Method) and GMRES (Generalized Minimal RESidual) is a pair of Q-OR/Q-MR methods using an orthonormal basis for the Krylov subspaces constructed with the Arnoldi process. FOM is a Q-OR method for which the residual vectors are orthogonal to each other. It is a true OR (Orthogonal Residual) method. In GMRES, since the basis is orthonormal, the norm of the quasi-residual is equal to the norm of the residual. Therefore GMRES is a true MR (Minimum Residual) method. The minimization of the norm of the residual implies that in GMRES the residual norms are decreasing.

In FOM and GMRES we construct the orthonormal basis vectors v_k as well as the upper Hessenberg matrices H_k and $\underline{H}_k$. The construction of the basis is done by one of the variants of the Arnoldi process. This yields several variants of FOM and GMRES. For instance, we have GMRES-CGS using the classical Gram-Schmidt orthonormalization, GMRES-MGS using the modified Gram-Schmidt algorithm and GMRES-H using Householder reflections to orthogonalize the basis vectors [1106, 1107].

To compute the iterates we need to solve problems involving H_k or $\underline{H}_k$. This could also lead to several variants of the algorithms. Generally, as in [972] and [980], the matrices H_k and $\underline{H}_k$ are transformed to upper triangular form by using Givens rotations. This can be done in place, without any additional storage. In these implementations of the algorithms the computation of the basis vectors and the transformation of the upper Hessenberg matrix H to upper triangular form are intertwined. A code implementing GMRES with MGS is following.

```
function [x,nit,res] = GMRES_MGS(A,b,x0,epss,nitmax)
n = size(A,1);
rhs = zeros(nitmax+1,1);
V = zeros(n,nitmax+1);
H = zeros(nitmax+1,nitmax);
rot = zeros(2, nitmax); % init Givens rotations
res = zeros(1,nitmax);
x = x0;
r = b - A * x;
nb = norm(b);
bet = norm(r);
res(1) = bet;
rhs(1) = bet;
resid = bet;
v = r / bet;
V(:,1) = v;
nit = 0;

while resid >= epss * nb && nit < nitmax
 nit = nit + 1;
 w = A * v;
 for j=1:nit % modified Gram-Schmidt
  vj = V(:,j);
  vw = vj' * w;
  H(j,nit) = vw;
  w = w - vw * vj;
 end % for j
 nw = norm(w);
 v = w / nw;
 H(nit+1,nit) = nw;
 V(:,nit+1) = v; % next basis vector
 nw1 = nw;
 % apply the preceding Givens rotations to the last column
 for kk=1:nit-1
  h1 = H(kk,nit);
  h2 = H(kk+1,nit);
  H(kk+1,nit) = -rot(2,kk) * h1 + rot(1,kk) * h2;
  H(kk,nit) = rot(1,kk) * h1 + conj(rot(2,kk)) * h2;
 end % for kk
 % compute, store and apply a new rotation to zero
 % the last term in nit th column
 nw = H(nit,nit);
 cs = sqrt(abs(nw1)^2 + abs(nw)^2);
```

```
 if abs(nw) < abs(nw1)
  mu = nw / nw1;
  tau = conj(mu) / abs(mu);
 else
  mu = nw1 / nw;
  tau = mu / abs(mu);
 end % if abs
 % store the rotation for the next columns
 rot(1,nit) = abs(nw) / cs; % cosine
 rot(2,nit) = abs(nw1) * tau / cs; % sine
 % modify the diagonal entry and the right-hand side
 H(nit,nit) = rot(1,nit) * nw + conj(rot(2,nit)) * nw1;
 c = rhs(nit);
 rhs(nit) = rot(1,nit) * c;
 rhs(nit+1) = -rot(2,nit) * c;
 resid = abs(rhs(nit+1)); %  estimate of the residual norm
 res(nit+1) = resid;
end  %  while
% computation of the solution
y = triu(H(1:nit,1:nit)) \ rhs(1:nit);
x = x0 + V(:,1:nit) * y;
res = res(1:nit+1);
```

The CGS variant is obtained by changing `w = A * v;` to `Av = A * v; w = Av;` and `vw = vj' * w;` to `vw = vj' * Av;`.

The code for FOM is almost the same as for GMRES except that we do not want to apply the last rotation if we wish to compute the iterate x_k at iteration k and we have to compute the residual norm differently. Hence, the following lines in the GMRES code above,

```
 H(nit,nit) = rot(1,nit) * nw + conj(rot(2,nit)) * nw1;
 c = rhs(nit);
 rhs(nit) = rot(1,nit) * c;
 rhs(nit+1) = -rot(2,nit) * c;
 resid = abs(rhs(nit+1)); %  estimate of the residual norm
 res(nit+1) = resid;
```

have to be changed to

```
 %  estimate of the residual norm
 resid = H(nit+1,nit) * abs(rhs(nit) / H(nit,nit));
 res(nit+1) = resid;
 if nit < nitmax
  H(nit,nit) = rot(1,nit) * nw + conj(rot(2,nit)) * nw1;
  c = rhs(nit);
  rhs(nit) = rot(1,nit) * c;
  rhs(nit+1) = -rot(2,nit) * c;
 end % if nit
```

Since GMRES minimizes the norm of the residual, the residual norms given by FOM are larger than or equal to those given by GMRES. Moreover, if GMRES residual norms exactly stagnate for some iterations, the corresponding matrices H_k are singular and the FOM iterates

are not defined, but this does not prevent to continue iterating. Moreover, exact stagnation almost never happens in finite precision arithmetic.

As we have seen above for general Q-MR methods, the GMRES residual vectors are polynomials in A times the initial residual vector. At iteration k we have $r_k^G = p_k^G(A)r_0$ where p_k^G is a polynomial of degree k whose value is one at the origin. It is the solution of the minimization problem

$$\|r_k^G\| = \min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(A)r_0\|. \tag{6.26}$$

Then, we have the following result which appeared in [415] Theorem 5.4, [403] Theorem 3.3 and [980] Proposition 4. It follows from the results for general Q-MR methods described above.

Theorem 6.10. *Let A be a diagonalizable matrix $A = X\Lambda X^{-1}$ with a spectrum $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$ and π_k be the set of polynomials of degree k. The norm of the residual vector at iteration $k < n$ of GMRES with the initial residual $r_0 = b - Ax_0$ satisfies*

$$\|r_k^G\| \le \|r_0\| \, \|X\| \, \|X^{-1}\| \min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)|. \tag{6.27}$$

If the matrix A is normal we have

$$\|r_k^G\| \le \|r_0\| \min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)|. \tag{6.28}$$

Unfortunately, the solution of the min-max problem is not known explicitly. Several techniques have been considered to obtain upper bounds. One such approach is based on the field of values of A which is defined as

$$\mathcal{F}(A) = \{v^*Av \mid v^*v = 1, \, v \in \mathbb{C}^n\}.$$

The field of values is a convex set containing the spectrum $\sigma(A)$. If the matrix A is normal the field of values is the convex hull of the eigenvalues. If the origin is not in $\mathcal{F}(A)$, and

$$\mu(\mathcal{F}(A)) = \min_{z \in \mathcal{F}(A)} |z|,$$

that is, the distance of $\mathcal{F}(A)$ from the origin, we have

$$\|r_k^G\| \le \|r_0\| \, \left(1 - \mu(\mathcal{F}(A)) \, \mu(\mathcal{F}(A^{-1}))\right)^{k/2}.$$

A refined bound is proved in [396], Theorem 6.1; see also [395].

When the Hermitian part $M = (A + A^*)/2$ of A (whose eigenvalues are real) is positive definite, we have the bound

$$\|r_k^G\| \le \|r_0\| \, \left(1 - \frac{\lambda_{\min}(M)^2}{\lambda_{\max}(A^*A)}\right)^{k/2}. \tag{6.29}$$

which was proved in [415], Theorem 5.4. In this case the GMRES residual norms decrease strictly monotonically without stagnation.

A generalization of the field of values is the polynomial numerical hull introduced in [853], for any k, as

$$\mathcal{H}_k(A) = \{z \in \mathbb{C} \mid \|p(A)\| \ge |p(z)|, \, \forall p \in \pi_k\}.$$

It has been shown that $\mathcal{H}_1(A) = \mathcal{F}(A)$. The polynomial numerical hull provides a lower bound on the quantity involved in the upper bound of $\|r_k^G\|$,

$$\min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{z \in \mathcal{H}_k(A)} |p(z)| \leq \min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(A)\|.$$

The polynomial numerical hull has been studied in [565, 434, 185, 566, 186, 567] for different types of matrices.

Another tool that has been used to try to explain GMRES convergence is the ϵ-pseudospectrum of A which is defined as

$$\Lambda_\epsilon(A) = \{z \in \mathbb{C} \mid \|(zI - A)^{-1}\| \geq \epsilon^{-1}\}.$$

For properties and applications of the ϵ-pseudospectrum, see [1072]. As in [1071], let L_ϵ be the arc length of the boundary Γ_ϵ of $\Lambda_\epsilon(A)$. Then, using the Cauchy integral formula,

$$p(A) = \frac{1}{2\pi\imath} \int_{\Gamma_\epsilon} (zI - A)^{-1} p(z)\, dz,$$

where $\imath$ is the square root of -1 and taking norms, we obtain the bound,

$$\min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(A)\| \leq \frac{L_\epsilon}{2\pi\epsilon} \min_{\substack{p \in \pi_k \\ p(0)=1}} \max_{z \in \Lambda_\epsilon(A)} |p(z)|.$$

The problem is, of course, the choice of the parameter ϵ. It has to be chosen for the multiplicative factor to be of moderate size but small enough for the set Λ_ϵ to be not too large.

There is a discussion of the relative merits and failures of the bound (6.27) based on eigenvalues and those based on the field of values and the pseudospectrum in [422]. Unfortunately, depending on the chosen example, it is not always the same bound which is descriptive of the behavior of the residual norms. Moreover, there are examples for which none of the bounds is satisfactory.

Bounds that depend on the initial residual vector were considered in [1065]. The idea was to keep the residual vector together with the polynomial and to write

$$\|p(A)r_0\| = \|Xp(\Lambda)X^{-1}r_0\| \leq \|X\|\, \|p(\Lambda)X^{-1}r_0\|.$$

Let $c = X^{-1}r_0/\|r_0\|$. Then,

$$\frac{\|r_k^G\|}{\|r_0\|} \leq \|X\| \min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(\Lambda)c\| = \|X\| \min_{\substack{p \in \pi_k \\ p(0)=1}} \left(\sum_{j=1}^{n} |c_j|^2 |p(\lambda_j)|^2 \right)^{\frac{1}{2}}.$$

We observe that $\min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(\Lambda)c\|$ is the residual norm at iteration k when using GMRES with $x_0 = 0$ for solving $\Lambda x = c$.

FOM convergence has been less studied than GMRES convergence. There can be iterations for which H_k is singular and the FOM iterates are said to be infinite. Bounds of the residual norm must include a factor that can be potentially very large, if not infinite. In fact, we can use results in Theorem 6.4, relation (6.16) which yields

$$\|r_k^F\| = \frac{1}{|c_k|} \|r_k^G\|,$$

where c_k is the cosine of the Givens rotation. When GMRES stagnates ($\|r_k^G\| = \|r_{k-1}^G\|$) the sine s_k of the rotation is equal to 1 and $c_k = 0$. In any case we can write

$$|c_k| = \left(1 - \frac{\|r_k^G\|}{\|r_{k-1}^G\|}\right)^{\frac{1}{2}}.$$

Therefore, the residual norms of FOM and GMRES are linked by the relation,

$$\|r_k^F\| = \frac{\|r_{k-1}^G\|}{(\|r_{k-1}^G\|^2 - \|r_k^G\|^2)^{\frac{1}{2}}} \|r_k^G\|.$$

We observe that if $\|r_{k-1}^G\| \gg \|r_k^G\|$ that is, when GMRES converges fast, the multiplying factor is close to 1 and the FOM and GMRES residual norms are almost the same. When GMRES almost stagnates, the FOM residual norms are much larger than those of GMRES. This is the peak-plateau phenomenon that we have already seen for general Q-OR/Q-MR methods.

Unfortunately, the known bounds of the residual norms do not tell us what are the properties of the matrix A determining the convergence of GMRES residual norms. The fact that GMRES convergence can be prescribed as well as the eigenvalues shows that everything can happen for the GMRES residual norms as long as they are decreasing and their behavior cannot be only linked to the eigenvalues of A, at least for non-normal matrices. To prescribe the residual norms in FOM and GMRES we use the general construction in [823].

Theorem 6.11. *Let f_j, $j = 0, \ldots, n-1$, $f_0 = 1$ be n nonzero, finite positive real numbers, λ_j, $j = 1, \ldots, n$ be n nonzero complex numbers and $\theta_j^{(k)}$, $k = 1, \ldots, n-1$ and $j = 1, \ldots, k$ be given nonzero complex numbers and α a positive real number. There exist matrices A and right-hand sides b such that when applying the FOM method for solving $Ax = b$ starting with a given vector x_0, the norms of the residual vectors satisfy*

$$\frac{\|r_k^F\|}{\|r_0\|} = f_k, \ k = 0, \ldots, n-1,$$

the eigenvalues of A are λ_j, $j = 1, \ldots, n$, the eigenvalues of H_k, $k = 1, \ldots, n-1$ (the Ritz values) are $\theta_j^{(k)}$, $j = 1, \ldots, k$ and $\|r_0\| = \alpha$. The matrices are of the form $A = VUCU^{-1}V^*$ and the right-hand sides are of the form $Ax_0 + \alpha V e_1$ for a unitary matrix V, for the companion matrix C corresponding to the eigenvalues λ_j, $j = 1, \ldots, n$ and for an upper triangular matrix U such that the entries $\vartheta_{i,j}$ of its inverse satisfy $\vartheta_{1,1} = 1$ and for $k = 1, \ldots, n-1$,

$$\begin{aligned}
|\vartheta_{1,k+1}| &= 1/f_k, \\
\vartheta_{j,k+1} &= \frac{\alpha_{j-1}^{(k)}}{\alpha_0^{(k)} f_k}, \quad j = 2, \ldots, k, \\
\vartheta_{k+1,k+1} &= \frac{1}{\alpha_0^{(k)} f_k},
\end{aligned}$$

where the coefficients of the monic polynomial with roots $\theta_j^{(k)}$, $j = 1, \ldots, k$ are $\alpha_0^{(k)}, \ldots, \alpha_{k-1}^{(k)}$ (in ascending order of powers).

The converse of this theorem is also true, that is, any pair of matrix and right-hand side generating given prescribed FOM residual norms, Ritz values and spectrum are of the described form.

We also note that the constructed matrix A is nonderogatory since it is similar to a companion matrix. One can also prescribed the harmonic Ritz values instead of the Ritz values, see [823]. To prescribe the residual norms in GMRES (that must be decreasing) we choose a strictly decreasing sequence g_j, $j = 0, \ldots, n-1$. We compute the corresponding prescribed FOM residual norms by using the following result which is following from relation (6.17),

$$\frac{1}{\|r_k^F\|^2} = \frac{1}{\|r_k^G\|^2} - \frac{1}{\|r_{k-1}^G\|^2}. \tag{6.30}$$

Therefore,

$$\frac{1}{f_k^2} = \frac{1}{g_k^2} - \frac{1}{g_{k-1}^2},$$

and we construct the matrix A using these values of f_k. If we do not want to prescribe the Ritz values, the rows from 2 to n of the upper triangular matrix $\hat{U}^{-1}$ can be chosen arbitrarily with nonzero entries on the diagonal. The only freedom in prescribing GMRES residual norms and the Ritz values of all n iterations using the previous construction is in the unitary matrix V which represents a unitary basis for the Krylov subspace $\mathcal{K}_n(A, r_0)$.

To obtain closed-form expressions for the residual norms as functions of the eigenvalues and eigenvectors we specialized what we have seen for general Q-OR/Q-MR methods for the case where V is unitary. The proof use the Cauchy-Binet formula. Remember that $\mathcal{I}_\ell$ (or $\mathcal{J}_\ell$) is defined to be a set of ℓ ordered indices $(\, i_1, \quad i_2, \quad \ldots, \quad i_\ell \,)$ such that $1 \leq i_1 < \cdots < i_\ell \leq n$.

Theorem 6.12. *Let A be a diagonalizable matrix with a spectral factorization $X\Lambda X^{-1}$ where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ contains distinct eigenvalues, let b be the right-hand side and $r_0 = b - Ax_0$ such that $c = X^{-1}r_0$ has no zero components. Then for $k < n$,*

$$\|r_k^G\|^2 = \mu_{k+1}^N / \mu_k^D,$$

where $\mu_1^D = \sum_{i=1}^n \left| \sum_{j=1}^n X_{i,j}\, c_j\, \lambda_j \right|^2$, and for $k \geq 2$

$$\mu_k^D = \sum_{\mathcal{I}_k} \left| \sum_{\mathcal{J}_k} \det(X_{\mathcal{I}_k,\mathcal{J}_k})\, c_{j_1} \cdots c_{j_k}\, \lambda_{j_1} \cdots \lambda_{j_k} \prod_{j_\ell < j_p \in \mathcal{J}_k} (\lambda_{j_p} - \lambda_{j_\ell}) \right|^2,$$

$$\mu_{k+1}^N = \sum_{\mathcal{I}_{k+1}} \left| \sum_{\mathcal{J}_{k+1}} \det(X_{\mathcal{I}_{k+1},\mathcal{J}_{k+1}})\, c_{j_1} \cdots c_{j_{k+1}} \prod_{j_\ell < j_p \in \mathcal{J}_{k+1}} (\lambda_{j_p} - \lambda_{j_\ell}) \right|^2.$$

The summations are over all sets of indices $\mathcal{I}_{k+1}, \mathcal{J}_{k+1}, \mathcal{I}_k, \mathcal{J}_k$ and $X_{\mathcal{I}_\ell,\mathcal{J}_\ell}$ is the submatrix of X whose row and column indices of entries are defined, respectively, by $\mathcal{I}_\ell$ and $\mathcal{J}_\ell$.

In the case of GMRES, the formula for the residual norm simplifies when the matrix A is normal. The notation is the same as in Theorem 6.12.

Theorem 6.13. *Let A be a normal matrix with a spectral factorization $X\Lambda X^*$ with Λ diagonal (with distinct diagonal entries λ_i) and X unitary and let r_0 be the initial residual vector such that $c = X^* r_0$ has no zero components. Applying GMRES to (A, r_0), the residual norms are given by*

$$\|r_1^G\|^2 = \frac{\sum_{\mathcal{I}_2} |c_{i_1}|^2 |c_{i_2}|^2 \prod_{\substack{i_1 \leq i_\ell < i_j \leq i_2 \\ i_\ell,\, i_j \in \mathcal{I}_2}} |\lambda_{i_j} - \lambda_{i_\ell}|^2}{\sum_{i=1}^n |c_i|^2 |\lambda_i|^2}, \tag{6.31}$$

$$\|r_k^G\|^2 = \frac{\sum_{\mathcal{I}_{k+1}} \left[\prod_{j=1}^{k+1} |c_{i_j}|^2\right] \prod_{\substack{i_1 \le i_\ell < i_j \le i_{k+1} \\ i_\ell,\, i_j \in \mathcal{I}_{k+1}}} |\lambda_{i_j} - \lambda_{i_\ell}|^2}{\sum_{\mathcal{I}_k} \left[\prod_{j=1}^{k} |c_{i_j}|^2 |\lambda_{i_j}|^2\right] \prod_{\substack{i_1 \le i_\ell < i_j \le i_k \\ i_\ell,\, i_j \in \mathcal{I}_k}} |\lambda_{i_j} - \lambda_{i_\ell}|^2}. \tag{6.32}$$

In the normal case the residual norms depend on the eigenvectors of A only through the projections of the initial residual on the eigenvectors, whereas in the non-normal case they also depend on determinants of submatrices of X. Hence, generally, the dependence on the eigenvectors is stronger in the non-normal case. Formula (6.32) for the GMRES residual norm with a normal matrix offers an insight to the fact that outlying eigenvalues can often be associated with an initial stage of slow GMRES convergence, see, for instance, the example in [746]. From (6.32) we see that if there is one tight cluster of eigenvalues and, say, m other eigenvalues well separated from this cluster, then after $m+1$ iterations there will be at least one small factor $|\lambda_{i_j} - \lambda_{i_\ell}|^2$ in every summation term of the numerator because $m+2$ eigenvalues are involved and at least one pair of eigenvalues will belong to the cluster. If the weights are of similar size, one can therefore expect acceleration of convergence after the m initial steps. An analogous argument can be used in the presence of multiple clusters.

We can obtain simple bounds of the residual norms by using Theorem 6.9.

Theorem 6.14. *Let A be a diagonalizable matrix, $A = X\Lambda X^{-1}$, with Λ diagonal (with distinct diagonal entries λ_i), the initial residual r_0 such that $c = X^{-1}r_0$ has no zero components and*

$$\mu_k = \frac{\sum_{1 \le i_1 < \cdots < i_{k+1} \le n} \left[\prod_{j=1}^{k+1} |c_{i_j}|^2\right] \prod_{i_1 \le i_\ell < i_p \le i_{k+1}} |\lambda_{i_p} - \lambda_{i_\ell}|^2}{\sum_{1 \le i_1 < \cdots < i_k \le n} \left[\prod_{j=1}^{k} |c_{i_j}|^2 |\lambda_{i_j}|^2\right] \prod_{i_1 \le i_\ell < i_p \le i_k} |\lambda_{i_p} - \lambda_{i_\ell}|^2}.$$

Then,

$$\mu_k \left[\sigma_{\min}(X)\right]^2 \le \|r_k^G\|^2 \le \mu_k \|X\|^2.$$

We observe that μ_k is the same as the result for normal matrices, except for the definition of c. In these bounds the right-hand side is contained in μ_k through $c = X^{-1}r_0$ and there is no need for $\|X^{-1}\|$ as in many other bounds in the literature. If the matrix is "almost" normal in the sense that $\sigma_{\min}(X)$ is not much different from $\|X\|$, then the dependence on the eigenvalues is "almost" the same as for a normal matrix.

The upper bound of Theorem 6.14 is equivalent to the one in [1065] which is

$$\frac{\|r_k^G\|}{\|r_0\|} \le \|X\| \min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(\Lambda)c\|,$$

with $c = X^{-1}r_0$. Since Λ is diagonal and therefore normal, we can apply the result of Theorem 6.13. The value μ_k defined in Theorem 6.14 is an explicit expression for the minimum $\min_{\substack{p \in \pi_k \\ p(0)=1}} \|p(\Lambda)c\|$.

The case of non-diagonalizable matrices is discussed in [823]. Closed-form expressions for the coefficients of the FOM and GMRES residual polynomials are given in [820, 823].

As we have said above, the GMRES residual norms may stagnate. Let us assume for simplicity that GMRES does not stop before iteration n. Partial stagnation for a given matrix A and right-hand side b can be defined as having

$$\|r_k^G\| < \|r_{k-1}^G\|,\ k = 1, \ldots, m, \quad \|r_k^G\| = \|r_{k-1}^G\|,\ k = m+1, \ldots, m+p-1,$$

and

$$\|r_k^G\| < \|r_{k-1}^G\|,\ k = m+p, \ldots, n.$$

Hence the norms of the residual stay the same for p iterations starting from $k = m$. Complete stagnation corresponds to $m = 0$ and $p = n$. Thus

$$\|r_k^G\| = \|r_0\|,\ k = 1, \ldots, n-1, \text{ and } \|r_n^G\| = 0.$$

Complete stagnation of GMRES was studied in [1149, 1150], see also [1014, 1011] where conditions for non-stagnation were studied, and [748] where the worst-case convergence of GMRES for normal matrices was considered. A study of necessary and sufficient conditions for complete stagnation when $n \leq 4$ is given in [818]. A complete characterization of matrices yielding complete or partial stagnation was given in [819]. This is summarized in the following results.

Theorem 6.15. *We have partial stagnation in GMRES for iteration $m \geq 0$ to iteration $m+p-1$ if and only if the matrix A and the right-hand side b can be written as*

$$A = W\mathcal{R}QW^*, \quad b = Ax_0 + WQ^*e_1,$$

where W is unitary, $\mathcal{R}$ is upper triangular, Q is unitary and the columns $j = m+1, \ldots, m+p-1$, of Q are zero except for the subdiagonal entries $(j+1, j)$ which are $\pm e^{-i\phi_i}$, and the rows $i = m+2, \ldots, m+p$ are zero for columns i to n. We have also the same partial stagnation if and only if $A = VQ\mathcal{R}V^*$ and $b = Ax_0 + Ve_1$ with V unitary and Q and $\mathcal{R}$ as before.

Theorem 6.16. *We have complete stagnation in GMRES if and only if the matrix A and the right-hand side b can be written as in one of the four cases:*

$$\begin{aligned}
&(1)\ A = W\mathcal{U}PW^*, \quad b = Ax_0 + We_n,\\
&(2)\ A = WP^T\mathcal{L}W^*, \quad b = Ax_0 + We_n,\\
&(3)\ A = VP\mathcal{U}V^*, \quad b = Ax_0 + Ve_1,\\
&(4)\ A = V\mathcal{L}P^TV^*, \quad b = Ax_0 + Ve_1.
\end{aligned}$$

where W and V are unitary matrices, $\mathcal{U}$ is a nonsingular upper triangular matrix, $\mathcal{L}$ is a nonsingular lower triangular matrix and P is the permutation matrix

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

Corollary 6.17. *We have complete stagnation in GMRES if and only if the matrix A is $A = VBV^*$ with V unitary, the matrix B being one of the four matrices below and the right-hand sides b are chosen properly,*

(1) B is an upper Hessenberg matrix with the last column proportional to e_1 and $b = Ax_0 + Ve_n$,

(2) B is a lower Hessenberg matrix with the last row proportional to e_1^T and $b = Ax_0 + Ve_n$,

(3) B is an upper Hessenberg matrix with the first row proportional to e_n^T and $b = Ax_0 + Ve_1$,

(4) B is a lower Hessenberg matrix with the first column proportional to e_n and $b = Ax_0 + Ve_1$.

Stagnation of the residual norms implies that the residual vectors stay the same. This is also the case for the iterates and the error vectors. In practice, when solving real-life problems, exact stagnation is a rather rare phenomenon. However, what is more likely to happen are phases of quasi-stagnation where the residual norms still decrease but very slowly. This leads to large values of the FOM residual norms for these iterations.

In [817], it is shown how to compute estimates of the ℓ_2 norm of the error during the FOM or GMRES iterations. This can be used to derive robust stopping criteria.

The advantage of GMRES is that it minimizes the residual norm at each iteration. However, because of the long recurrence, the storage grows with the iteration number as well as the number of floating-point operations per iteration. When solving large problems, the storage capacity that is needed may become an issue and the standard GMRES method, which is sometimes referred as "full GMRES", cannot be used. This problem was noticed very early, see [971, 979] where remedies were proposed.

The most used technique is to periodically restart the algorithm. One runs the algorithm for a given number m of iterations and then the algorithm is restarted from the current approximate solution or using more sophisticated techniques. When restarting from the current iterate, these methods are denoted as FOM(m) and GMRES(m) respectively. The set of iterations between two restarts is called a cycle. In this way, only at most m iterations of the Arnoldi process have to be computed successively and the storage is under control, as well as the number of floating-point operations per cycle.

Mathematically, full FOM and GMRES obtain the solution of the linear system in at most n iterations. This is not true for the restarted versions. Restarting usually slows down convergence compared to the full algorithms. In fact, there are cases for which FOM(m) or GMRES(m) does not converge to the solution even though GMRES(m) produces non-increasing residual norms. It is possible that this method stagnates forever. GMRES(m) may produce identical approximations during an entire cycle of m iterations, and consequently all subsequent cycles behave in the same way.

A common misconception about restarted methods is to believe that increasing the restart parameter m could always improve convergence. There exist examples where convergence is faster for a given value m than for other values $m' > m$, see [397, 423], and the numerical experiments of Section 6.8.

Even though in restarted GMRES we throw away the previous basis vectors at the beginning of the second cycle, the second cycle depends strongly on the first cycle. The following theorem proved in [823] shows that stagnation for some of the last iterations of the first cycle implies stagnation for the same number of iterations at the beginning of the second cycle.

Theorem 6.18. *Let A be a nonderogatory matrix such that $A = VHV^*$ with $V^*V = I$ and H an unreduced upper Hessenberg. Let us assume that*

$$\|r^{(1)}_{m-j-1}\| > \|r^{(1)}_{m-j}\| = \ldots = \|r^{(1)}_m\|.$$

Then,

$$\|r^{(2)}_0\| = \|r^{(2)}_1\| = \cdots = \|r^{(2)}_j\|,$$

which means that we have stagnation for the first j iterations of the second cycle.

The result of Theorem 6.18 holds for any two successive cycles of GMRES(m) and stagnation at the end of a cycle implies stagnation at the beginning of the next cycle. The corresponding FOM(m) iterates are not defined, the upper Hessenberg matrices involved being singular. This result shows that it is a bad idea to restart with the current iterate in case of stagnation of GMRES(m) at the end of a cycle.

There are not many theoretical results about GMRES(m) convergence in the literature. We can cite [688, 1009, 397, 497]. Sufficient conditions for convergence are given in [1160, 1161, 1163, 1162, 1164]. But the conditions in these papers are not easy to check a priori.

All the bounds for the full GMRES residual norms described above can be iterated. For instance, from Theorem 6.10 we have, for a diagonalizable matrix A and after n_c cycles,

$$\|r_m^{(n_c)}\| \leq \|r_0\| \left(\|X\| \, \|X^{-1}\| \min_{\substack{p \in \pi_m \\ p(0)=1}} \max_{\lambda \in \sigma(A)} |p(\lambda)| \right)^{n_c}.$$

Of course, this bound is not of any help if the term within parenthesis is larger than 1. If the matrix A is normal, we can replace $\|X\| \, \|X^{-1}\|$ by 1.

For the ith cycle we have the bound $\|r_k^{(i)}\|^2 \leq \mu_k^{(i)} \, \|X\|^2$ with $k = 1, \ldots, m$ and

$$\mu_k^{(i)} = \frac{\sum_{1 \leq i_1 < \cdots < i_{k+1} \leq n} \left[\prod_{j=1}^{k+1} |c_{i_j}|^2 \right] \prod_{i_1 \leq i_\ell < i_p \leq i_{k+1}} |\lambda_{i_p} - \lambda_{i_\ell}|^2}{\sum_{1 \leq i_1 < \cdots < i_k \leq n} \left[\prod_{j=1}^{k} |c_{i_j}|^2 \, |\lambda_{i_j}|^2 \right] \prod_{i_1 \leq i_\ell < i_p \leq i_k} |\lambda_{i_p} - \lambda_{i_\ell}|^2},$$

where $c = X^{-1} r_m^{(i-1)}$ for $i = 1, \ldots$ When A is normal, we have an equality and $c = X^* r_m^{(i-1)}$. We observe that, when we restart from the current approximate solution, the only thing which changes in the upper bound from one cycle to the next is the vector c. Hence, we have a fast convergence only if one cycle decreases the components of $|c|$ efficiently.

In the ith cycle we have $r_k^{(i)} = p_k^{(i)}(A) r_m^{(i-1)}$ with a polynomial of order k. It yields

$$r_k^{(i)} = p_k^{(i)}(A) p_m^{(i-1)}(A) \cdots p_m^{(1)}(A) r_0.$$

The vector $r_k^{(i)}$ is given by a polynomial of degree $(i-1)m + k$ applied to the initial residual vector r_0. However, the residual norm is not minimized over the global Krylov subspace defined by the polynomial.

In recent years some results appeared about constructing linear systems that generate prescribed residual norms for restarted FOM or GMRES. In [1001] it was proved that any residual norms can be generated in the restarted FOM method. It was shown in [1103] that any history of decreasing residual norms at the last iteration of every cycle of GMRES(m) is possible, provided that at the end of every cycle, there is a strict decrease of the residual norm for the mth iterate in comparison with the previous iteration of that cycle. The residual norm history at the last iteration of every cycle is sometimes referred to as cycle convergence. A possible stagnation at the end of one cycle implies stagnation at the beginning of the next cycle, as we have seen above. Thus in that case, the admissible decreasing residual norm history inside the next cycle is not arbitrary anymore.

The most general result was proved in [823]. The GMRES(m) residual norms are prescribed inside cycles instead of cycle convergence assuming that there is no stagnation at the end of any cycle. In fact, [823] constructs linear systems that generate in every cycle fully prescribed Hessenberg matrices (all their entries are prescribed values). This allows to choose the residual norms inside cycles and as a by-product, one can prescribe the Ritz values or the harmonic Ritz values generated in the restart cycles as well.

GMRES(m) is not always very efficient, particularly for small values of m. Hence, a straightforward idea is to change adaptively the restart parameter m to try to improve convergence. This is not easy since we have already said that increasing m does not always give an improved convergence. Papers about this topic are [688, 1036, 1075, 597, 1155, 842, 89, 294].

A technique for restarting GMRES with vectors different from the current iterate is augmentation. In [90], a method named LGMRES (meaning "Loose" GMRES) is proposed, augmenting the Krylov subspace with what is supposed to be an approximation of the error vector, $z^{(i)} = x_m^{(i)} - x_m^{(i-1)}$. Of course, if m is small, this can only be a very crude approximation of the error vector. The rationale for doing this is that if we add the exact error vector to the current approximate solution we obtain the exact solution of the linear system. LGMRES(m, k) augments the standard Krylov subspace with k previous approximations of the error vector,

$$x_m^{(i+1)} = x_m^{(i)} + q_{m-1}^{(i)}(A) r_m^{(i)} + \sum_{j=i-k+1}^{i} \alpha_j^{(i)} z^{(j)},$$

where the polynomial and the coefficients are chosen to minimize the corresponding residual norm.

The idea which is most used is to augment the Krylov subspace with approximate eigenvectors. The development of these methods is based on the belief that, for some problems, eigenvalues of A of small modulus slow down the convergence of GMRES. Unfortunately, we have seen that this is not always true. In GMRES-E [838], k approximate harmonic Ritz vectors are added at the end of the Krylov subspace. The harmonic Ritz vectors y_i are obtained from GMRES iterations by solving

$$(H_j + h_{j+1,j}^2 H_j^{-*} e_j e_j^T) g_i = \zeta_i g_i, \quad y_i = V_j g_i, \ i = 1, \ldots, k,$$

when H_j with $j \geq k$ is nonsingular, which means that GMRES does not stagnate. Generally, the vectors corresponding to the smallest harmonic Ritz values in magnitude are chosen.

Let $s = m + k$, $W_s = (\, V_m \quad y_1 \quad \cdots \quad y_k \,)$ and Q be an $n \times (s+1)$ matrix whose first $m+1$ columns are V_{m+1} and the k last columns are obtained by orthogonalizing the vectors Ay_i against the previous columns of Q. We have the relation

$$AW_s = Q \underline{H}_s,$$

where $\underline{H}_s$ is an $(s+1) \times s$ upper Hessenberg matrix. The approximate solution is $x_0 + W_s y$ where y is obtained by minimizing the residual norm, that is, by solving $\min_y \|Q^* r_0 - \underline{H}_s y\|$.

The implicitly restarted GMRES, GMRES-IR [839], uses also harmonic Ritz vectors. It is mathematically equivalent to GMRES-E if the same approximate eigenvectors are used. In this method ideas from the implicit Arnoldi algorithm (IRA) are used, see [1035, 729, 730]. Compared to GMRES-E the number of matrix-vector products is smaller. GMRES-IR uses the same subspace as GMRES-E.

Another method [840] is known as GMRES with deflated restarting or GMRES-DR, even though it is an augmentation method. It is mathematically equivalent to the methods GMRES-E and GMRES-IR. This method generalizes the thick restarting technique developed in [1125] for symmetric problems. The idea is to add the harmonic Ritz vectors $y_i = V_m g_i$ at the beginning of the subspace for the next cycle. This is cleverly implemented in [840] using only the vectors g_i. These vectors are orthonormalized giving the columns $p_1, \ldots, p_k$ of a matrix $P_{m,k}$. This matrix is extended to the size $(m+1) \times k$ by appending a zero row at the bottom. The least squares residual (of length $m+1$) is orthonormalized against the columns of the extended matrix yielding a vector p_{k+1} and a matrix $P_{m+1,k+1}$ of size $(m+1) \times (k+1)$. The matrices $\underline{H}_k =$

$P_{m+1,k+1}^T \underline{\tilde{H}}_m P_{m,k}$ and $V_{k+1} = \tilde{V}_{m+1} P_{m+1,k+1}$ are the beginning of the upper Hessenberg matrix and of the basis vectors of the new cycle. For stability reasons, the last basis vector v_{k+1} is orthogonalized against the other columns of V_{k+1}. Then, the Arnoldi recurrence is used to compute the other basis vectors and the rest of the upper Hessenberg matrix. At the end of a cycle we obtain a relation

$$AV_m = V_{m+1}\underline{H}_m,$$

where $\underline{H}_m$ is upper Hessenberg except for the (small) leading block of order $k+1$ which is a full matrix with nonzero entries. At the end of the cycle, we have to solve the least squares problem

$$\min_y \|c - \underline{H}_m y\|,$$

for y^M with $c = V_{m+1}^* r_{old}$, r_{old} being the residual vector at the end of the previous cycle and the approximate solution is $x_m = x_{old} + V_{m+1} y^M$, x_{old} being the approximate solution corresponding to r_{old}. This may seem strange since r_{old} normalized is not the first basis vector. But this can be done as long as r_{old} belongs to the subspace generated by the basis vectors because we can write $r_{old} = V_{m+1} c$. Here, since the basis vectors are orthonormal we obtain $c = V_{m+1}^* r_{old}$. In the general case we would have $c = (V_{m+1}^* V_{m+1})^{-1} V_{m+1}^* r_{old}$.

Solving the least squares problem is slightly different from what is usually done in GMRES because of the dense block at the top-left of $\underline{H}_m$. A QR factorization is used to transform this block to triangular form and then Givens rotations are used for the rest of $\underline{H}_m$.

GMRES-DR is simpler and more efficient than GMRES-IR. A variant was proposed in [948]. In [841] it was suggested to use pseudo-eigenvectors to augment the Krylov subspace.

Another possibility to control the storage in FOM and GMRES is to truncate the long recurrences used to compute the basis vectors. The incomplete orthogonalization method (IOM) was introduced in [971]. The basis vectors are computed as

$$\tilde{v} = Av_k - \sum_{i=i_0}^{k} h_{i,k} v_i, \quad h_{k+1,k} = \|\tilde{v}\|, \quad v_{k+1} = \frac{1}{h_{k+1,k}} \tilde{v},$$

and $i_0 = \max(1, k - m + 1)$. The upper bandwidth of the matrix H_k is m. In the original IOM(m) method the linear system $H_k y^{(k)} = \|r_0\| e_1$ is solved using an LU factorization with partial pivoting and $x_k = x_0 + V_k y^{(k)}$. This is a FOM-like Q-OR method since the basis is only locally orthogonal. The algorithm in [972] is based on updating the LU factorization H_k.

A GMRES-like algorithm, named IGMRES, using a truncated Arnoldi basis was proposed in [175]. Quasi-GMRES (QGMRES), described in [982], is just standard GMRES with a truncated Arnoldi recurrence as described above. This is similar to IGMRES. The direct quasi-GMRES (DQGMRES) method, proposed in [982], is a better implementation of the truncated version of GMRES where x_k is updated from x_{k-1} using the fact that the $(k+1) \times k$ upper Hessenberg matrix $\underline{H}_k$ is banded. This matrix is transformed to upper triangular form by applying the product of rotation matrices. Restarted variants of the truncated method are proposed in [1124].

Methods that are mathematically equivalent to FOM or GMRES have been proposed over the years. Some of them are based on the following result.

Theorem 6.19. *The norm of the residual vector r_k is minimized in $x_0 + \mathcal{K}_k(A, r_0)$ if and only if r_k is orthogonal to $A\mathcal{K}_k(A, r_0)$.*

Proof. We can write

$$\min_{x_k \in x_0 + \mathcal{K}_k(A,r_0)} \|r_k\| = \min_{x_k \in x_0 + \mathcal{K}_k(A,r_0)} \|b - Ax_k\|,$$

$$= \min_{y \in \mathcal{K}_k(A,r_0)} \|r_0 - Ay\|,$$
$$= \min_{w \in A\mathcal{K}_k(A,r_0)} \|r_0 - w\|.$$

The solution to the last minimization problem is given by the orthogonal projection of r_0 onto $A\mathcal{K}_k(A, r_0)$. Let P_k be this orthogonal projector. Then, the solution is $w = P_k r_0$ and $r_k = (I - P_k)r_0$. But $I - P_k$ is the orthogonal projector onto the orthogonal complement of $A\mathcal{K}_k(A, r_0)$. Hence, $r_k \perp A\mathcal{K}_k(A, r_0)$. □

Let us assume that we have a basis of $\mathcal{K}_k(A, r_0)$ given by the columns of the matrix V_k with basis vectors of unit norm. Let W_k be an orthonormal matrix obtained by the QR factorization of AV_k,

$$AV_k = W_k T_k, \tag{6.33}$$

where T_k is upper triangular of order k. The columns of W_k give an orthonormal basis of $A\mathcal{K}_k(A, r_0)$. To have residual vectors which are orthogonal to $A\mathcal{K}_k(A, r_0)$ we define $r_k = (I - W_k W_k^*)r_0$. This is obtained by

$$r_j = r_{j-1} - \alpha_j w_j, \quad \alpha_j = (w_j, r_{j-1}), \qquad j \le k. \tag{6.34}$$

Let $R_{k+1} = (\, r_0 \quad \cdots \quad r_k \,)$, D_k be a diagonal matrix with the α_j's as diagonal entries and $L_{k+1,k}$ be a lower bidiagonal matrix with 1's on the diagonal and -1's on the first subdiagonal. Then, we have the matrix relation,

$$R_{k+1} L_{k+1,k} = W_k D_k.$$

As in GMRES the iterates are defined as

$$x_k = x_0 + V_k\, y_k.$$

From relation (6.33) it yields

$$r_k = r_0 - W_k T_k\, y_k.$$

Multiplying by W_k^* and using the orthogonality property of the residual vectors, the vector of coefficients y_k is obtained by solving a triangular linear system,

$$T_k y_k = W_k^* r_0 = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix}.$$

The last equality is obtained because $(w_j, r_0) = (w_j, r_{j-1}) = \alpha_j$, see (6.34).

We have to choose the matrices V_k. The simpler GMRES method (which we abbreviate as SGMRES) proposed in [1108] corresponds to choosing

$$V_k = \left(\tfrac{r_0}{\|r_0\|} \quad W_{k-1} \right). \tag{6.35}$$

If r_0 is not in $A\mathcal{K}_{k-1}(A, r_0)$, the columns of V_k give a basis of $\mathcal{K}_k(A, r_0)$. The condition number of the matrix V_k was computed in [745, 823],

$$\kappa(V_k) = \frac{\|r_0\| + \|\alpha^{(k-1)}\|}{\|r_{k-1}\|},$$

where $\alpha^{(k-1)} = (\, \alpha_1 \quad \cdots \quad \alpha_{k-1} \,)^T$. The SGMRES basis is well conditioned if and only if the residual norms do not decrease too fast. Small residual norms lead to a badly conditioned basis

and to large condition numbers for AV_k and for T_k. This may lead to instability of SGMRES. The interest of SGMRES compared to GMRES is that we do not have to compute the QR factorization of the upper Hessenberg matrix $\underline{H}_k$ using Givens rotations. We just have to compute the upper triangular matrix T_k and to solve a triangular system.

Another choice for V_k was proposed in [683]. The method is called the residual-based SGMRES (RB-SGMRES for short) and

$$V_k = \begin{pmatrix} \frac{r_0}{\|r_0\|} & \cdots & \frac{r_{k-1}}{\|r_{k-1}\|} \end{pmatrix}. \tag{6.36}$$

If r_0 is not in $A\mathcal{K}_{k-1}(A, r_0)$ and the residual norms are strictly decreasing, the columns of V_k are linearly independent. If stagnation occurs, the residual vectors are linearly dependent and the method breaks down. Bounds for the condition number of V_k are given in [823]. RB-SGMRES is a Q-OR method since the basis vectors of $\mathcal{K}(A, r_0)$ are proportional to the residual vectors. However, this method minimizes the residual norms, and the matrix $V_k^* A V_k$ is upper triangular.

An adaptive version of this method was proposed in [682] to cure the problems we may have with SGMRES and RB-SGMRES. Whenever the residual norm nearly stagnates it uses the vector w_{k-1} at iteration k. Otherwise, when there is a sufficient residual norm decrease, it sets the new direction vector equal to the normalized residual vector $r_{k-1}/\|r_{k-1}\|$. This adaptive choice keeps the basis well conditioned.

The Generalized Conjugate Residual method (GCR) was introduced in [415, 403] as a generalization of the Conjugate Residual (CR) method for symmetric matrices. Let $p_0 = r_0 = b - Ax_0$. Then, for $k = 0, \ldots$ the iterates and the residual vectors are constructed as

$$\begin{aligned} \alpha_k &= \frac{(r_k, Ap_k)}{(Ap_k, Ap_k)}, \\ r_{k+1} &= r_k - \alpha_k Ap_k, \\ x_{k+1} &= x_k + \alpha_k p_k. \end{aligned}$$

This choice of α_k minimizes the norm of $\|b - A(x_k + \alpha p_k)\|$. In the steepest descent method, one takes $p_k = r_k$. But, we would like to minimize the residual norm in $\mathcal{K}_{k+1}(A, r_0)$. This is done by constructing the direction vectors p_j to be $A^T A$ (or $A^* A$ in the complex case) orthogonal. Since the vectors Ap_j will give a basis of $A\mathcal{K}_{k+1}(A, r_0)$, we will obtain residual vectors orthogonal to $A\mathcal{K}_{k+1}(A, r_0)$. The vectors p_j are defined by a long recurrence involving all the previous direction vectors,

$$p_{k+1} = r_{k+1} + \sum_{j=0}^{k} \beta_j^{(k)} p_j.$$

The coefficients $\beta_j^{(k)}$ are

$$\beta_j^{(k)} = -\frac{(Ar_{k+1}, Ap_j)}{(Ap_j, Ap_j)}.$$

We observe that to compute the coefficients we need to have Ar_{k+1} and Ap_j for $j = 0, \ldots, k$. To avoid having two matrix-vector products per iteration, the vectors Ap_j are constructed as

$$Ap_{k+1} = Ar_{k+1} + \sum_{j=0}^{k} \beta_j^{(k)} Ap_j.$$

The properties of the residual and direction vectors are given in the following theorem from [403] (Theorem 3.1, p. 348).

Theorem 6.20. *For the GCR method we have the following properties,*

- $(Ap_i, Ap_j) = 0,\ i \neq j,$
- $(r_i, Ap_j) = 0,\ i > j,$
- $(r_i, Ap_i) = (r_i, Ar_i),$
- $(r_i, Ar_j) = 0,\ i > j,$
- $(r_j, Ap_i) = (r_0, Ap_i),\ i \geq j,$
- $\{p_0, \ldots, p_{k-1}\}$ *and* $\{r_0, \ldots, r_{k-1}\}$ *span* $\mathcal{K}_k(A, r_0)$,
- x_{k+1} *minimizes the residual norm over* $x_0 + span(p_0, \ldots, p_k)$.

□

The residual vectors r_k are orthogonal to $A\mathcal{K}_k(A, r_0)$. From Theorem 6.19 this minimizes the norm of r_k over the Krylov subspace. Therefore, GCR is mathematically equivalent to GMRES. GCR is easy to implement as one can see in the following code.

```
function [x,nit,res] = GCR(A,b,x0,epss,nitmax)
n = size(A,1);
nb = norm(b);
x = x0;
r = b - A * x;
P = zeros(n,nitmax+1);
AP = zeros(n,nitmax+1);
res = zeros(1,nitmax+1);
pAAp = zeros(nitmax+1,1);
p = r;
Ap = A * p;
Ar = Ap;
P(:,1) = p;
AP(:,1) = Ap;
pAAp(1) = Ap' * Ap;
resid = norm(r);
res(1) = resid;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 alpha = (r' * Ap) / pAAp(nit);
 r = r - alpha * Ap;
 x = x + alpha * p;
 resid = norm(r);
 res(nit+1) = resid;
 Ar = A * r;
 beta = -(Ar' * AP(:,1:nit))' ./ pAAp(1:nit);
 p = r + P(:,1:nit) * beta;
 Ap = Ar + AP(:,1:nit) * beta;
 P(:,nit+1) = p;
 AP(:,nit+1) = Ap;
```

```
 pAAp(nit+1) = Ap' * Ap;
end  % while
res = res(1:nit+1);
```

GCR is easier to code than GMRES, but contrary to GMRES, GCR can potentially break down if for some index j, $(Ap_j, Ap_j) = 0$.

A method named GCRO, based on GCR with inner GMRES iterations was proposed by E. de Sturler and D.R. Fokkema [326, 324]. Let U_k and C_k be two $n \times k$ matrices such that

$$AU_k = C_k, \quad C_k^T C_k = I,$$

with columns u_j and c_j. Let $x_0 = 0$ and $x_k \in$ range(U_k) such that the norm of the residual is minimized. Then,

$$r_k = b - AU_k y_k = b - C_k y_k,$$

with

$$y_k = \text{argmin}_y \|b - C_k y\| = C_k^T b.$$

The iterates are $x_k = U_k C_k^T b$ and $r_k = (I - C_k C_k^T)b$. But one has to compute u_{k+1} and c_{k+1} for the next iteration. GCRO requires that the inner iteration maintains orthogonality to the columns of C_k. The storage for GCRO increases with each outer iteration. A truncated version named GCROT was derived in [325]. The truncation strategy was based on determining which subspace in range(C_k) is the most important for convergence of the inner GMRES iteration and throwing away the rest of C_k. A simplified strategy was considered by J.E. Hicken and D.W. Zingg [630].

A predecessor of GCR named Orthomin(d) was proposed in [1104]. The recurrence defining the direction vectors is truncated as

$$p_{k+1} = r_{k+1} + \sum_{j=k-d+1}^{k} \beta_j^{(k)} p_j.$$

Thus p_{k+1} is only $A^T A$-orthogonal to the last d vectors p_j. Orthomin is only mathematically equivalent to GCR when we keep all the vectors p_j. This is known as Orthomin(∞). The convergence of Orthomin(d) when the symmetric part of A is positive definite was proved in [415, 403].

Orthodir was introduced in [1146], see also [672, 673]. The method was defined using an auxiliary matrix Z which defined the dot product. A simplified version using $Z = A^T$ was given in [415], see also [1151]. The residual vectors and the approximate solutions are computed as in GCR, and the direction vectors are computed by

$$p_{k+1} = Ap_k + \sum_{j=0}^{k} \beta_j^{(k)} p_j, \quad \beta_j^{(k)} = -\frac{(A^2 p_k, Ap_j)}{(Ap_j, Ap_j)}.$$

The coefficients are chosen to give a set of $A^T A$-orthogonal vectors. As GCR, Orthodir can eventually break down.

Orthores was proposed in [1146], see also [672, 673] as a generalization of the three-term variant of CG. The simplest version of Orthodir is mathematically equivalent to FOM. These three methods, Orthomin, Orthodir, and Orthores are not much used nowadays.

An algorithm called GCG-LS was proposed in [59, 62] with long recurrences for the residuals and short recurrences for the p_k's.

GMRES is almost always used with a preconditioner M. One possibility is to use another iterative method as a preconditioner with a given stopping criterion. This is sometimes called inner-outer iterations. However, in that case the preconditioner M_k is not the same at every iteration and we do not have a Krylov subspace based on $M^{-1}A$. Y. Saad has proposed a modification of GMRES known as Flexible GMRES (FGMRES) to handle that case, see [975]. The algorithm is as follows

Let x_0 be given, $r_0 = b - Ax_0$,

$$v_1 = \frac{r_0}{\|r_0\|}, \quad f = \|r_0\|e_1,$$

for $k = 1, \ldots$

$$M_k z_k = v_k, \quad w = Az_k,$$

for $i = 1, \ldots, k$

$$h_{i,k} = (w, v_i), \quad w = w - h_{i,k}v_i,$$

end i

$$h_{k+1,k} = \|w\|, \quad v_{k+1} = \frac{w}{h_{k+1,k}},$$

Apply the rotations of iterations 1 to $k-1$ to $(h_{1,k} \ldots h_{k+1,k})^T$. Compute the rotation $R_{k+1,k}$ to zero $h_{k+1,k}$, $f = R_{k+1,k}f$, solve the triangular system for y_k. Compute the norm of the residual which is related to the last component of f, if it is small enough, compute $x_k = x_0 + Z_k y_k$, where $Z_k = [z_1 \cdots z_k]$ and stop
end k.

The main difference with GMRES is that we have to store the vectors z_j. In matrix form, we have

$$AZ_k = V_{k+1}\underline{H}_k.$$

Theorem 6.21. *The approximate solution x_k given by FGMRES minimizes the Euclidean norm (ℓ_2-norm) of the residual over $x_0 + span(z_1, \ldots, z_k)$.*

Proof. See Y. Saad [975]. □

It must be noted that convergence results about GMRES do not carry over to FGMRES because the subspace where the solution is sought is not a Krylov subspace.

6.4 ▪ CGNR and CGNE

Another possibility to solve $Ax = b$ when A is nonsymmetric is to symmetrize the problem and to apply the conjugate gradient algorithm. The first method applies CG to the problem

$$A^T Ax = A^T b,$$

when the matrix A is real. This method is known as CGNR.

Theorem 6.22. *At each iteration CGNR minimizes the Euclidean norm (ℓ_2 norm) of the residual in $x_0 + K_k(A^T A, A^T r_0)$.*

Proof. We know that CG minimizes the norm of the error in the norm given by the matrix of the system. But,

$$\|\epsilon_k\|^2_{A^T A} = (A^T A\epsilon_k, \epsilon_k) = (r_k, r_k) = \|r_k\|^2.$$

□

The second method solves

$$AA^T y = b, \quad x = A^T y. \tag{6.37}$$

When using CG on these equations, the method is denoted as CGNE. Solving equation (6.37) was used by E.J. Craig in his Ph.D. thesis [288] in 1954, see also [289] in 1955.

Theorem 6.23. *At each iteration CGNE minimizes the Euclidean norm (ℓ_2 norm) of the error in $x_0 + A^T K_k(AA^T, r_0)$.*

Proof. The iterates x_k are given by $x_k = A^T y_k$ where y_k is the CG iterate with the matrix AA^T. Then, $x - x_k = A^T(y - y_k)$ where $AA^T y = b$ and

$$\|\epsilon_k\|^2 = (x - x_k, x - x_k) = (A^T(y - y_k), A^T(y - y_k)) = \|y - y_k\|^2_{AA^T}.$$

Moreover, $y_k \in y_0 + K_k(AA^T, r_0)$. □

A downside of these approaches is that the condition number is worse than in the original problem.

6.5 ▪ CMRH

This method uses a Hessenberg basis of the Krylov subspace. Let $k < n$, we consider a rectangular LU factorization of the $n \times k$ Krylov matrix K_k of the form $K_k = L_{n,k} R_k$ where $L_{n,k}$ is a lower trapezoidal $n \times k$ matrix (that is, its top $k \times k$ submatrix is lower triangular) and R_k is an upper triangular matrix of order k with ones on the main diagonal. This factorization exists if the leading principal minors (the determinants corresponding to leading principal submatrices) of K_k are nonzero. The following result was proved in [984].

Theorem 6.24. *Let K_k be the Krylov matrix corresponding to $\mathcal{K}_k(A, r_0)$ for $k < n$. Let us assume that the principal minors of K_k are nonzero. Then, the columns ℓ_j of the lower trapezoidal matrix $L_{n,k}$, such that $K_k = L_{n,k} R_k$ with R_k upper triangular and ones on the main diagonal, are given by $\ell_1 = r_0$ and*

$$\ell_j = A\ell_{j-1} - L_{n,j-1} L^L_{j-1,n} A\ell_{j-1}, \; j = 2, \ldots, k, \tag{6.38}$$

where $L^L_{j-1,n} = \begin{pmatrix} [L^{(1)}_{j-1}]^{-1} & 0 \end{pmatrix}$, $L^{(1)}_{j-1}$ being the top $(j-1) \times (j-1)$ submatrix of $L_{n,j-1}$. Moreover, $\ell_1, \ldots, \ell_k$ span the Krylov subspace $\mathcal{K}_k(A, r_0)$ and

$$AL_{n,k-1} = L_{n,k} \underline{\tilde{H}}_{k-1}, \tag{6.39}$$

where $\underline{\tilde{H}}_{k-1}$ is a $k \times (k-1)$ upper Hessenberg matrix with ones on the subdiagonal, the other nonzero entries of the jth column being $L^L_{j,n} A\ell_j$.

□

Two things have to be modified to obtain a practical algorithm. First, the factorization can break down if $[\ell_j]_j = 0$ for some j and if we have a small $[\ell_j]_j$ we may have stability problems. This was solved in [984] by using a pivoting strategy as in Gaussian elimination. Doing this, the matrix $L_{n,k}$ is no longer trapezoidal, but there exists a permutation matrix P such that $PL_{n,k}$ is trapezoidal. Second, we have to normalize the vectors ℓ_j in some way.

We introduce a permutation vector s to represent P. We assume that $s_1, \ldots, s_j$ are known, and we would like to determine s_{j+1}. We compute $w = A\ell_j$, and we subtract multiples of the

vectors $\ell_1, \ldots, \ell_j$ to annihilate the components $s_1, \ldots, s_j$ of w. This yields a new vector w, and we look for the index i giving the maximum of the moduli of the components of w. Then, we set $s_{j+1} = i$ and we normalize w by the maximum. Because of the permutations the subdiagonal entries of H are no longer equal to 1.

We remark that there is no dot product in this algorithm. However, since we have to pivot for stability, we have to look for the maximum of the components of a vector, and this is also a reduction operation.

CMRH (which is an acronym for Changing Minimal Residual method based on the Hessenberg process) uses the Hessenberg basis. GMRES can be interpreted as solving the problem

$$\min_{w \in \mathbb{R}^{k+1}, z \in K_k(A, r_0)} \|w\|, \quad Az = r_0 + V_{k+1} w.$$

CMRH was introduced in [984] by solving

$$\min_{w \in \mathbb{R}^{k+1}, z \in K_k(A, r_0)} \|w\|, \quad Az = r_0 + L_{n,k+1} w,$$

where $L_{n,k+1}$ is the matrix computed by the Hessenberg process. This method produces an upper Hessenberg matrix H_k. Then, the CMRH algorithm is similar to GMRES, the differences being that

$$x_k = x_0 + L_{n,k} y_k,$$

where y_k is the solution minimizing $\| \|r_0\|_\infty e_1 - H_k y\|$. This is very similar to GMRES. However, a CMRH iteration is cheaper than a GMRES iteration. CMRH is a Q-MR method.

A code for CMRH is following. The maximum number of iterations nitmax has to be smaller than or equal to $n-1$. If one reaches this number of iterations without convergence the algorithm has to be restarted.

```
function [x,nit,res] = CMRH(A,b,x0,epss,nitmax)
n = size(A,1);
nitmax = min(nitmax,n);
rhs = zeros(nitmax+1,1);
H = zeros(nitmax+1,nitmax);
rot = zeros(2, nitmax);  % init Givens rotations
res = zeros(1,nitmax+1);
LP = zeros(n,nitmax+1);
LPs = zeros(n,nitmax+1);
s = [1:n]';
x = x0;
r = b - A * x;
nb = norm(b);
i0 = index(r);
LP(:,1) = r / r(i0);
s = swap(s,1,i0);
LPs(:,1) = swap(LP(:,1),1,i0);
bet = r(i0);
resid = norm(r);
res(1) = resid;
rhs(1) = bet;
nit = 0;
```

```
while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 w = A * LP(:,nit);
 ws = w(s);
 wst = LPs(1:nit,1:nit) \ ws(1:nit);
 H(1:nit,nit) = wst;
 ws(1:nit) = 0;
 ws(nit+1:n) = ws(nit+1:n) - sum(LPs(nit+1:n,1:nit) * wst,2);
 sp = invperm(s);
 w = ws(sp);
 if nit+1 < n
  sj = s(nit+1:n);
  i1 = index(w(sj));
  i0 = i1 + nit;
  s = swap(s,nit+1,i0);
  maxw = w(sj(i1));
 else
  maxw = w(n);
 end % if
 H(nit+1,nit) = maxw;
 LP(:,nit+1) = w / maxw;
 LPs(:,nit+1) = ws / maxw;
 LPs = swapLP(LPs,nit+1,i0);
 nw1 = maxw;
 % apply the preceding Givens rotations to the last column
 for kk = 1:nit-1
  h1 = H(kk,nit);
  h2 = H(kk+1,nit);
  H(kk+1,nit) = -rot(2,kk) * h1 + rot(1,kk) * h2;
  H(kk,nit) = rot(1,kk) * h1 + conj(rot(2,kk)) * h2;
 end  % for kk
 % compute, store and apply a new rotation to zero
 % the last term in nit th column
 nw = H(nit,nit);
 [cc,ss] = givens(nw,nw1);
 % store the rotation for the next columns
 rot(1,nit) = cc; % cosine
 rot(2,nit) = ss; % sine
 % modify the diagonal entry and the right-hand side
 H(nit,nit) = rot(1,nit) * nw + conj(rot(2,nit)) * nw1;
 c = rhs(nit);
 rhs(nit) = rot(1,nit) * c;
 rhs(nit+1) = -rot(2,nit) * c;
 resid = abs(rhs(nit+1));  %  estimate of the residual norm
 res(nit+1) = resid;
end  %  while
% computation of the solution
y = triu(H(1:nit,1:nit)) \ rhs(1:nit);
x = x0 + LP(:,1:nit) * y;
res = res(1:nit+1);
```

```
end

function ip = invperm(perm)
%inverse permutation of perm
n = length(perm);
in = [1:n];
ip(perm) = in;
end

function s = swap(s,i,j);
ss = s(i);
s(i) = s(j);
s(j) = ss;
end

function L = swapLP(L,i,j)
ss = L(i,:);
L(i,:) = L(j,:);
L(j,:) = ss;
end

function [i0,val] = index(u);
[y,I] = max(abs(u));
i0 = I(1);
val = u(i0);
end
```

The residual norms are related by the following result which was proved in [986].

Theorem 6.25. *We have the following relations between the residual norms of GMRES and CMRH,*

$$\|r_k^{GMRES}\| \leq \|r_k^{CMRH}\| \leq \kappa(L_{n,k+1}) \, \|r_k^{GMRES}\|. \tag{6.40}$$

□

Theorem 6.25 shows that, as long as the condition number of $L_{n,k+1}$ is not too large, the residual norm of CMRH is not much larger than the residual norm of GMRES at a given iteration. The norm of $L_{n,k+1}$ is bounded by the Frobenius norm. It yields

$$\|L_{n,k+1}\| \leq \left(\frac{k+1}{2}(2n-k) \right)^{\frac{1}{2}}.$$

6.6 ▪ BiCG, BiCGStab and related methods

When the matrix A is nonsymmetric it is in general not possible to construct an orthogonal basis with a three-term recurrence using only matrix-vector products with A, see [437, 435, 436, 747]. However, a three-term recurrence algorithm for nonsymmetric matrices was introduced by C. Lanczos in [721], see also [722]. It uses the transpose matrix A^T (or the conjugate transpose A^* in the complex case) and generates two sets of biorthogonal basis vectors.

Let v_1 and w_1 be two starting vectors such that $(v_1, w_1) = 1$. We set $v_{-1} = w_{-1} = 0$. Then, for $k = 0, 1, \ldots$,

$$\tilde{v}_k = Av_k - \delta_k v_k - \eta_k v_{k-1},$$
$$\tilde{w}_k = A^T w_k - \delta_k w_k - \tilde{\eta}_k w_{k-1}.$$

The coefficient δ_k is computed as

$$\delta_k = (w_k, Av_k).$$

The coefficients η_k and $\tilde{\eta}_k$ are chosen such that

$$\eta_{k+1}\tilde{\eta}_{k+1} = (\tilde{v}_k, \tilde{w}_k).$$

Finally, the new vectors are normalized to obtain the basis vectors,

$$v_{k+1} = \frac{\tilde{v}_k}{\tilde{\eta}_{k+1}}, \qquad w_{k+1} = \frac{\tilde{w}^k}{\eta_{k+1}}.$$

The relations defining the basis vectors can be written in matrix form. Let

$$T_k = \begin{pmatrix} \delta_1 & \eta_2 & & & \\ \tilde{\eta}_2 & \delta_2 & \eta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \tilde{\eta}_{k-1} & \delta_{k-1} & \eta_k \\ & & & \tilde{\eta}_k & \delta_k \end{pmatrix},$$

and

$$V_k = [v_1 \ \cdots v_k], \quad W_k = [w_1 \cdots w_k].$$

Then, we can write

$$AV_k = V_k T_k + \tilde{\eta}_{k+1} v_{k+1} e_k^T,$$
$$A^T W_k = W_k T_k^T + \eta_{k+1} w_{k+1} e_k^T.$$

The first relation can also be written as $AV_k = V_{k+1}\underline{T}_k$, where

$$\underline{T}_k = \begin{pmatrix} T_k \\ \tilde{\eta}_{k+1} e_k^T \end{pmatrix},$$

is a $(k+1) \times k$ matrix, and we have $V_k^T A V_k = T_k$. We have the following properties for this nonsymmetric Lanczos process.

Theorem 6.26. *If the nonsymmetric Lanczos process does not break down, it computes biorthogonal vectors, that is, such that*

$$(w_i, v_j) = 0, \ i \neq j, \quad i, j = 1, \ldots, k$$

The vectors $v_1, \ldots, v_k$ span $K_k(A, v_1)$ and $w_1, \ldots, w_k$ span $K_k(A^T, w_1)$. The two sequences of vectors can be written as

$$v_k = P_k(A)v_1, \quad w_k = P_k(A^T)w_1,$$

where P_k is a polynomial of degree k.

There exist several variants of the nonsymmetric Lanczos process with different scalings for the basis vectors, see M.H. Gutknecht [593].

The previous algorithm breaks down if at some step we have $(\tilde{v}_k, \tilde{w}_k) = 0$. Either,

a) $v_k = 0$ and/or $w_k = 0$. In both cases we have found an invariant subspace. If $v_k = 0$ we can compute the "exact" solution of the linear system. If only $w_k = 0$, almost the only way to deal with this situation is to restart the algorithm with another vector w_1. Usually, using a random initial vector is enough to avoid this kind of breakdown,

b) The more dramatic situation (which is called a "serious breakdown") is when $(\tilde{v}_k, \tilde{w}_k) = 0$ with $\tilde{v}_k$ and $\tilde{w}_k \neq 0$. A way to solve this problem is to use a look-ahead strategy, the solution being to construct also the vectors v_{k+1} and w_{k+1} at step k maintaining biorthogonality in a blockwise sense. If this is not possible, one tries to construct also the vectors v_{k+2} and w_{k+2} and so on. The worst case is when we reach the dimension of the system without being able to return to the normal situation. This is known as an "incurable breakdown".

Look-ahead techniques were studied in [474, 589, 592, 477], as well as [164, 166, 165, 167, 168, 169]. For details, see [823].

The BiConjugate Gradient (BiCG) algorithm can be derived from the nonsymmetric Lanczos process in the same way as CG was derived from the Lanczos process in the symmetric case. To derive algorithms that use short recurrences we have to use the LU factorization of the nonsymmetric tridiagonal matrix T_k (without permutations). Depending on the version of the nonsymmetric Lanczos process we use, we obtain different versions of BiCG, see M.H. Gutknecht [593].

The classical version of BiCG introduced by R. Fletcher [459] using coupled two-term recurrences is the following. A detailed derivation is given in [823].

```
function [x,nit,res] = BiCG(A,b,x0,epss,nitmax)
nb = norm(b);
x = x0;
r = b - A * x;
res = zeros(1,nitmax+1);
resid = norm(r);
res(1) = resid;
rt = r;
p = r;
pt = rt;
At = A';
rrt = rt' * r;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 Ap = A * p;
 Atp = At * pt;
 gamma = rrt / (pt' * Ap);
 x = x + gamma * p;
 r = r - gamma * Ap;
 rt = rt - gamma * Atp;
 resid = norm(r);
 res(nit+1) = resid;
 rrtn = rt' * r;
 mu = rrtn / rrt;
 rrt = rrtn;
 p = r + mu * p;
```

```
 pt = rt + mu * pt;
end % while
```

The interest of this method is that it is very simple to code. Moreover, when A is symmetric, it is the same as the CG method. However, this algorithm breaks down if $(\tilde{r}_k, r_k) = 0$ and/or $(\tilde{p}_k, Ap_k) = 0$.

Theorem 6.27. *If BiCG does not break down we have the following properties for $k \neq j$,*

$$\begin{aligned}(\tilde{r}_k, r_j) &= 0,\\ (\tilde{p}_k, Ap_j) &= 0,\\ (\tilde{r}_k, p_j) &= 0,\\ (r_k, \tilde{p}_j) &= 0.\end{aligned}$$

BiCG was obtained from the Lanczos process but this method can also be derived from scratch from the conditions

$$x_k \in x_0 + \mathcal{K}_k(A, r_0), \quad r_k \perp \mathcal{K}_k(A^T, \tilde{r}_0),$$

$$\tilde{r}_k \in \tilde{r}_0 + A^T\mathcal{K}_k(A^T, \tilde{r}_0), \quad \tilde{r}_k \perp \mathcal{K}(A, r_0).$$

This was done in [1060].

Above we chose $\tilde{r}_0 = r_0$ but this is not mandatory. The initial vector $\tilde{r}_0$ is often called the shadow vector. Its choice for the nonsymmetric Lanczos process was considered in [563, 1059]. In [1059] it is shown that for a given linear system with a given starting vector, one can choose the shadow vector in such a way to have the same residual vector as another Krylov method at one given iteration k. Defining the shadow vector in another way, one can obtain the same residual vectors every 2^ℓ iterations with $\ell = 0, 1, \ldots, k$. However, there is no mathematical study in the literature about the influence of the shadow vector on the BiCG convergence curve. The two most common choices are $\tilde{r}_0 = r_0$ and $\tilde{r}_0$ random. The latter is sometimes used to try to avoid breakdowns.

Contrary to what happens for the symmetric case, there are not many papers proving results about the behavior of nonsymmetric Lanczos methods in finite precision arithmetic. The author of [80] was interested in eigenvalue computations. His analysis was in the same spirit as what C.C. Paige did for the symmetric Lanczos algorithm; see [889] and [816, 825] for a summary of the results. In finite precision arithmetic the biorthogonality among the basis vectors v_j and w_j can be lost. Under the assumption that local biorthogonality is satisfied up to machine precision and that no (near-)breakdown has occurred, it is shown in [80] that the loss of biorthogonality is linked to the convergence of the Lanczos process as it was the case for symmetric problems.

Another analysis of BiCG in finite precision arithmetic was done in [1068]. We have seen that we can derive BiCG from the Lanczos process. In that paper the opposite direction is taken. The authors first derived the terms due to rounding errors in all the steps of BiCG and then eliminate the $\tilde{r}_j$ and $\tilde{p}_j$ to obtain a matrix equation for the computed residual vectors.

The nonsymmetric Lanczos process (there called the two-sided Lanczos tridiagonalization process) was also considered in [896]. In [894] an "augmented" rounding error analysis was done for the symmetric case. This means that the computed tridiagonal matrix comes from an error-free Lanczos process applied to a slightly symmetrically perturbed block diagonal matrix which is not the same at each iteration. At iteration $k + 1$ this matrix is $diag(T_k, A)$ where T_k

is the Lanczos tridiagonal matrix at iteration k. The authors of [896] studied how this can be extended to the nonsymmetric case. They considered perturbed recurrences relations,

$$AV_k + E_k = V_k \underline{T}_k,$$
$$A^T W_k + F_k = W_k \underline{T}_k^T.$$

When there is no perturbation, the basis vectors satisfy $W_k^T V_k = I$. Through a slight modification of the perturbation terms it can be assumed that the diagonal entries of $W_k^T V_k$ are exactly equal to 1. Then, it is proved in [896] Corollary 4.1 that running k steps of the Lanczos process on A in the presence of perturbations is equivalent to running k steps of an exact Lanczos process on a perturbation of the augmented matrix $diag(T_k, A)$. However, contrary to the symmetric case, the perturbation terms cannot be bounded a priori in terms of $\|E_k\|$ and $\|F_k\|$ because the perturbation terms depend on V_k and W_k whose column norms can grow unboundedly. Hence, the nonsymmetric Lanczos process is not numerically stable in the augmented sense that the symmetric Lanczos process was proved to be.

Parallel versions of BiCG can be derived from parallel variants of the nonsymmetric Lanczos algorithm, see [702, 639, 201]. One of these variants is the CA-Lanczos algorithm, a communication-avoiding method. A CA-BiCG algorithm was derived from it in [91]. However, CA-BiCG can be derived from scratch without referring to CA-Lanczos, see [206, 201]. Analyses of CA-BiCG in finite precision arithmetic were done in [203, 201]. It was shown that bounds for the residual norms and the maximum attainable accuracy can be written in the same form as the corresponding bounds for the standard method multiplied by an amplification factor depending on the condition number of the local bases.

Another variant of BiCG was proposed in [179, 180]. The goal was to reduce the data dependencies of standard BiCG by introducing new vectors and to be able to overlap computation and communication. Today this could be called a communication-hiding algorithm. In this variant there are 5 vector updates and 4 independent dot products that can be all computed in parallel instead of 3 in standard BiCG (if we compute the norm of the residual).

An s-step nonsymmetric Lanczos algorithm was presented in [448] from which a BiCG algorithms was derived. It is a synchronization-reducing algorithm which, unfortunately, uses the monomial basis and only small values of $s \leq 5$ can be used. It has only one single global synchronization point per s iterations for which it needs $2s$ dot products.

Methods like BiCG have two shortcomings. First, they can suffer from breakdowns, but this can be solved, in some cases, by using look-ahead techniques or by changing the shadow vector. Second, in each iteration there is a matrix-vector product with A^T (or A^* in the complex case). This can be a problem if the transpose matrix is not easily available. This could be the case, for instance, on parallel computers where the data for the matrix A is distributed in the local memories. This is why some people developed Krylov methods with short recurrences but not needing multiplications with A^T. These methods are based on the useful remark that if we have two polynomials p and q with real coefficients then,

$$(q(A^T)\tilde{r}_0, p(A)r_0) = (\tilde{r}_0, q(A)p(A)r_0). \tag{6.41}$$

They are sometimes called product-type or transpose-free methods.

The Conjugate Gradient Squared (CGS) was proposed in [1030]. It is based on the BiCG method in which we have

$$r_k = \phi_k(A)r_0, \quad p_k = \theta_k(A)r_0,$$
$$\tilde{r}_k = \phi_k(A^T)\tilde{r}_0, \quad \tilde{p}_k = \theta_k(A^T)\tilde{r}_0,$$

where ϕ_k and θ_k are polynomials of degree less than or equal to k. These polynomials satisfy the following recurrences,

$$\phi_{k+1}(\xi) = \phi_k(\xi) - \gamma_k \, \xi \, \theta_k(\xi), \quad \theta_{k+1}(\xi) = \phi_{k+1}(\xi) + \mu_{k+1} \, \theta_k(\xi), \tag{6.42}$$

In BiCG we have to compute expressions like

$$(\tilde{r}_k, r_k) = (\phi_k(A^T)\tilde{r}_0, \phi_k(A)r_0) = (\tilde{r}_0, [\phi_k(A)]^2 r_0).$$

The idea behind CGS (and more generally behind all product-type methods) is to derive recurrence relations for all the polynomial products we need. From relation (6.42),

$$[\phi_k(A)]^2 = [\phi_{k-1}(A)]^2 + \gamma_{k-1}^2 A^2 [\theta_{k-1}(A)]^2 - 2\gamma_{k-1} A \, \phi_{k-1}(A) \, \theta_{k-1}(A). \tag{6.43}$$

We need to find expressions for the last polynomial in the right-hand side. We also have

$$[\theta_k(A)]^2 = [\phi_k(A)]^2 + \mu_k^2 [\theta_{k-1}(A)]^2 + 2\mu_k \phi_k(A) \, \theta_{k-1}(A), \tag{6.44}$$

and we have also to find relations for $\phi_k(A) \, \theta_{k-1}(A)$.

Proposition 6.28. *For $k > 1$,*

$$\begin{aligned}
\phi_{k-1}(A) \, \theta_{k-1}(A) &= [\phi_{k-1}(A)]^2 + \mu_{k-1} \phi_{k-1}(A) \, \theta_{k-2}(A), \\
\phi_{k-1}(A) \, \theta_{k-2}(A) &= \phi_{k-2}(A) \, \theta_{k-2}(A) - \gamma_{k-2} A \, [\theta_{k-2}(A)]^2.
\end{aligned}$$

Proof. For the first relation we multiply

$$\theta_{k-1}(A) = \phi_{k-1}(A) + \mu_{k-1} \, \theta_{k-2}(A),$$

with $\phi_{k-1}(A)$ on the left. For the second relation, we multiply

$$\phi_{k-1}(A) = \phi_{k-2}(A) - \gamma_{k-2} \, A \, \theta_{k-2}(A),$$

by $\theta_{k-2}(A)$ on the right. □

We would like to construct an algorithm such that the residual vector is $r_k = [\phi_k(A)]^2 r_0$. Changing the notation, we introduce new vectors

$$\begin{aligned}
r_k &= [\phi_k(A)]^2 r_0, \\
p_k &= [\theta_{k-1}(A)]^2 r_0, \\
u_k &= \phi_{k-1}(A) \, \theta_{k-1}(A) r_0, \\
q_k &= \phi_k(A) \, \theta_{k-1}(A) r_0.
\end{aligned}$$

We observe that $\phi_k(A)r_0$ is the residual vector we would have obtained from the BiCG algorithm starting from r_0. Then, we apply again the same polynomial but the resulting residual vector is not what we would have obtained after k iterations by running BiCG starting from $\phi_k(A)r_0$.

Proposition 6.29. *The vectors defined above satisfy the following recurrence relations,*

$$\begin{aligned}
r_k &= r_{k-1} - \gamma_{k-1} A(u_k + q_k), \\
p_k &= u_k + \mu_{k-1}[\mu_{k-1} p_{k-1} + q_{k-1}], \\
u_k &= r_{k-1} + \mu_{k-1} q_{k-1}, \\
q_k &= u_k - \gamma_{k-1} A p_k.
\end{aligned}$$

Proof. These vectors satisfy the following recurrence relations,

$$r_k = r_{k-1} + \gamma_{k-1}^2 A^2 p_k - 2\gamma_{k-1} A u_k = r_{k-1} - \gamma_{k-1} A \left[2u_k - \gamma_{k-1} A p_k\right],$$

$$p_k = r_{k-1} + \mu_{k-1}^2 p_{k-1} + 2\mu_{k-1} q_{k-1} = r_{k-1} + \mu_{k-1}[\mu_{k-1} p_{k-1} + 2q_{k-1}],$$

$$u_k = r_{k-1} + \mu_{k-1} q_{k-1},$$

$$q_{k-1} = u_{k-1} - \gamma_{k-2} A p_{k-1}.$$

Then,

$$r_k = r_{k-1} - \gamma_{k-1} A(u_k + q_k),$$

and

$$p_k = u_k + \mu_{k-1}[\mu_{k-1} p_{k-1} + q_{k-1}].$$

□

We also need to compute the BiCG coefficients γ_k and μ_k. Let $r_k^B = \phi_k(A) r_0$, $p_k^B = \theta_k(A) r_0$ (and similar definitions for the vectors with a tilde) be the BiCG vectors. To compute μ_k and the numerator of γ_k we need the dot product of residual vectors,

$$\rho_k = (\tilde{r}_k^B, r_k^B) = (\phi_k(A^T)\tilde{r}_0, \phi_k(A) r_0) = (\tilde{r}_0, [\phi_k(A)]^2 r_0) = (\tilde{r}_0, r_k),$$

and $\mu_k = \rho_k / \rho_{k-1}$. To compute γ_k we also need

$$\begin{aligned}(\tilde{p}_k^B, A p_k^B) &= (\theta_k(A^T)\tilde{r}_0, A\theta_k(A) r_0) = (\tilde{r}_0, \theta_k(A)\, A\, \theta_k(A) r_0) \\ &= (\tilde{r}_0, A\,[\theta_k(A)]^2 r_0) = (\tilde{r}_0, A p_{k+1})\end{aligned}$$

We can easily compute the coefficients provided we store $\tilde{r}_0$. The other BiCG vectors $\tilde{r}_k^B$ and $\tilde{p}_k^B$ are not needed. Since the residual vectors are defined as $r_k = [\phi_k(A)]^2 r_0$, mathematically, CGS has the finite termination property as BiCG.

A code implementing CGS is following.

```
function [x,nit,res] = CGS(A,b,x0,epss,nitmax)
nb = norm(b);
x = x0;
r = b - A * x;
res = zeros(1,nitmax+1);
resid = norm(r);
res(1) = resid;
p = r;
u = r;
r0 = r;
Ap = A * p;
rho0 = transpose(r0) * r;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 gamma = rho0 / (transpose(Ap) * r0);
 q = u - gamma * Ap;
 qu = q + u;
 x = x + gamma * qu;
```

```
 Aqu = A * qu;
 r = r - gamma * Aqu;
 resid = norm(r);
 res(nit+1) = resid;
 rho = transpose(r) * r0;
 mu = rho / rho0;
 rho0 = rho;
 u = r + mu * q;
 p = u + mu * (q + mu * p);
 Ap = A * p;
end % while
res = res(1:nit+1);
```

CGS needs two matrix-vector products per iteration as BiCG does, but the transpose (or conjugate transpose) of A is not required. If BiCG converges well, the convergence of CGS is better. However, it is well known that the residual norms of BiCG can be very oscillatory. In this case the CGS residual norms are also oscillatory and the magnitude of the oscillations is larger since the residual polynomial has been squared. In some problems the magnitude of the oscillations can be so large that this may spoil the approximate solution in finite precision arithmetic. We also note that CGS may have (near-) breakdowns if one or both of the denominators are zero or too small.

In Lanczos-type methods the residual vectors r_k must be orthogonal to the Krylov subspace $\mathcal{K}_k(A^T, \tilde{r}_0)$. In CGS the basis for that subspace is constructed using the residual polynomial for r_k, but we can choose any polynomial we wish for constructing the auxiliary basis as long as the vectors spanning $\mathcal{K}_k(A^T, \tilde{r}_0)$ are linearly independent. We can construct the residual vectors of the method as

$$r_k = q_k(A)\phi_k(A)r_0,$$

as long as $q_k(0)\phi_k(0) = 1$ to be able to obtain x_k from r_k with any polynomial q_k of degree k. The choice which is made in BiCGStab [1086, 1087] is

$$q_k(\xi) = (1 - \omega_k\xi)\,(1 - \omega_{k-1}\xi)\cdots(1 - \omega_1\xi) = (1 - \omega_k\xi)\,q_{k-1}(\xi),$$

where the inverses of the roots ω_i can be computed during the iterations. In BiCGStab, ω_k is chosen to (locally) minimize the norm of r_k. We use as a starting point the BiCG polynomials,

$$\phi_k(\xi) = \phi_{k-1}(\xi) - \gamma_{k-1}\,\xi\,\theta_{k-1}(\xi), \quad \theta_k(\xi) = \phi_k(\xi) + \mu_k\,\theta_{k-1}(\xi).$$

Since $q_k(A) = (1 - \omega_k A)q_{k-1}(A)$, we have

$$\begin{aligned} q_k(A)\phi_k(A) &= (1 - \omega_k A)q_{k-1}(A)\,(\phi_{k-1}(A) - \gamma_{k-1}\,A\theta_{k-1}(A)), \\ &= q_{k-1}(A)\,(\phi_{k-1}(A) - \gamma_{k-1}\,A\theta_{k-1}(A)) \\ &\quad -\omega_k A q_{k-1}(A)\,(\phi_{k-1}(A) - \gamma_{k-1}\,A\theta_{k-1}(A)). \end{aligned}$$

We need a recurrence relation for $q_k(A)\theta_k(A)$. We have

$$\begin{aligned} q_k(A)\theta_k(A) &= q_k(A)(\phi_k(A) + \mu_k\,\theta_{k-1}(A)), \\ &= q_k(A)\phi_k(A) + \mu_k(1 - \omega_k A)q_{k-1}(A)\theta_{k-1}(A), \\ &= q_k(A)\phi_k(A) + \mu_k q_{k-1}(A)\theta_{k-1}(A) - \omega_k\mu_k A q_{k-1}(A)\theta_{k-1}(A). \end{aligned}$$

We set $r_k = q_k(A)\phi_k(A)r_0$ and $p_k = q_k(A)\theta_k(A)r_0$ to obtain

$$p_k = r_k + \mu_k(p_{k-1} - \omega_k A p_{k-1}),$$

$$r_k = r_{k-1} - \gamma_{k-1} A p_{k-1} - \omega_k A(r_{k-1} - \gamma_{k-1} A p_{k-1}).$$

With $s_k = r_{k-1} - \gamma_{k-1} A p_{k-1}$, this is

$$r_k = s_k - \omega_k A s_k. \tag{6.45}$$

Finally, we have to find expressions for the BiCG coefficients γ_{k-1} and μ_k. The two quantities that were involved in the BiCG coefficients are

$$(\phi_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0), \quad (\theta_k(A^T)\tilde{r}_0,\, A\theta_k(A)r_0).$$

In was observed in [1086] that $\phi_k(A)r_0$ is mathematically orthogonal to all vectors $t_{k-1}(A^T)\tilde{r}_0$ where t_{k-1} is any polynomial of degree less than or equal to $k-1$. This is based on global orthogonality properties that may not be valid in finite precision arithmetic. Instead of $\phi_k(A^T)\tilde{r}_0$, it is enough to consider only the term of degree k in $(\phi_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0)$. Using the recurrence relation for ϕ_k, it is equal to

$$(-1)^k \gamma_{k-1} \cdots \gamma_0 (A^T)^k,$$

and

$$\begin{aligned}(\phi_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0) &= (-1)^k \gamma_{k-1} \cdots \gamma_0\, ((A^T)^k \tilde{r}_0,\, \phi_k(A)r_0), \\ &= (-1)^k \gamma_{k-1} \cdots \gamma_0\, (\tilde{r}_0,\, A^k \phi_k(A)r_0).\end{aligned} \tag{6.46}$$

Let us consider

$$(q_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0) = (-1)^k \omega_k \cdots \omega_1\, (\tilde{r}_0,\, A^k \phi_k(A)r_0).$$

Hence,

$$(\phi_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0) = \frac{\gamma_{k-1} \cdots \gamma_0}{\omega_k \cdots \omega_1}\, (q_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0).$$

The coefficient μ_k is

$$\mu_k = \frac{(\phi_k(A^T)\tilde{r}_0,\, \phi_k(A)r_0)}{(\phi_{k-1}(A^T)\tilde{r}_0,\, \phi_{k-1}(A)r_0)} = \frac{\gamma_{k-1}}{\omega_k} \frac{(\tilde{r}_0,\, q_k(A)\phi_k(A)r_0)}{(\tilde{r}_0,\, q_{k-1}(A)\phi_{k-1}(A)r_0)}.$$

To obtain μ_k we just have to compute $(\tilde{r}_0,\, q_k(A)\phi_k(A)r_0) = (\tilde{r}_0,\, r_k)$ at each iteration and

$$\mu_k = \frac{\gamma_{k-1}}{\omega_k} \frac{(\tilde{r}_0,\, r_k)}{(\tilde{r}_0,\, r_{k-1})}.$$

For the other coefficient γ_{k-1}, we remark, using (6.46), that the leading coefficient of θ_k is the same as for ϕ_k. All the products of coefficients cancel and we obtain

$$\gamma_{k-1} = \frac{(\tilde{r}_0,\, A^{k-1}\phi_{k-1}(A)r_0)}{(\tilde{r}_0,\, A^k \theta_{k-1}(A)r_0)} = \frac{(\tilde{r}_0,\, r_{k-1})}{(\tilde{r}_0,\, A p_{k-1})}.$$

The roots of the polynomials q_k are chosen to locally minimize the residual norm. From (6.45), minimizing the norm of the residual as a function of ω yields

$$\omega_k = \frac{(s_k, As_k)}{(As_k, As_k)}. \tag{6.47}$$

A code implementing BiCGStab is following. We chose $\tilde{r}_0 = r_0$, but this is not mandatory. As in CGS there are two matrix-vector products per iteration.

```
function [x,nit,res] = BiCGStab(A,b,x0,epss,nitmax)
nb = norm(b);
x = x0;
r = b - A * x;
res = zeros(1,nitmax+1);
resid = norm(r);
res(1) = resid;
p = r;
r0 = r;
Ap = A * p;
rkrt = r0' * r;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 gamma = rkrt / (r0' * Ap);
 s = r - gamma * Ap;
 As = A * s;
 om = (As' * s) / (As' * As);
 x = x + gamma * p + om * s;
 r = s - om * As;
 resid = norm(r);
 res(nit+1) = resid;
 rkr = r0' * r;
 mu = (rkr / rkrt) * (gamma / om);
 rkrt = rkr;
 p = r + mu * (p - om * Ap);
 Ap = A * p;
end % while
res = res(1:nit+1);
```

The generalization to complex data is done in [591]. BiCGStab is as simple to code as CGS and, in many problems, the residual norms are less oscillatory. As BiCG and CGS, BiCGStab may also suffer from (near-)breakdowns. For recipes to cure breakdowns, see [162].

Motivated by the fact that to have good convergence the residual polynomials must be small at the eigenvalues of A and that the zeros of the polynomials q_k in BiCGStab are real in case of real data (see (6.47)), it was proposed in [590] to modify the method to let the polynomial having complex conjugate roots. This was also supposed to give a better damping effect for the residuals. The resulting method is known as BiCGStab2. To be able to still use real arithmetic the polynomial q_k is not chosen in the same way in the odd and even iterates,

$$
\begin{aligned}
q_{2j+1}(\xi) &= (1 - \omega_j \xi) q_{2j}(\xi), \\
q_{2j+2}(\xi) &= (1 - \nu_j) q_{2j}(\xi) + (\nu_j - \eta_j \xi) q_{2j+1}(\xi).
\end{aligned}
$$

When the data is real, the coefficients are real and q_{2j+2} has real or complex conjugate roots. The parameters are chosen to minimize the residual norms in a one- or two-dimensional subspace.

A reordered BiCGStab algorithm was considered in [713], the aim being to be able to overlap the communications for the dot products and the computations with the preconditioner. However, in finite precision arithmetic, the maximum attainable accuracy can be worse than what is obtained with the standard version of BiCGStab.

Communication-avoiding s-step versions of CGS and BiCGStab were derived in [201]. As with CA-BiCG, the goal was to be able to use a matrix powers kernel and to reduce the number of global synchronization points.

A technique called "pipelining" was proposed for BiCGStab in [279]. The authors introduced auxiliary vectors to decouple matrix-vector products and dot products. The data dependencies due to the dot products are alleviated by deriving recurrences for the vectors involved and by computing these dot products with variables of the previous iterations. However, the number of dot products is larger than with standard BiCGStab (6 instead of 4) as well as the number of vector updates (8 instead of 4) and the resulting algorithm tends to be unstable. It was proposed to improve this by replacing the computed residual by the true residual $b - Ax_k$ every m iterations. This is a very simple strategy that may not work for all problems. A mathematical study of the difference between the computed residual and the true residual is done in [275]. The convergence speeds of parallel variants of BiCGStab are studied in [2].

BiCGStab was further generalized in [1017], see also [1019]. This method, called BiCGStab(ℓ), is doing ℓ BiCG-like steps, and then a residual norm minimization in a subspace of dimension ℓ. The residual vector and the approximate solution are only available at the end of these two steps. Even though the ideas on which the algorithm is based are easy to understand, its derivation in [1017] is quite involved. A code implementing BiCGStab(ℓ) is following.

```
function [x,nit,res] = BiCGStabl(A,b,x0,epss,nitmax,ell,kappa)
if isempty(ell)
 l = 2; % BiCGStab(2)
else
 l = ell;
end % if
if isempty(kappa)
 kappa = 0.7;
end % if
nb = norm(b);
nA = size(A,1);
x = x0;
r = b - A * x;
resid = norm(r);
res = zeros(1,nitmax+1);
res(1) = resid;
rt = r;
u = zeros(nA,1);
gamma = 1;
omega = 1;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 gamma = -omega * gamma;
 y = r;
 for j=1:l
  rho = rt' * y;
  beta = rho / gamma;
  u = r - beta * u;
  y = A * u(:,j);
```

```
  u(:,j+1) = y;
  gamma = rt' * y;
  alpha = rho / gamma;
  x = x + alpha * u(:,1);
  r = r - alpha * u(:,2:j+1);
  y = A * r(:,j);
  r(:,j+1) = y;
 end % for j
 G = r' * r;
 Gamma0 = [1; -G(2:l,2:l) \ G(2:l,1); 0];
 Gamma1 = [0; -G(2:l,2:l) \ G(2:l,l+1); 1];
 NGamma0 = Gamma0' * G * Gamma0;
 NGamma1 = Gamma1' * G * Gamma1;
 omega = Gamma0' * G * Gamma1;
 cosine = abs(omega) / sqrt(NGamma0 * NGamma1);
 omega = omega / NGamma1;
 if cosine < kappa
  omega = (kappa / cosine) * omega;
 end % if
 Gamma = Gamma0 - omega * Gamma1;
 x = x - r * [Gamma(2:l+1); 0];
 r = r * Gamma;
 u = u * Gamma;
 resid = norm(r(:,end));
 res(nit+1) = resid;
end % while
res = res(1:nit+1);
```

Usually BiCGStab(ℓ) is used with small values of ℓ. Note that BiCGStab(2) is not equivalent to BiCGStab2. The relatively good performances and the ease of coding of CGS, BiCGStab and BiCGStab(ℓ) triggered an interest for this type of methods in the 1990s and a plethora of product-type or transpose-free methods appeared. Some of them are described in [823].

The QMR algorithm was proposed in [474], see also [473, 469, 476, 470, 477, 478]. It was designed to have a smoother behavior of the residual norms than with BiCG, but still using only short recurrences. In its basic form it can be seen as using the general Q-MR framework with the biorthogonal basis generated by the nonsymmetric Lanczos process. Instead of minimizing the residual norm as in GMRES, QMR minimizes the quasi-residual norm that is,

$$\min_y \| \, \|r_0\| e_1 - \underline{T}_k y \|. \tag{6.48}$$

The matrix $\underline{T}_k$ is a $(k+1) \times k$ tridiagonal matrix. The least squares problem (6.48) is solved using Givens rotations. Since $\underline{T}_k$ is tridiagonal, the upper triangular matrix that is obtained when applying the Givens rotations has only three nonzero diagonals, the main diagonal and the two diagonals next to it. The iterates are defined as

$$x_k = x_0 + V_k y^{(k)},$$

where $y^{(k)}$ is the solution of the least squares problem. However, the iterates can be computed with short recurrences, see, for instance, [823]. Moreover, coupled two-term recurrences can be used as in the following code.

```
function [x,nit,res] = qmr_2t(A,b,x0,epss,nitmax)
n = size(A,1);
res = zeros(1,nitmax);
v = b;
nb = norm(b);
rho = nb;
v = v / rho;
w = v;
p = zeros(n,1);
q = p;
d = p;
s = q;
At = transpose(A);
r = b - A * x0;
x = x0;
resid = norm(r);
res(1) = resid;
c1 = 1;
epsi = 1;
zeta = 1;
theta = 0;
eta = -1;
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 delta = transpose(w) * v;
 p = v - ((zeta * delta) / epsi) * p;
 q = w - ((rho * delta) / epsi) * q;
 Ap = A * p;
 Atq = At * q;
 epsi = transpose(q) * Ap;
 beta = epsi / delta;
 v = Ap - beta * v;
 w = Atq - beta * w;
 rho1 = rho;
 rho = norm(v);
 zeta = norm(w);
 theta1 = theta;
 theta = rho / (c1 * abs(beta));
 c = 1 / sqrt(1 + theta^2);
 eta = -eta * (rho1 * c^2) / (beta * c1^2);
 c1 = c;
 d = eta * p + (theta1 * c)^2 * d;
 s = eta * Ap + (theta1 * c)^2 * s;
 v = v / rho;
 w = w / zeta;
 x = x + d;
 r = r - s;
 resid = norm(r);
```

```
 res(nit+1) = resid;
end % while
res = res(1:nit+1);
```

Since QMR is based on the nonsymmetric Lanczos process, there may be (near-) breakdowns. In [477] a QMR algorithm with look-ahead using coupled two-term recurrences is described. QMRPACK, a Fortran 77 package implementing QMR with look-ahead, is still available in Netlib (`www.netlib.org`). There are relations between the BiCG and QMR iterates, see [474].

Theorem 6.30. *Let the QMR iterates be written as $x_k^{QMR} = x_{k-1}^{QMR} + \tau_k p_k$ and let s_k and c_k be the sine and cosine of the Givens rotations used to reduce $\underline{T}_k$ to upper triangular form. We have*

$$\|r_k^{QMR}\| \leq \|r_0\|\sqrt{k+1}|s_1 s_2 \cdots s_k|,$$

$$x_k^{BiCG} = x_k^{QMR} + \frac{\tau_k |s_k|^2}{c_k^2} p_k.$$

Therefore, the BiCG iterates can be computed from the QMR ones and vice versa. Usually, QMR has a much smoother residual norm convergence curve than BiCG. Other residual bounds can be obtained from those of the general Q-MR framework.

From the parallel Lanczos algorithm, a parallel QMR algorithm was derived in [178], see also [1136]. An s-step QMR algorithm was devised in [471, 472].

The TFQMR method [468] construct iterates that satisfy a quasi-minimum residual property without using the matrix transpose. However, TFQMR is not mathematically equivalent to the original QMR algorithm in [474]. Let

$$y_m = u_k \text{ if } m = 2k-1, \quad y_m = q_k \text{ if } m = 2k$$

where u_k and q_k are CGS vectors, and

$$w_m = [\phi_k(A)]^2 r_0 \text{ if } m = 2k-1, \quad w_m = \phi_k(A)\phi_{k-1}(A) r_0 \text{ if } m = 2k.$$

Let

$$Y_m = (\, y_1 \quad y_2 \quad \cdots \quad y_m \,), \quad W_{m+1} = (\, w_1 \quad w_2 \quad \cdots \quad w_{m+1} \,).$$

One can check that the two bases are related by $AY_m = W_{m+1}\underline{B}_m$, where $\underline{B}_m$ is a lower bidiagonal $(m+1) \times m$ matrix whose nonzero entries depend on the CGS coefficients μ_j. The vectors $y_1, \ldots, y_m$ span the Krylov subspace $\mathcal{K}_m(A, r_0)$. Writing the iterates as $x_m = x_0 + Y_m z$, we can choose z to minimize the norm of the quasi-residual. It turns out that, since $\underline{B}_m$ has a simple nonzero structure, the solution of the least squares problem can be updated cheaply during the iterations, see [468]. The resulting algorithm only needs two matrix-vector products per iteration.

6.7 ▪ IDR methods

The acronym IDR means "Induced Dimension Reduction". The idea is to generate residual vectors which are in subspaces of decreasing dimensions until obtaining a zero residual vector.

The first IDR algorithm, published in [1115], is based on the so-called IDR theorem. We use the formulation given in [1018].

Lemma 6.31. *Let $\mathcal{G}_0$ and $\mathcal{S}$ be subspaces of $\mathbb{C}^n$, μ_0, μ_1 be complex numbers and $\mathcal{G}_1 = (\mu_0 I - A)(\mathcal{G}_0 \cap \mathcal{S})$. Then,*

1. *If $\mathcal{G}_1 \subset \mathcal{G}_0$, then $(\mu_1 I - A)(\mathcal{G}_1 \cap \mathcal{S}) \subset \mathcal{G}_1$.*
2. *If $\mathcal{G}_1 = \mathcal{G}_0 \neq 0$ then $\mathcal{G}_0 \cap \mathcal{S}$ contains an eigenvector of A.*

□

In most IDR algorithms the subspace $\mathcal{S}$ is defined as the orthogonal complement of the linear subspace spanned by the columns of a given matrix that we denote as $\tilde{R}_0$ which is $n \times s$ where s is a given (small) integer. This is denoted as $\mathcal{S} = \tilde{R}_0^{\perp}$. Then, we can formulate the IDR theorem.

Theorem 6.32. *Let $\mathcal{G}_0 = \mathbb{C}^n$ and μ_j, $j = 1, \ldots$, be a sequence of complex numbers. We define*

$$\mathcal{G}_{j+1} = (\mu_j I - A)(\mathcal{G}_j \cap \tilde{R}_0^{\perp}),\ j = 0, 1, \ldots \tag{6.49}$$

If $\tilde{R}_0^{\perp}$ does not contain an eigenvector of A, then for $j = 0, 1, \ldots$ and unless $\mathcal{G}_j = \{0\}$, $\mathcal{G}_{j+1} \subset \mathcal{G}_j$ and $\dim(\mathcal{G}_{j+1}) < \dim(\mathcal{G}_j)$.

□

The proof uses Lemma 6.31. This theorem also holds if $\mathcal{G}_0 = \mathcal{K}_n(A, v)$ for a given vector v. The first IDR algorithm generates residual vectors which are in subspaces of decreasing dimensions. This is done by enforcing orthogonality against a single given vector p, which means that $s = 1$. It uses a variable splitting $I - \omega_j A$ instead of $\mu_j I - A$. The coefficients ω_j are computed every second iteration to minimize the residual norm, see [1115]. We observe that, even though the IDR theorem is more general, this first algorithm used only one vector p to define $\tilde{R}_0$.

The subspaces defined in Theorem 6.32 are related to Krylov subspaces, see [1015, 1018]. Let p_j be the polynomial of degree j defined as $p_j(\xi) = \prod_{i=0}^{j-1}(\mu_i - \xi)$ where the μ_j's are those of Theorem 6.32. The following proposition was proved in [1018].

Proposition 6.33. *Assume $\tilde{R}_0^{\perp}$ does not contain an eigenvector of A. The subspaces $\mathcal{G}_j$ defined by (6.49) are characterized by*

$$\mathcal{G}_j = \{p_j(A)v \,|\, v \perp \mathcal{K}_j(A^*, \tilde{R}_0)\} = p_j(A)[\mathcal{K}_j(A^*, \tilde{R}_0)]^{\perp}, \tag{6.50}$$

where $\mathcal{K}_j(A^*, \tilde{R}_0)$ is defined as

$$\mathcal{K}_k(\tilde{A}, B) = \text{span}\left\{\sum_{j=0}^{k-1}(\tilde{A})^j B c_j \,|\, c_j \in \mathbb{C}^s\right\} = \sum_{j=1}^{s}\mathcal{K}_k(\tilde{A}, b_j), \quad k \geq 1,$$

where B is an $n \times s$ matrix with $n \geq s \geq 1$ of rank s with columns b_i. Moreover, let

$$\mathcal{K}(A^*, \tilde{R}_0) = \cup_{j=0}^{\infty}\mathcal{K}_j(A^*, \tilde{R}_0).$$

Then the following statements are equivalent:

1. *There is an eigenvector of A that is orthogonal to $\tilde{R}_0$.*
2. *$\mathcal{K}(A^*, \tilde{R}_0) \neq \mathbb{C}^n$.*

3. *There exists* $x \neq 0$ *such that* $\mathcal{K}(A, x) \perp \tilde{R}_0$.

□

Even though the dimension of $\mathcal{G}_j$ depends on the choice of $\tilde{R}_0$, the generic situation is that $\dim(\mathcal{G}_j) = n - \dim(\mathcal{K}_j(A^*, \tilde{R}_0))$. Let us assume that none of the μ_j's is an eigenvalue of A and, following [1015], let $\mathcal{W}_j = [p_j(A)]^{-*}\mathcal{K}_j(A^*, \tilde{R}_0)$. Then,

$$\mathcal{W}_j = \mathcal{K}_j(A^*, [p_j(A)]^{-*}\tilde{R}_0),$$

and let $d_j = \dim(\mathcal{W}_j) = \dim(\mathcal{K}_j(A^*, \tilde{R}_0))$. The subspaces $\mathcal{W}_j$ are nested and

$$d_{j+1} - d_j \leq d_j - d_{j-1} \leq s,$$

see [1015].

The idea behind IDR(s) is to generate residual vectors such that $r_k \in \mathcal{G}_j$ for $j = \lfloor k/(s+1) \rfloor$. The following result was proved in [1015], see also [1018].

Proposition 6.34. *Let* $\mathcal{W}_j = [p_j(A)]^{-*}\mathcal{K}_j(A^*, \tilde{R}_0)$. *A vector* r *belongs to* $\mathcal{G}_j$ *if and only if* $r \perp \mathcal{W}_j$.

□

The IDR subspaces $\mathcal{G}_j$ are polynomial images of the orthogonal complements of subspaces of increasing dimensions. Hence, they are a sequence of nested spaces of shrinking dimension. There exist several variants of IDR(s). The first one was proposed in [1034]. Another variant was proposed later in [1090] that enforces biorthogonality conditions of the intermediate residual vectors and the columns of $\tilde{R}_0$. The residual vectors are computed from the previous residual vectors as $r_{k+1} = (I - \omega_j A)v_k$ with v_k defined as

$$v_k = r_k - \sum_{\ell=1}^{s} \gamma_\ell^{(k)} g_{k-\ell}, \tag{6.51}$$

where the vectors g_i are made orthogonal to columns of $\tilde{R}_0$. This is done using the modified Gram-Schmidt algorithm, meaning that we have to compute dot products. For details on the construction of the vectors, see [823]. In the following code the matrices `U` are used to update the approximate solution and the matrices `G=A U` to update the residual. Note that to avoid additional matrix-vector products these matrices are computed independently. Therefore, if s is large, the relation `G=A U` may not be satisfied up to working precision in finite precision arithmetic. The matrix `P` corresponds to $\tilde{R}_0$. For its columns we chose s random vectors that we orthogonalize. The function `orth` orthogonalizes the columns of its input matrix.

```
function [x,nit,res] = IDR_Bio(A,b,x0,epss,nitmax,s)
nb = norm(b);
nA = size(A,1);
x = x0;
r = b - A * x;
res = zeros(1,nitmax+1);
resid = norm(r);
res(1) = resid;
rng('default');
```

```
P = randn(nA,s); % shadow vectors
P = orth(P);
Pt = P';
om = 1;
G = zeros(nA,s);
U = zeros(nA,s);
M = eye(s);
nit = 0;

while resid > epss * nb && nit < nitmax
 nit = nit + 1;
 Ptr = Pt * r;
 for kk=1:s
  % solve small system and make v orthogonal to P
  c = M(kk:s,kk:s) \ Ptr(kk:s);
  v = r - G(:,kk:s) * c;
  U(:,kk) = U(:,kk:s) * c + om * v;
  G(:,kk) = A * U(:,kk);
  % bi-orthogonalize the new basis vectors
  for j=1:kk-1
   alp = (Pt(j,:) * G(:,kk)) / M(j,j);
   G(:,kk) = G(:,kk) - alp * G(:,j);
   U(:,kk) = U(:,kk) - alp * U(:,j);
  end % for j
  M(kk:s,kk) = Pt(kk:s,:) * G(:,kk);
  % make r orthogonal to p_j, j = 1,...,kk
  beta = Ptr(kk) / M(kk,kk);
  r = r - beta * G(:,kk);
  x = x + beta * U(:,kk);
  if kk < s
   Ptr(kk+1:s) = Ptr(kk+1:s) - beta * M(kk+1:s,kk);
  end % if
 end % for kk
 v = r;
 t = A * v;
 om = omega(t,r);
 r = r - om * t;
 x = x + om * v;
 resid = norm(r);
 res(nit+1) = resid;
end % while
end

function om = omega(t,s)
% om = (s' * t) / (t' * t); % standard formula
angle = 0.7;
ns = norm(s);
nt = norm(t);
ts = t' * s;
rho = abs(ts / (nt * ns));
```

```
om = ts / (nt * nt);
if rho < angle
 om = om * angle / rho;
end % if
end
```

Not much is known about the convergence of IDR when considered as an iterative method; however, see [1032, 823]. Other IDR algorithms are described in [1020, 1089, 940, 1152]. An interesting paper about IDR is [594].

Concerning parallel computing, the IDR(s) algorithm in [1034] has a single global synchronization point per matrix-vector product. The IDR(s)-Bio variant has $s(s+1)/2+2$ global synchronization points per cycle. Another variant named IDR(s)-minsync derived from IDR(s)-Bio was proposed in [271], see also [270]. It has only one global synchronization point per step.

A predecessor of IDR(s) was proposed in [1142]. It was submitted ten years before IDR(s) was proposed. This algorithm called ML(k)BiCGStab by its authors uses k ($k > 1$) left starting vectors or shadow vectors. Unfortunately, this algorithm did not draw too much attention, probably because its derivation was too complicated. A new derivation and a reformulation of the algorithm as well as some simplifications were proposed by one of the authors in [1140]. IDR(s) is related to ML(k)BiCGStab but not completely equivalent, although this depends on the variant of ML(k)BiCGStab that is considered, see [1140].

6.8 ▪ Numerical experiments

Surprisingly, let us start with the symmetric matrix bcsstk01. We use this matrix to show the difference in results with a method like CG using short recurrences and FOM using long recurrences. Mathematically, these two methods are equivalent for a symmetric matrix, but Figure 6.1 shows the difference in convergence. The residual norms are almost the same at the beginning of the iterations, but there is a delay in convergence due to rounding errors for CG and a large difference in the number of operations to reach the maximum attainable accuracy which is smaller for CG.

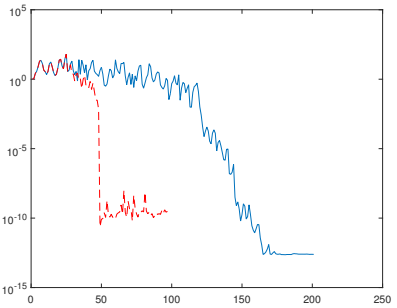


Figure 6.1. *bcsstk01, true residual norms, CG (solid), and FOM (dashed)*

Table 6.1 shows the maximum attainable accuracies for the true residual norms for a set of nonsymmetric matrices. We compare full GMRES and FOM with BiCG, BiCGStab and BiCGStab2. The maximum accuracies are of the same order except for two of the matrices, jpwh_991 for which the short recurrence methods do not converge, and watt1 for which

BiCGStab2 has a breakdown. The maximum accuracies are very large for fs_680_1 because the initial residual norm $8.0425\ 10^{13}$ is very large.

Table 6.1. *Maximum attainable accuracy for the true residual norm*

matrix	GMRES	FOM	BiCG	BiCGStab	BiCGStab2
pde_225	$2.691\ 10^{-14}$	$4.475\ 10^{-14}$	$3.605\ 10^{-14}$	$2.497\ 10^{-14}$	$2.851\ 10^{-14}$
pde_2961	$2.684\ 10^{-13}$	$4.883\ 10^{-13}$	$6.647\ 10^{-11}$	$1.45\ 10^{-12}$	$1.921\ 10^{-12}$
jpwh_991	$1.299\ 10^{-13}$	$2.073\ 10^{-13}$	breakdown	no conv.	NaN
fs_680_1c	$2.325\ 10^{-13}$	$4.425\ 10^{-13}$	$1.439\ 10^{-11}$	$3.978\ 10^{-13}$	$3.605\ 10^{-13}$
fs_680_1	$5.651\ 10^{-1}$	$9.136\ 10^{-1}$	$9.277\ 10^{-1}$	$7.850\ 10^{-2}$	$2.220\ 10^{-1}$
sherman1	$1.804\ 10^{-13}$	$2.575\ 10^{-13}$	$2.387\ 10^{-13}$	$2.417\ 10^{-13}$	$2.771\ 10^{-13}$
nos3	$2.021\ 10^{-11}$	$5.666\ 10^{-11}$	$5.432\ 10^{-12}$	$5.879\ 10^{-12}$	$6.297\ 10^{-12}$
add20	$3.012\ 10^{-23}$	$1.150\ 10^{-22}$	$1.466\ 10^{-23}$	$1.965\ 10^{-23}$	$1.579\ 10^{-23}$
watt1	$2.840\ 10^{-7}$	$1.085\ 10^{-8}$	$3.877\ 10^{-12}$	$1.660\ 10^{-11}$	NaN

In Table 6.2 we display the number of iterations to have $\|r_k\| \leq 10^{-10}\|b\|$, the norm of the residual at convergence, the number of matrix-vector products (mv), and the number of dot products (dp). The number of iterations seems much smaller for BiCGStab2 than for BiCGStab, but this is because one iteration of BiCGStab2 corresponds in fact to two iterations of BiCGStab. The number of matrix-vector products is smaller for GMRES and FOM than for the short recurrence methods, but the number of dot products is much larger because of the long recurrences. For watt1 GMRES converges but the maximum attainable accuracy does not satisfy the stopping criterion.

Figures 6.2-6.4 show the true residual norms as functions of the number of matrix-vector products for three matrices. One can see that BiCGStab converges faster than BiCG. There are less oscillations, but the convergence curve is not smooth. For these examples, BiCGStab2 does not converge faster than BiCGStab. Since GMRES converges fast, there is not much differences between FOM and GMRES.

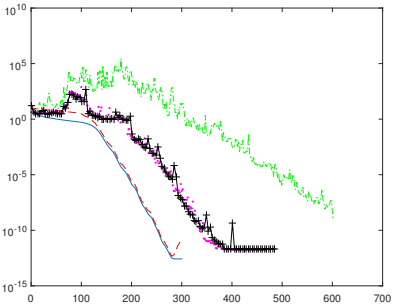


Figure 6.2. *pde_2961, true residual norms as a function of the number of matrix-vector products, GMRES (solid), FOM (dashed), BiCG (dot-dashed), BiCGStab (dotted), and BiCGStab2 (+)*

As we have seen above, to control the storage GMRES is usually used in its restarted form GMRES(m). Table 6.3 shows the number of iterations, the number of matrix-vector products, and the number of dot products for full GMRES and GMRES(m) for $m = 5, 10, 20, 40$ for the same matrices as in Table 6.2. For the matrices not in the table, the restarted versions do not converge in 500 iterations. One can see that, as we said above, increasing m does not always reduce the number of iterations (see pde_225 and pde_2961). Generally, using small values of

Table 6.2. *Number of iterations and absolute residual norms,* $\epsilon = 10^{-10}$

matrix	GMRES	FOM	BiCG	BiCGStab	BiCGStab2
pde_225	75	75	81	53	25
residual	$1.012\ 10^{-9}$	$1.233\ 10^{-9}$	$1.118\ 10^{-9}$	$3.837\ 10^{-10}$	$1.121\ 10^{-9}$
mv	76	76	163	108	101
dp	2927	2927	245	267	252
pde_2961	238	241	300	163	81
residual	$1.534\ 10^{-9}$	$1.488\ 10^{-9}$	$9.904\ 10^{-10}$	$5.404\ 10^{-10}$	$7.345\ 10^{-10}$
mv	239	242	601	328	325
dp	28681	29404	902	817	812
jpwh_991	68	69	breakdown	no conv.	NaN
residual	$1.170\ 10^{-9}$	$1.150\ 10^{-9}$	-	-	-
mv	69	70	-	-	-
dp	2416	2486	-	-	-
fs_680_1c	86	86	97	57	31
residual	$1.058\ 10^{-9}$	$1.143\ 10^{-9}$	$1.587\ 10^{-9}$	$1.723\ 10^{-9}$	$2.266\ 10^{-9}$
mv	87	87	195	116	125
dp	3829	3829	293	287	312
fs_680_1	100	100	407	258	144
residual	$4.213\ 10^{3}$	$4.777\ 10^{3}$	$6.976\ 10^{3}$	$6.851\ 10^{3}$	$7.989\ 10^{3}$
mv	101	101	815	518	577
dp	5152	5152	122	1292	1442
sherman1	388	391	604	479	249
residual	$3.367\ 10^{-10}$	$3.049\ 10^{-10}$	$3.602\ 10^{-10}$	$2.442\ 10^{-10}$	$3.753\ 10^{-10}$
mv	389	392	1209	960	997
dp	75856	77029	1814	2397	2492
nos3	266	267	283	229	113
residual	$2.649\ 10^{-8}$	$2.956\ 10^{-8}$	$2.838\ 10^{-8}$	$2.144\ 10^{-8}$	$3.104\ 10^{-8}$
mv	267	268	567	460	453
dp	35779	36047	851	1147	1132
add20	370	389	428	588	275
residual	$9.401\ 10^{-21}$	$8.041\ 10^{-21}$	$7.349\ 10^{-21}$	$7.755\ 10^{-21}$	$9.698\ 10^{-21}$
mv	371	390	857	1178	1101
dp	69007	76246	1286	2942	2752
watt1	-	-	493	286	-
residual	-	-	$3.131\ 10^{-9}$	$3.622\ 10^{-9}$	-
mv	-	-	987	574	-
dp	-	-	1481	1432	-

m implies having more matrix-vector products because the number of iterations is large and less dot products because the recurrences are not too long.

Figures 6.5-6.6 show the true residual norms as functions of the number of matrix-vector products for different values of m. For fs_680_1c a value of m larger than 30 is necessary to have convergence with the restarted versions. It illustrates the fact that the value of m has to be chosen carefully.

Figure 6.7 shows the residual norms for BiCG and CGS. In the first 100 iterations the CGS residual norms are larger than those of BiCG because there is no convergence. Later, CGS converges faster than BiCG, but the maximal attainable accuracy is much worse.

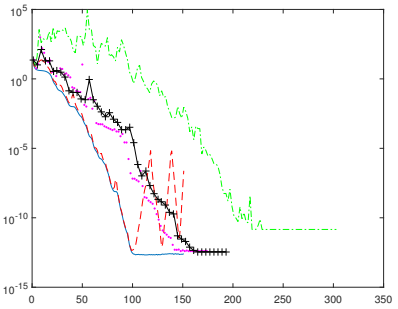


Figure 6.3. *fs_680_1c, true residual norms as a function of the number of matrix-vector products, GMRES (solid), FOM (dashed), BiCG (dot-dashed), BiCGStab (dotted), and BiCGStab2 (+)*

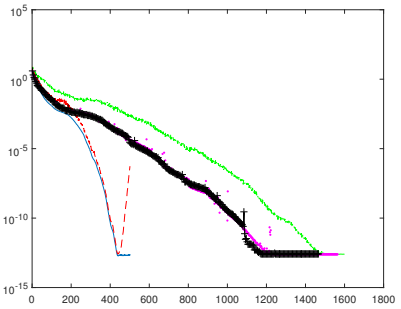


Figure 6.4. *sherman1, true residual norms as a function of the number of matrix-vector products, GMRES (solid), FOM (dashed), BiCG (dot-dashed), BiCGStab (dotted), and BiCGStab2 (+)*

Table 6.3. *GMRES(m), number of iterations,* $\epsilon = 10^{-10}$

matrix	full	5	10	20	40
pde_225	75	169	139	155	111
mv	76	203	153	163	114
dp	2927	762	927	1761	2253
pde_2961	238	814	590	471	501
mv	239	977	649	495	514
dp	28681	3580	3953	5415	10598
jpwh_991	68	212	163	107	77
mv	69	255	180	113	79
dp	2416	931	1023	1197	1604

Figure 6.8 compares the residual norms of full GMRES and CMRH. There is not too much difference betweens the two. Table 6.4 displays the number of iterations to satisfy $\|r_k\| \leq 10^{-10}\|b\|$. The number of iterations for CMRH are only slightly larger than those of GMRES.

Figure 6.9 compares BiCG and the two-term recurrence version of QMR for the matrix pde_2961. The convergence curve of QMR is smooth but it does not converge faster than BiCG.

Table 6.5 displays the results for IDR biorthogonal for different values of the number of shadow vectors, $s = 1, 2, 4, 8$. Increasing s decreases the number of iterations, but the number of

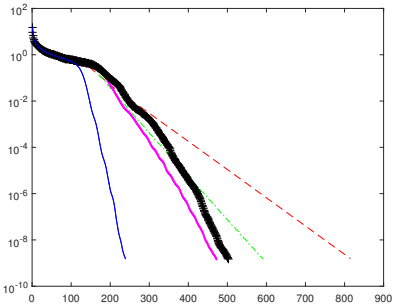


Figure 6.5. *pde_2961, true residual norms as a function of the number of matrix-vector products, GMRES(m), full (solid), $m = 5$ (dashed), $m = 10$ (dot-dashed), $m = 20$ (dotted), and $m = 40$ (+)*

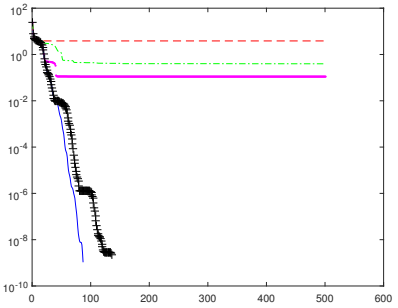


Figure 6.6. *fs_680_1c, true residual norms as a function of the number of matrix-vector products, GMRES(m), full (solid), $m = 5$ (dashed), $m = 10$ (dot-dashed), $m = 20$ (dotted), and $m = 40$ (+)*

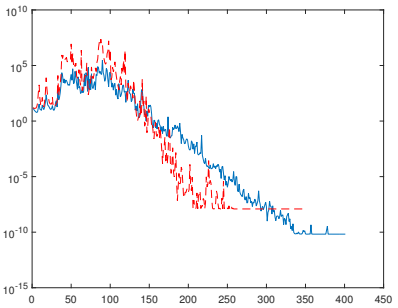


Figure 6.7. *pde_2961, true residual norms as a function of the number of iterations, BiCG (solid) and CGS (dashed)*

matrix-vector products is sometimes only slightly smaller. This is apparently problem dependent.

Figure 6.10 shows the residual norms as functions of the number of matrix-vector products for pde_2961. The convergence is only slightly improved when increasing s, and the maximum attainable accuracy is getting worse.

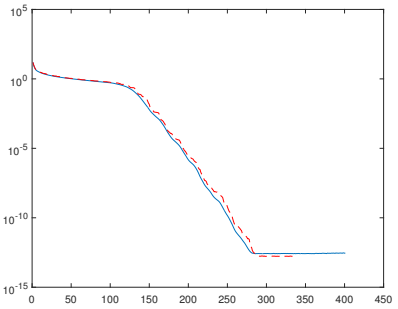


Figure 6.8. *pde_2961, true residual norms as a function of the number of iterations, GMRES (solid) and CMRH (dashed)*

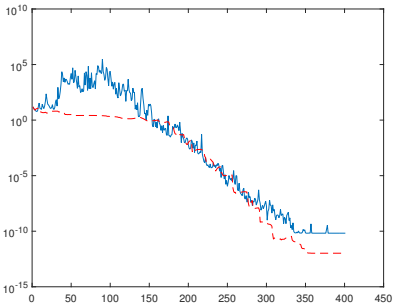


Figure 6.9. *pde_2961, true residual norms as a function of the number of iterations, BiCG (solid) and QMR (dashed)*

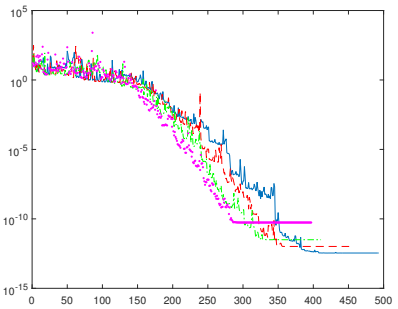


Figure 6.10. *pde_2961, true residual norms as a function of the number of matrix-vector products, IDR bio, $s = 1$ (solid), $s = 2$ (dashed), $s = 4$ (dot-dashed) and $s = 8$ (dotted)*

6.9 ▪ Historical and bibliographical comments

A very nice paper was published in 2001 by M. Eiermann and O. Ernst [396]. Rather than studying the Krylov methods one by one, they show that most methods can be studied in a general abstract framework. In their own words,

Table 6.4. *Number of iterations and absolute residual norms,* $\epsilon = 10^{-10}$

matrix	GMRES	CMRH
pde_225	75	76
residual	$1.012\ 10^{-9}$	$1.289\ 10^{-9}$
mv	76	77
dp	2927	1
pde_2961	238	245
residual	$1.534\ 10^{-9}$	$1.061\ 10^{-8}$
mv	239	246
dp	28681	1
jpwh_991	68	72
residual	$1.170\ 10^{-9}$	1.00610^{-9}
mv	69	73
dp	2416	1
fs_680_1c	86	87
residual	$1.058\ 10^{-9}$	$1.536\ 10^{-9}$
mv	87	88
dp	3829	1
fs_680_1	100	101
residual	$4.213\ 10^{3}$	$3.881\ 10^{3}$
mv	101	102
dp	5152	1
sherman1	388	395
residual	$3.367\ 10^{-10}$	$3.579\ 10^{-10}$
mv	389	396
dp	75856	1
nos3	266	271
residual	$2.649\ 10^{-8}$	$1.925\ 10^{-8}$
mv	267	272
dp	35779	1
add20	370	388
residual	$9.401\ 10^{-21}$	$9.915\ 10^{-21}$
mv	371	389
dp	69007	1
watt1	-	-
residual	-	-
mv	-	-
dp	-	-

We show that essentially any Krylov subspace method for solving a linear system can be classified as either an MR or OR method by appropriate choice of the inner product.

This motivated our interest in the Q-OR/Q-MR framework. The relation between the entries of the first row of the inverse of U_k and the Q-OR and Q-MR residual norms was proved by J. Duintjer Tebbens and G.M. [387] in 2016 following some previous results for FOM and GMRES.

The methods for computing an orthonormal basis of a Krylov subspace originated in the techniques used to orthogonalize a given set of vectors or functions. It goes back to the 19th cen-

Table 6.5. *IDR bio, number of iterations and absolute residual norms,* $\epsilon = 10^{-10}$

matrix	1	2	4	8
pde_225	51	32	18	10
residual	$5.717\ 10^{-10}$	$9.280\ 10^{-10}$	$1.095\ 10^{-9}$	$1.730\ 10^{-10}$
mv	103	97	91	91
pde_2961	173	104	56	30
residual	$3.422\ 10^{-11}$	$6.307\ 10^{-10}$	$1.203\ 10^{-9}$	$7.397\ 10^{-10}$
mv	347	313	281	271
jpwh_991	41	26	16	9
residual	$1.123\ 10^{-9}$	$1.061\ 10^{-10}$	$4.092\ 10^{-10}$	$1.338\ 10^{-10}$
mv	83	79	81	82
fs_680_1c	63	39	22	11
residual	$9.509\ 10^{-10}$	$1.561\ 10^{-9}$	$9.489\ 10^{-12}$	$1.169\ 10^{-10}$
mv	127	118	111	100
fs_680_1	351	165	60	25
residual	$7.540\ 10^{3}$	$7.594\ 10^{3}$	$7.791\ 10^{3}$	$5.371\ 10^{3}$
mv	703	496	301	226
sherman1	793	406	198	75
residual	$2.497\ 10^{-10}$	$1.841\ 10^{-10}$	$4.154\ 10^{-10}$	$4.651\ 10^{-10}$
mv	1587	1219	991	676
nos3	237	129	66	35
residual	$2.751\ 10^{-8}$	$2.923\ 10^{-8}$	$1.072\ 10^{-8}$	$5.878\ 10^{-9}$
mv	475	388	331	316
add20	979	571	158	59
residual	$4.671\ 10^{-21}$	$9.764\ 10^{-21}$	$7.046\ 10^{-21}$	$5.049\ 10^{-20}$
mv	1959	1714	791	532
watt1	410	289	184	-
residual	$1.198\ 10^{-8}$	$5.296\ 10^{-6}$	$2.369\ 10^{-7}$	-
mv	821	868	921	-

tury. But, the main origin is in the works of J.P. Gram, a Danish actuary and mathematician and E. Schmidt, a German mathematician. Gram published a paper in 1883 where he orthogonalized a set of linearly independent vectors. Schmidt orthogonalized a set of functions with respect to a dot product defined by a Riemann integral. He used what is now known as the classical Gram-Schmidt process in 1905 and 1907.

The paper [43] describing the Arnoldi process was received in May 1950. W.E. Arnoldi was an American engineer working for Hamilton Standard Propellers in Wethersfield (Connecticut). There were not many references to Arnoldi's paper in the 1960s and 70s, a few tenths, even though it was cited and commented in J.H. Wilkinson's book [1120] in 1965, pages 382-385.

An early paper on iterative minimization techniques for general matrices is by I.M. Khabaza [594] in 1963. Essentially the same method (without a reference to Khabaza) was proposed by G.I. Marchuk and Y.A. Kuznetsov [784] in 1968, see also [785]. The formulation for the residual was slightly more general. Their algorithm is essentially restarted GMRES with a different orthogonal basis. The differences are that the basis vectors are not of unit norm and the Gram-Schmidt algorithm is used to orthogonalize $A^i v_1$ and not Av_i.

In 1980, Y. Saad [970] used the Arnoldi process for computing approximations of eigenvalues and proved some convergence results. He introduced the Incomplete Orthogonalization

Method (IOM) by truncating the Arnoldi relation. In 1981, Saad [971] introduced FOM (Full Orthogonalization Method) under the name "The method of Arnoldi" to solve linear systems. He considered again the IOM method and proved some convergence results using ellipses enclosing the eigenvalues. The name FOM was first used in [972].

GMRES was presented in the report [979] in May 1985 by Y. Saad and M.H. Schultz. However, the original version of the report was from August 1983. They used the Arnoldi process and Givens rotations to solve the least squares problems, a technique which is now considered as the standard implementation of GMRES. The paper [980] was received originally by the editors in November 1983, in revised form in May 1985 and published in July 1986.

P.N. Brown [174] in 1991 established the relation between FOM and GMRES residual norms and the fact that when GMRES stagnates the upper Hessenberg matrices H_k are singular. This phenomenon is called the peak-plateau behavior. It has also been studied in a more general setting by J.K. Cullum [295, 296] in 1995 and J.K. Cullum and A. Greenbaum [297] in 1996.

It was known at the beginning of the 1990s that eigenvalues alone cannot explain GMRES convergence for non-normal matrices. This was first clearly shown in the 1994 paper [572] by A. Greenbaum and Z. Strakoš. They studied the matrices B that generate the same Krylov residual space as the one given by the pair (A, b). Moreover, it was shown that the spectrum of B can consist of arbitrary nonzero values. In [569] with V. Pták, this was extended in 1996 by proving that any non-increasing sequence of residual norms can be generated by GMRES. The paper [41] by M. Arioli, V. Pták and Z. Strakoš in 1998 closed this series of papers with a full parametrization of the class of matrices and right-hand sides giving prescribed convergence history while the system matrix has prescribed nonzero spectrum. In [385] by J. Duintjer Tebbens and G.M. in 2012 a parametrization was given of the class of matrices and right-hand sides generating, in addition to prescribed residual norms and eigenvalues, prescribed Ritz values in all iterations. In the paper [366] with K. Du in 2017 it was shown that, instead of the Ritz values, one can prescribe the harmonic Ritz values. Prescribing the behavior of early terminating GMRES was studied in the paper [386] in 2014. A study of the role eigenvalues play in forming GMRES residual norms with non-normal matrices was done in 2015 [822].

Complete stagnation of GMRES was studied by I. Zavorin, D.P. O'Leary and H.C. Elman [1150] in 2003, see also Zavorin's thesis [1149] in 2001. In 2008, V. Simoncini and D.B. Szyld [1014] gave conditions involving the symmetric (or Hermitian) part and the skew-symmetric parts of A implying that GMRES is not stagnating. These results were extended by V. Simoncini [1011] in 2010. Necessary and sufficient conditions for GMRES complete and partial stagnation were given in the paper [819] in 2014.

Exact expressions for the residual norms were found starting in 2000. I.C.F. Ipsen gave in 2000 an exact expression for the GMRES residual norm with normal matrices using a minimization problem over $k + 1$ distinct eigenvalues [665]. H. Sadok [985] in 2005 established expressions for the residual norms involving the singular values of the matrices H_k and $\underline{H}_k$. J. Duintjer Tebbens, G.M., H. Sadok and Z. Strakoš [388] gave in 2014 exact expressions for the GMRES residual norms for normal matrices. G.M. and J. Duintjer Tebbens [822] in 2015 extended these results to diagonalizable matrices.

The backward stability of GMRES-MGS was finally proved by C.C. Paige, M. Rozložník and Z. Strakoš [897] in 2006.

Analyses of augmented Krylov methods were published by Y. Saad [977] and A. Chapman and Y. Saad [234] in 1997. This was also considered by M. Eiermann, O. Ernst and O. Schneider [397] in 2000, and V. Simoncini and D.B. Szyld [1013] in 2007. Augmented GMRES methods were considered by J. Baglama and L. Reichel [158] in 1998.

An augmented method named LGMRES was proposed by A.H. Baker, E.R. Jessup and T.A. Manteuffel [90] in 2005. Another implementation was published by A.H. Baker, J.M. Den-

nis and E.R. Jessup [87] in 2003, see also [88]. Augmenting GMRES with approximate eigenvectors was mainly considered in a series of papers by R.B. Morgan. He proposed different implementations of this idea, GMRES-E [838] in 1995, GMRES-IR [839] in 2000 based on the implicitly restarted Arnoldi algorithm and GMRES-DR [840] in 2002 generalizing to nonsymmetric problems the idea of thick restarting by K. Wu and H. Simon [1125] in 2000.

The construction of the Hessenberg basis originated in the work of K. Hessenberg [624] in 1940. This basis was considered by J.H. Wilkinson [1120] page 377 as a particular case of the generalized Hessenberg process (as well as the Arnoldi process). It does not seem that this basis attracted much attention until H. Sadok used it to define a new Q-MR method in the paper [984] introducing CMRH in 1999. In 2012, H. Sadok and D.B. Szyld published theoretical comparisons of the residual norms in GMRES and CMRH in [986].

The biorthogonal basis was introduced by C. Lanczos in 1950 [721], see also [722]. For a good introduction to these methods, see M.H. Gutknecht [593]. BiCG was proposed by R. Fletcher in 1976 and published in the proceedings of the 1975 Dundee Conference on Numerical Analysis, see [459]. Originally the method was introduced to solve symmetric indefinite problems.

After their work on methods with long recurrences [1146] in 1980, D.M. Young and K.C. Jea considered Lanczos/Orthodir, Lancos/Orthores and Lanczos/Orthomin [673] in 1983; see also Jea's thesis [672].

The Conjugate Gradient Squared (CGS) was developed by P. Sonneveld in the 1980s. The paper was submitted to the SIAM J. on Scientific and Statistical Computing (SISSC) on April 24, 1984 but the revised version was only accepted on February 2, 1988 and finally published in January 1989.

A collaboration between Sonneveld and H. Van der Vorst resulted in 1990 in a report [1088] whose title was *CGSTAB, a more smoothly converging variant of CG-S*. Finally, H. van der Vorst published a paper whose title is *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems* [1086]. The paper was submitted to SISSC May 21, 1990, accepted for publication (in revised form) February 18, 1991 and finally published in March 1992.

M.H. Gutknecht proposed BiCGStab2, a complex version of BiCGStab in 1993 [590]. That same year, G.L.G. Sleijpen and D.R. Fokkema proposed BiCGStab(ℓ) [1017].

A review of Lanczos-type solvers for nonsymmetric linear systems of equations was published by M.H. Gutknecht in Acta Numerica in 1997 [593]. This 127-page paper describes many variants of Lanczos-like methods.

In 1991, C. Brezinski and H. Sadok [170] showed how to avoid breakdown in Sonneveld's CGS algorithm. Near-breakdowns were handled using formal orthogonal polynomials in [161] in 1994 by C. Brezinski and M. Redivo-Zaglia. Look-ahead for BiCGStab was considered in [162] in 1995. In 1998 they considered look-ahead for transpose-free Lanczos-type algorithms [163].

Communication-avoiding CGS and BiCGStab methods were described in the Ph.D. thesis [201] of E.C. Carson. A communication-hiding pipelined variant of BiCGStab was proposed in 2016 by S. Cools and W. Vanroose [279], see also [275, 278].

The QMR algorithms were introduced at the beginning of the 1990s as an improvement of BiCG. The first paper [474] in 1991 was by R.W. Freund and N.M. Nachtigal. They used the three-term Lanczos nonsymmetric process with look-ahead, see also Natchtigal's thesis. A joint paper with M.H. Gutknecht [470] was published in 1993. These papers followed the technical reports [473, 469, 476, 475] from 1990 to 1992. An algorithm using two-term recurrences was proposed [477] in 1994.

The TFQMR algorithm from R.W. Freund [468] used the CGS basis vectors to compute

iterates satisfying a quasi-minimal residual property. The paper was submitted in September 1991 and published in 1993. Note that TFQMR and the QMR algorithm of Freund and Nachtigal which uses A^T are not equivalent.

The development of IDR algorithms is largely due to P. Sonneveld. For the history of IDR methods, see Sonneveld's paper [1033]. The first algorithm was published in the proceedings of the IUTAM Symposium on Approximation Methods for Navier-Stokes Problems in 1980 as a contribution with P. Wesseling [1115]. The outcome of the 2008 revival of this type of algorithm was the first IDR(s) algorithm and the paper [1034] with M. Van Gijzen, see [1031, 1033].

In 2010, M.H. Gutknecht published an expository paper whose title is "IDR explained" [594] in which he gave details on the IDR algorithms, related them to other Krylov methods and summarized the early history of IDR methods. V. Simoncini and D.B. Szyld [1015] in 2010 showed how one can interpret IDR(s) as a Petrov-Galerkin method.

To improve the stability of the first IDR algorithm, M.B. Van Gijzen and P. Sonneveld proposed in 2011 another variant of IDR using biorthogonality properties [1090]. In 2011 T.P. Collignon and M.B. Van Gijzen studied how to minimize the number of synchronization points in IDR(s) for its implementation on parallel computers [271].

In 2013 O. Rendel, A. Rizvanolli and J.-P.M. Zemke [940] summarized what was known about IDR algorithms by describing in detail the generalized Hessenberg relations corresponding to different variants of IDR. This allows to transfer techniques known for classical Krylov subspace methods to IDR-based methods, in particular, to develop eigenvalue or Q-MR algorithms.

Recent developments on IDR are given in a paper by J.-P.M. Zemke [1152] who proposed variants in which some blocks of "basis" vectors are orthogonalized, this is called partial orthogonalization.

In 1999, M.-C. Yeung and T.F. Chan published a paper [1142] (received in 1997) describing a method called ML(k)BiCGStab which can be seen as a predecessor of IDR. It can be seen as a transpose-free extension à la BiCGStab of a method ML(k)BiCG which is using k left vectors. A simpler derivation was done by M.C. Yeung [1140] in 2012.

7
Preconditioning

The convergence of many iterative methods for solving a linear system depends on the condition number $\kappa(A)$ and/or the distribution of the eigenvalues of the matrix A. A natural idea is to transform the original linear system $Ax = b$ in such a way that the new system has the same solution (or a solution from which the original one is easily recovered) and the transformed matrix has better numerical properties for the iterative method to be used. Given a nonsingular matrix M (that is called a *preconditioner*), we can transform the linear system into

$$M^{-1}Ax = M^{-1}b. \tag{7.1}$$

Then we solve the system (7.1) instead of the original one. This is usually called *left preconditioning* since we multiply the matrix A from the left by M^{-1}. We could also transform the system by *right preconditioning* as

$$AM^{-1}y = b, \quad Mx = y. \tag{7.2}$$

If $M = M_1 M_2$, we could use a *two-sided preconditioner*,

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, \quad M_2 x = y. \tag{7.3}$$

If the matrix A is symmetric positive definite (SPD), it is generally important to keep the transformed system symmetric. Assuming that M is symmetric positive definite, we could solve

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}}y = M^{-\frac{1}{2}}b, \tag{7.4}$$

and we recover the solution of the original system by $M^{\frac{1}{2}}x = y$. Of course, we do not want to explicitly compute M^{-1} or $M^{-\frac{1}{2}}$. But, we can eliminate this problem by a change of variables and for all the methods we have described in this book the only thing we have to do is solving a linear system whose matrix is M at each iteration.

Considering a symmetric linear system, the properties that would like for the preconditioner M are

- M being symmetric and positive definite,
- if A is sparse, M must be sparse, since we do not want to use much more storage for M than for A,
- M being easy to construct since we do not want to spend most of the computing time constructing M,

- M such that a linear system $Mz = r$ is easy to solve,
- M such that we have good numerical properties for the iterative method to be used, whatever that means.

For most problems, it is almost impossible to construct preconditioners that satisfy all these criteria. The properties that are mandatory for sparse SPD matrices are the first two, and only the second one if the matrix A is nonsymmetric. For the third one, everything really depends on the rate of convergence that we could obtain for the preconditioned system (7.4). There is usually a trade-off between spending more time constructing a better preconditioner and having fewer iterations with a better convergence rate that could give a smaller iteration time. What is of interest is the total computing time. The choice also depends on knowing if we want to solve a single linear system or if we have to solve several systems with the same matrix and different right-hand sides. In that case, we can spend more time constructing a good preconditioner only once. The fourth item is also important since we must not generate auxiliary problems that are as hard to solve as the original one. The property about which we generally do not have too much control is the last one.

The optimal preconditioner giving the best rate of convergence is always $M = A$, but obviously this is not feasible since we won't get any gain over the original problem. However, this suggests that we would like to construct M such that M^{-1} is, in some sense, a good sparse approximation to A^{-1}.

There is an infinite number of degrees of freedom to construct preconditioners. It has been said that this is more an art than a science. One thing people generally agree on is that the more one knows about the problem to be solved, the easier it is to construct a "good" preconditioner. Hence, some preconditioners are very specific of the problems whence other ones try to be as general as possible. In the following sections we describe what are the most used general preconditioners. For the sake of simplicity, we mainly consider symmetric matrices but explain how the preconditioners can be extended to the nonsymmetric case.

7.1 ▪ Diagonal preconditioner

Using a diagonal preconditioner corresponds to a scaling of the matrix, see Section 2.9. The diagonal matrix D which is usually used is constructed from the diagonal entries of A.

Theorem 7.1. *If the matrix A is symmetric and has property A, that is, it is similar to*

$$\begin{pmatrix} D_1 & F \\ F^T & D_2 \end{pmatrix},$$

with D_1 and D_2 being square diagonal matrices, then the diagonal preconditioner that minimizes the condition number is D with $d_{i,i} = a_{i,i}$, and

$$\kappa(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) = \min_{\hat{D} \text{ diagonal}} \kappa(\hat{D}^{-\frac{1}{2}} A \hat{D}^{-\frac{1}{2}}).$$

Proof. See G.E. Forsythe and E.G. Straus [462]. □

If the matrix A does not have property A, the following result shows that D is not too far from being optimal when A is sparse.

Theorem 7.2. *If the matrix A is symmetric positive definite, then*

$$\kappa(D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) \leq p \min_{\hat{D}} \kappa(\hat{D}^{-\frac{1}{2}} A \hat{D}^{-\frac{1}{2}}),$$

where p is the maximum number of nonzero entries in any row of A.

Proof. See A. van der Sluis [1083]. □

However, these results do not tell if there really is an improvement of the condition number. For instance, for the Poisson model problem using the diagonal of the matrix A as preconditioner does not change the condition number since all the diagonal entries of A are equal to 4. A numerical check of the Forsythe and Straus result was done by A. Greenbaum and G.H. Rodrigue [570]. They used an optimization code to numerically compute the optimal diagonal preconditioners for several model problems. For the Poisson model problem, the code converges to the diagonal of A.

The main interest of a diagonal preconditioner is that it is well suited for parallel computers. However, in most cases it is not the best preconditioner, even on parallel computers.

One can also use a block diagonal preconditioner provided that the linear systems with the chosen diagonal blocks are easy to solve. Theorem 7.2 has been generalized to block diagonal preconditioners, in which case p is the number of blocks in a block row.

7.2 ▪ SSOR preconditioner

The diagonal preconditioner corresponds to using the matrix $M = D$ from the Jacobi splitting $A = M - N$ with $N = -(L + U)$. It was a natural idea to use the matrices arising from the Gauss-Seidel, SOR and SSOR iterative methods that we described in Chapter 4. When the matrix A is symmetric, one can use the symmetric Gauss-Seidel or the SSOR matrices.

The SSOR preconditioner has been proposed by D.J. Evans [430] and O. Axelsson [56, 57]. If the symmetric matrix A is written as $A = D + L + L^T$ where L is strictly lower triangular, the preconditioner M is defined as

$$M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega L^T), \quad 0 < \omega < 2. \tag{7.5}$$

The factor $\omega(2 - \omega)$ is just a technical convenience and does not change the results for the condition number. Taking $\omega = 1$ corresponds to the symmetric Gauss-Seidel preconditioner. The matrix M is straightforward to construct since it is directly obtained from the entries of A. Moreover, no additional storage is needed for M and linear systems $Mz = r$ are easy to solve by forward and backward sweeps computing the solutions of two triangular systems, but this is a problem on parallel computers. As we have seen in Chapter 4 a renumbering of the unknowns can be used to introduce some parallelism.

SSOR can be straightforwardly generalized to nonsymmetric matrices, $A = D + L + U$, setting

$$M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U).$$

Of course, M is not symmetric and Gauss-Seidel can be used instead.

Proposition 7.3. *Let A be a symmetric positive definite matrix and λ_i be the eigenvalues of $M^{-1}A$ where M is defined by (7.5). Then, $\lambda_i \in (0, 1], \forall i$.*

Proof. Since A is positive definite, the diagonal of D is strictly positive and M is positive definite. The matrix $M^{-1}A$ is similar to a symmetric matrix and its eigenvalues are real. Let

$$B = \frac{2-\omega}{\omega}D, \quad C = \frac{\omega-1}{\omega}D + L.$$

We can write M and A as

$$M = (B + C)B^{-1}(B + C^T), \quad A = B + C + C^T.$$

It implies that $M = A + CB^{-1}C^T$. From this relation, B being positive definite, we obtain

$$(Mx, x) = (Ax, x) + (B^{-1}C^T x, C^T x) \geq (Ax, x) > 0, \ \forall x \neq 0.$$

This proves that the largest eigenvalue of $M^{-1}A$ is less than or equal to 1. □

If $\omega = 1$, 1 is an eigenvalue of $M^{-1}A$ with the first column e_1 of the identity matrix as an eigenvector. If $\omega \neq 1$, all the eigenvalues of $M^{-1}A$ are strictly less than 1.

The condition number $\kappa(M^{-1}A)$ was studied in the context of problems arising from finite element methods by O. Axelsson [57] who proved the following result.

Theorem 7.4. *Let M be the SSOR preconditioner and let μ and δ be real numbers such that*

$$\max_{x \neq 0} \frac{(Dx, x)}{(Ax, x)} \leq \mu, \quad \max_{x \neq 0} \frac{(LD^{-1}L^T x, x) - 1/4(Dx, x)}{(Ax, x)} \leq \delta.$$

Then, there exists an optimal parameter

$$\omega_{opt} = \frac{2}{1 + 2\sqrt{\frac{1}{\mu}(\frac{1}{2} + \delta)}},$$

such that

$$\kappa(M^{-1}A) \leq \frac{1}{2} + \sqrt{(\frac{1}{2} + \delta)\mu}.$$

For $\omega = 1$, the condition number is bounded by

$$\kappa(M^{-1}A) \leq \frac{1}{2} + \frac{\mu}{4} + \delta.$$

□

For the Poisson model problem we have

$$\mu = \frac{4}{\lambda_{max}(A)}, \quad \delta = 0.$$

Theorem 7.4 shows that there exists an optimal ω ($\approx 2/(1 + h)$) such that

$$\kappa(M^{-1}A) \leq \frac{1}{2} + \frac{1}{2\sin\left(\frac{\pi h}{2}\right)} = O\left(\frac{1}{h}\right).$$

Remember that for that problem $\kappa(A) = O\left(\frac{1}{h^2}\right)$. Therefore, we gain an order of magnitude if we are able to compute the optimal value of ω.

We can also easily define a block SSOR preconditioner provided the diagonal blocks are nonsingular. As for the point case, the eigenvalues are in $(0, 1]$, see [811].

7.3 ▪ Incomplete factorizations

Factorizations arising from Gaussian elimination and its variants have been used to define preconditioners. The main idea is to neglect some entries during the elimination process. Let us start with a Cholesky factorization and the outer product algorithm described in Chapter 2. If A is a sparse symmetric positive definite matrix, this algorithm can go through since all the pivots are nonzero. At each step some fill-in may be generated. As we have seen, the fill-ins can only appear within the profile of the matrix A. For symmetric M-matrices or positive definite matrices, there are also some properties of decay of the absolute values of the entries of the L factor, see [809]. Therefore, a way to generate an approximation of A, is to neglect a part of or all the fill-in during the steps of the factorization. There are different ways to accomplish this.

Let us first assume that we have a given set of indices $G = \{(i,j),\ i > j\}$. We would like to construct a splitting,

$$A = L\Sigma L^T - R,$$

L being lower triangular with $\ell_{i,i} = 1$ for $i = 1, \ldots, n$ and Σ diagonal. We construct L in such a way that $\ell_{i,j} = 0$ if $(i,j) \notin G$. Let us describe the first steps of the algorithm,

$$A = A_1 = \begin{pmatrix} a_{1,1} & a_1^T \\ a_1 & B_1 \end{pmatrix} = \begin{pmatrix} a_{1,1} & b_1^T \\ b_1 & B_1 \end{pmatrix} - \begin{pmatrix} 0 & r_1^T \\ r_1 & 0 \end{pmatrix} = M_1 - R_1,$$

with

$$a_1 = b_1 - r_1,$$
$$(b_1)_i = 0, \text{ if } (i,1) \notin G \Rightarrow (r_1)_i = -(a_1)_i,$$
$$(b_1)_i = (a_1)_i, \text{ if } (i,1) \in G \Rightarrow (r_1)_i = 0.$$

Then, if this is feasible, we factorize the matrix M_1,

$$M_1 = \begin{pmatrix} 1 & 0 \\ \ell_1 & I \end{pmatrix} \begin{pmatrix} a_{1,1} & 0 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} 1 & \ell_1^T \\ 0 & I \end{pmatrix} = L_1 \Sigma_1 L_1^T.$$

By identification, we obtain

$$\ell_1 = \frac{b_1}{a_{1,1}},$$
$$A_2 = B_1 - \frac{1}{a_{1,1}} b_1 b_1^T.$$

Then, we use the same process on the matrix A_2,

$$A_2 = \begin{pmatrix} a_{2,2}^{(2)} & a_2^T \\ a_2 & B_2 \end{pmatrix} = \begin{pmatrix} a_{2,2}^{(2)} & b_2^T \\ b_2 & B_2 \end{pmatrix} - \begin{pmatrix} 0 & r_2^T \\ r_2 & 0 \end{pmatrix} = M_2 - R_2,$$

where b_2 is obtained from a_2 by zeroing the entries whose indices $(i,2)$ do not belong to G. We define

$$L_2 = \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ \ell_2 & I \end{pmatrix} \end{pmatrix}.$$

With this notation, at the end of the second step, we have

$$A = L_1 L_2 \Sigma_2 L_2^T L_1^T - L_1 \begin{pmatrix} 0 & 0 \\ 0 & R_2 \end{pmatrix} L_1^T - R_1,$$

with

$$L_1 L_2 = \begin{pmatrix} 1 & 0 \\ \ell_1 & \begin{pmatrix} 1 & 0 \\ \ell_2 & I \end{pmatrix} \end{pmatrix} \quad \text{and } L_1 \begin{pmatrix} 0 & 0 \\ 0 & R_2 \end{pmatrix} L_1^T = \begin{pmatrix} 0 & 0 \\ 0 & R_2 \end{pmatrix}.$$

Of course, it is not mandatory to have ones on the diagonal of L and we have the same variants as for the complete factorization.

The previous algorithm constructs a complete factorization of $M = A + R$, where the matrix R is a priori unknown. Such an algorithm for M-matrices was described by J.A. Meijerink and H.A. van der Vorst [795]. An interesting question is to know under which conditions such a decomposition is feasible. We observe that the previous algorithm, which is generically known as IC for Incomplete Cholesky, can be easily generalized to nonsymmetric matrices (ILU) provided all the pivots are nonzero. As we said,any variant of Gaussian elimination can be used to construct incomplete factorizations; see, for instance, N. Li, Y. Saad, and E. Chow [738].

Theorem 7.5. *Let A be a nonsingular H-matrix with positive diagonal entries. Then, all the pivots of the incomplete factorization are nonzero for any set of indices G.*

Proof. First, let us show that if we set to zero some non-diagonal entries of an H-matrix A, we obtain an H-matrix. Let $A = B + R$, R having a zero diagonal and $b_{i,j} = a_{i,j}$ or 0. There exists a diagonal matrix E with a positive diagonal whose entries are denoted by $e_{i,i}$ such that $E^{-1}AE$ is strictly diagonally dominant, that is,

$$|a_{i,i}| > \sum_{i \neq j} |a_{i,j}| \frac{e_{j,j}}{e_{i;i}}.$$

The matrix $E^{-1}BE$ is also strictly diagonally dominant since

$$|b_{i,i}| = |a_{i,i}| > \sum_{i \neq j} |a_{i,j}| \frac{e_{j,j}}{e_{i,i}} > \sum_{i \neq j} |b_{i,j}| \frac{e_{j,j}}{e_{i,i}}.$$

The second step of the proof is to show that in one step of the complete factorization, if we start from an H-matrix, then we obtain an H-matrix. Assume that at step k, A_k is an H-matrix. There exists a diagonal matrix E_k whose diagonal entries are denoted $e_i^{(k)} > 0$ and

$$a_{i,i}^{(k)} e_i^{(k)} > \sum_{\substack{j \neq i \\ j \geq k}} |a_{i,j}^{(k)}| e_j^{(k)}.$$

We have

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{1}{a_{k,k}^{(k)}} a_{k,i}^{(k)} a_{k,j}^{(k)}.$$

Therefore,

$$\begin{aligned} \sum_{\substack{j \neq i \\ j \geq k+1}} |a_{i,j}^{(k+1)}| e_j^{(k)} &= \sum_{\substack{j \neq i \\ j \geq k+1}} |a_{i,j}^{(k)} - \frac{1}{a_{k,k}^{(k)}} a_{k,i}^{(k)} a_{k,j}^{(k)}| e_j^{(k)}, \\ &\leq \sum_{\substack{j \neq i \\ j \geq k+1}} |a_{i,j}^{(k)}| e_j^{(k)} + \frac{|a_{k,i}^{(k)}|}{a_{k,k}^{(k)}} \sum_{\substack{j \neq i \\ j \geq k+1}} |a_{k,j}^{(k)}| e_j^{(k)}. \end{aligned}$$

But,

$$\sum_{\substack{j \neq i \\ j \geq k+1}} |a_{i,j}^{(k)}| e_j^{(k)} = \sum_{\substack{j \neq i \\ j \geq k}} |a_{i,j}^{(k)}| e_j^{(k)} - |a_{i,k}^{(k)}| e_k^{(k)} \leq a_{i,i}^{(k)} e_i^{(k)} - |a_{i,k}^{(k)}| e_k^{(k)}.$$

We also have

$$\sum_{\substack{j \neq i \\ j \geq k+1}} |a_{k,j}^{(k)}| e_j^{(k)} = \sum_{j \geq k+1} |a_{k,j}^{(k)}| e_j^{(k)} - |a_{k,i}^{(k)}| e_i^{(k)} \leq a_{k,k}^{(k)} e_k^{(k)} - |a_{k,i}^{(k)}| e_i^{(k)}.$$

It yields

$$\begin{aligned}\sum_{\substack{j \neq i \\ j \geq k+1}} |a_{i,j}^{(k+1)}| e_j^{(k)} &\leq a_{i,i} e_i^{(k)} - |a_{i,k}^{(k)}| e_k^{(k)} \\ &\quad + \frac{|a_{k,i}^{(k)}|}{a_{k,k}^{(k)}} (a_{k,k}^{(k)} e_k^{(k)} - |a_{k,i}^{(k)}| e_i^{(k)}) \\ &\leq (a_{i,i}^{(k)} - \frac{1}{a_{k,k}^{(k)}} |a_{k,i}^{(k)}|^2) e_i^{(k)} = a_{i,i}^{(k+1)} e_i^{(k)}.\end{aligned}$$

This shows that $a_{i,i}^{(k+1)} > 0$ and the matrix at step $k+1$ of the factorization is an H-matrix. Putting together the two results that we have just proved, we have that at each step of the incomplete factorization we are left with an H-matrix and the process can go on to completion whatever the set of indices G is. □

If we suppose that A is a symmetric M-matrix, we have the following result.

Theorem 7.6. *Let A be a nonsingular symmetric M-matrix. Then, for all sets of indices G, the incomplete Cholesky factorization,*

$$A = L \Sigma L^T - R,$$

is a regular splitting.

Proof. With almost the same proof as for Theorem 7.5, we can show that starting from an M-matrix, the matrices A_k are also M-matrices. Therefore, $a_{i,i}^{(k)} > 0$ and $a_{i,j}^{(k)} \leq 0,\ i \neq j$. It implies that $R_k \geq 0$ and $(l_k)_j \leq 0$. Then,

$$\begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ \ell_k & I \end{pmatrix} \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ -\ell_k & I \end{pmatrix} \end{pmatrix} \geq 0.$$

Hence, $R \geq 0$ and $L^{-1} \geq 0$ and this shows that we have a regular splitting. □

R.S. Varga, E.B. Saff, and V. Mehrmann [1100] characterized the matrices that are incompletely factorizable. Let $\mathcal{F}_n$ be this set and $\mathcal{H}_n$ be the set of nonsingular H-matrices. We know that $\mathcal{H}_n \subset \mathcal{F}_n$. Let

$$\begin{aligned}\Omega(A) &= \{B |\ M(B) = M(A)\}, \\ \Omega^d(A) &= \{B |\ |b_{i,i}| = |a_{i,i}|,\ |b_{i,j}| \leq |a_{i,j}|, i \neq j\}, \\ \mathcal{F}_n^d &= \{A \in \mathcal{F}_n, \Omega^d(A) \subseteq \mathcal{F}_n\}.\end{aligned}$$

Varga et al. showed that $\mathcal{H}_n = \mathcal{F}_n^d$. Moreover, if

$$\mathcal{F}_n^c = \{A \in \mathcal{F}_n,\ \Omega(A) \subseteq \mathcal{F}_n\},$$

$\mathcal{H}_n$ is strictly contained in $\mathcal{F}_n^c$ which is also strictly contained in $\mathcal{F}_n$. It shows that there exist matrices that can be incompletely factorized which are not H-matrices.

One can find examples of SPD matrices which are not H-matrices and for which the incomplete Cholesky factorization fails. It was suggested by T.A. Manteuffel [782] to factorize $A(\alpha) = (1+\alpha)D + L + L^T$ instead of A, where α is a positive real parameter chosen such that $A(\alpha)$ is an H-matrix. For instance, α can be chosen to make $A + \alpha I$ diagonally dominant. However, it is not always easy to find a value of the parameter such that the incomplete factorization does not fail and such that the number of iterations is reduced.

In the nonsymmetric case, one can also use pivoting. An ILU factorization with pivoting based on monitoring the growth of the inverse factors was proposed by M. Bollhöfer [136, 137].

An algorithm for SPD matrices was proposed by Y. Robert [946] using ideas from A. Jennings and G.M. Malik [679]. Let

$$A = \begin{pmatrix} a_{1,1} & a_1^T \\ a_1 & B_1 \end{pmatrix}, \quad a_1 = b_1 - r_1.$$

The remainder R_1 is constructed as

$$R_1 = \begin{pmatrix} r_{1,1}^1 & r_1^T \\ r_1 & D_{R_1} \end{pmatrix},$$

with D_{R_1} diagonal such that $(D_{R_1})_{j,j} = |(r_1)_j|$ and $r_{1,1}^1 = \sum_{j=1}^{n-1} |(r_1)_j| > 0$. We split A as

$$A = A_1 = \begin{pmatrix} a_{1,1} + r_{1,1}^1 & b_1^T \\ b_1 & B_1 + D_{R_1} \end{pmatrix} - \begin{pmatrix} r_{1,1}^1 & r_1^T \\ r_1 & D_{R_1} \end{pmatrix} = M_1 - R_1, \tag{7.6}$$

and we factorize the matrix M_1,

$$M_1 = \begin{pmatrix} 1 & 0 \\ \ell_1 & I \end{pmatrix} \begin{pmatrix} a_{1,1} + r_{1,1}^1 & 0 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} 1 & \ell_1^T \\ 0 & I \end{pmatrix}.$$

By identification we obtain,

$$\ell_1 = \frac{b_1}{a_{1,1} + r_{1,1}^1}, \quad A_2 = B_1 + D_{R_1} - \frac{1}{a_{1,1} + r_{1,1}^1} b_1 b_1^T.$$

Proposition 7.7. *In the previous construction, R_1 is semi positive definite and A_2 is positive definite.*

Proof. R_1 is the sum of matrices of the following form,

$$\begin{pmatrix} \begin{pmatrix} |r| & 0 & \dots & 0 & r \\ 0 & \ddots & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & & \ddots & 0 \\ r & 0 & \dots & 0 & |r| \end{pmatrix} & 0 \\ 0 & \end{pmatrix},$$

with only four nonzero entries. The eigenvalues of this matrix are 0 and $2|r|$. Therefore, M_1 is semi-positive definite, and $M_1 = A + R_1$ is positive definite. Let y be such that

$$y = \begin{pmatrix} \alpha \\ x \end{pmatrix},$$

with $\alpha \in \mathbb{R}$. We have $(M_1 y, y) > 0$ which yields

$$\alpha^2(a_{1,1} + r^1_{1,1}) + 2\alpha(b_1, x) + ((B_1 + D_{R_1})x, x) > 0.$$

If we choose $\alpha = -\frac{(b_1,x)}{a_{1,1}+r^1_{1,1}}$, we have

$$-\frac{(b_1, x)^2}{a_{1,1} + r^1_{1,1}} + (x, (B_1 + D_{R_1})x) > 0,$$

which shows that A_2 is positive definite. □

Then, we use the same algorithm for A_2. In this factorization, R is positive definite and the eigenvalues of $M^{-1}A$ are less than 1.

This preconditioner is a so-called modified incomplete factorization. More generally, the idea is to try to compensate for entries left in the remainder R by summing them up to the diagonal of M. An early method (known as DKR), derived for for matrices arising from finite difference approximations of elliptic partial differential equations in rectangles, was proposed by T.F. Dupont, R.P. Kendall, and H.H. Rachford [391], following some older ideas by N.I. Buleev [181] and R.S. Varga [1097] in the 1960s. This method was studied by I. Gustafsson [585].

This type of method can be extended to more general sparse problems (MIC). Let us assume that we have a diagonally dominant symmetric M-matrix. We use a splitting as in (7.6), but with r_1 such that $(r_1)_j = -a_{j,1}$ if $(j, 1) \notin G$, $r^{(1)}_{1,1} = -\sum_{j=2}^n (r_1)_j \leq 0$, and $D^{(1)}_R$ a diagonal matrix of order $n - 1$ such that $(D^{(1)}_R)_{j,j} = -(r_1)_j$. Note that here we subtract something from the diagonal. The rowsums of R_1 are equal to zero, $a_{1,1} + r^{(1)}_{1,1} > 0$ and $B_1 + D^{(1)}_R$ is diagonally dominant. It means that the algorithm can proceed with the next steps. A small perturbation can be added to the diagonal to obtain a strictly diagonally dominant matrix. For more general M-matrices which are generalized diagonally dominant with a vector d (such that $Ad > 0$), we could require that $Rd = 0$. These methods are easily extended to the nonsymmetric case (MILU).

The relaxed incomplete Cholesky factorization (RIC or RILU in the nonsymmetric case) is basically the same as MIC except that the values that are added to the diagonal are multiplied by a relaxation parameter ω such that $0 \leq \omega \leq 1$. Therefore, $\omega = 0$ gives IC and $\omega = 1$ gives the unperturbed MIC.

Bounds for the eigenvalues and the condition number are given in O. Axelsson's book [64].

Theorem 7.8. *Let A and M be SPD matrices and let $\mu_1, \mu_2 > 0 \in \mathbb{R}$ such that $\lambda_{\max}(\mu_1 M - A) \geq 0$, $\lambda_{\min}(\mu_2 M - A) \geq 0$. Then,*

$$\frac{\mu_1 \lambda_i(A)}{\lambda_i(A) + \lambda_{\max}(\mu_1 M - A)} \leq \lambda_i(M^{-1}A) \leq \frac{\mu_2 \lambda_i(A)}{\lambda_i(A) + \lambda_{\min}(\mu_2 M - A)}.$$

□

Axelsson's result can be used to derive upper bounds for $\lambda_{\max}(M^{-1}A)$ and lower bounds for $\lambda_{\min}(M^{-1}A)$. If $A = \alpha I + L + L^T$ where α is a positive real number, $M = (X + L)X^{-1}(X + L^T)$ with X being an M-matrix and $\beta = \lambda_{\min}(X)$, then if $2\beta - \alpha > 0$,

$$\lambda_i(M^{-1}A) \leq \begin{cases} \frac{4\beta\lambda_i(A)}{(2\beta-\alpha+\lambda_i(A))^2}, & \text{if } \lambda_i(A) \leq 2\beta - \alpha, \\ \frac{\beta}{2\beta-\alpha}, & \text{if } \lambda_i(A) \geq 2\beta - \alpha. \end{cases}$$

For the incomplete Cholesky factorization where the nonzero structure of L is the same as that of the lower triangular part of A and the Poisson model problem $\beta \approx 2 + \sqrt{2}$ and it yields

$$\lambda_{\max}(M^{-1}A) \leq \frac{2+\sqrt{2}}{2\sqrt{2}} \approx 1.2071\ldots$$

Moreover, we have

$$\frac{\lambda_i(A)}{\lambda_i(A) + \frac{2}{\beta}} \leq \lambda_i(M^{-1}A) \leq \frac{4(2+\sqrt{2})\lambda_i(A)}{(2\sqrt{2} + \lambda_i(A))^2}.$$

It shows that $\lambda_{\min(M^{-1}A)} = O(h^2)$. For the smallest eigenvalues we have

$$\lambda_i(M^{-1}A) = \frac{2+\sqrt{2}}{2}\lambda_i(A) + O(\lambda_i(A)^2).$$

This result essentially shows that the distribution of the smallest eigenvalues of $M^{-1}A$ is almost the same as for the original problem. For this incomplete factorization and this problem, R. Chandra [227] proved that

$$\frac{1}{17}\kappa(A) \leq \kappa(M^{-1}A) \leq 17\kappa(A),$$

That is, the condition number of $M^{-1}A$ is of the same order as the condition number of A. As far as we know, there is no theoretical results for the nonsymmetric case.

Since we know how to compute some incomplete factorizations, we need a strategy for choosing the set of indices G for the nonzero entries in L (and U in the nonsymmetric case). The most widely used strategy is to use a sparsity pattern of L identical to that of the lower triangular part of A. This is generally known as IC(0) or ILU(0). However, more accurate and efficient factorizations can be obtained by retaining a part of the fill-in that is generated.

There exist several ways to enlarge the set G, starting from the nonzero pattern of the lower triangular part of A. One possibility is to associate a number called the *level* to any entry computed in the factorization. Entries which are nonzero in A are of level 0 and all the zero entries are of level ∞. Then, in the steps of the decomposition, fill-ins generated by level 0 entries are said of to be of level 1, fill-ins generated by fill-ins are of level 2, etc... This is formalized in the following way. Let $lev_{i,j}^{(k)}$ be the level of fill-in of $a_{i,j}^{(k)}$, then, after one elimination step, the new level is computed as

$$lev_{i,j}^{(k+1)} = \max(lev_{i,j}^{(k)}, lev_{i,k}^{(k)} + lev_{j,k}^{(k)} + 1).$$

The strategy for the incomplete factorization is to keep only the fill-ins having a level which is below a given threshold p. When the matrix is symmetric we have seen that the structure of the complete factors can be determined in a preprocessing phase. The same thing holds for the incomplete factorization where the level of fill-in and the structure of the incomplete factor L can be computed before the numerical factorization takes place. The rationale behind this strategy is that if some fill-ins are small, then, the fill-ins generated by those previous fill-ins will be even smaller and so on. Therefore, neglecting the fill-ins beyond a certain level is supposed to drop only small entries.

Another strategy was introduced in the 1970s by A.D. Tuff and A. Jennings [1076] and others. In this method, we not only look at the structure of the incomplete factors but also at the actual (absolute) values of the entries. At some step k, the fill-in is computed and it is discarded

or not according to some dropping strategy. The most used one is to compare $|a_{i,j}^{(k+1)}|$ to some threshold and to drop the fill-in if it is too small. Different ways of dropping the entries have been proposed over the years. For instance, one can drop the entry if

$$|a_{i,j}^{(k+1)}| < \epsilon \, [\max_{\ell \geq k+1} |a_{i,\ell}^{(k)}|, \max_{\ell \geq k+1} |a_{j,\ell}^{(k)}|].$$

Another possibility is to compare $|a_{i,j}^{(k+1)}|$ to the norm of the ith row of A. The main drawback of these methods is that the structure of L is not known before the elimination as when factoring nonsymmetric sparse matrices with pivoting. Moreover, we do not know a priori how much storage is needed. This kind of method can be slightly modified and, for instance, the fill-ins in the kth column can be compared to a threshold, but only the p largest ones are kept provided they satisfy the given criteria. In this way the amount of storage can be controlled. This type of method for nonsymmetric problems (named ILUT) was considered by Y. Saad [976], see also [770, 12].

The following code computes the incomplete Cholesky factorization IC(0), $A = LDL^T$ where L has a unit diagonal and the vector d is the diagonal of D.

```
function [L,d] = IC0(A)
B = A;
n = size(A,1);
L = sparse(n,n);
d = zeros(n,1);
for k=1:n-1
 m = size(B,1);
 b1 = 1 / B(1,1);
 i = find(A(k:n,k));
 sl = sparse(i,1,B(i,1)*b1,m,1);
 L(k:n,k) = sl;
 L(k,k) = 1;
 d(k) = B(1,1);
 % Schur complement
 ind = i(2:end)-1;
 sl = sl(2:m);
 BB = B(2:m,2:m);
 % do not take care of symmetry (faster)
 for i=ind
  BB(i,ind) = BB(i,ind) - B(1,1) * sl(i) * sl(ind)';
 end
 B = BB;
end
L(n,n) = 1;
d(n) = B(1,1);
ind = find(d <= 0);
if length(ind) > 0
 error('Pb with the incomplete factorization, negative diagonal
     entries')
end % if
```

A (non-optimal) code for computing a level-based IC factorization is following.

```
function [L,d] = IC_level(A)
B = A;
n = size(A,1);
L = sparse(n,n);
Lev = Inf * ones(n,n);
for i=1:n
 for j=1:n
  if A(i,j) ~= 0
   Lev(i,j) = 1;
  end % if
 end % for j
end % for i
d = zeros(n,1);
for k=1:n-1
 m = size(B,1);
 b1 = 1 / B(1,1);
 ii = find(B(:,1));
 sl = sparse(ii,1,B(ii,1)*b1,m,1);
 L(k:n,k) = sl;
 % dropping strategy
 i = find(Lev(2:end,1) > levmax);
 L(i+k,k) = zeros(length(i),1);
 L(k,k) = 1;
 d(k) = B(1,1);
 ind = find(L(k+1:n,k))';
 sl = sl(2:m);
 % update of levels
 for i=1:m
  for j=1:m
   Lev(i,j) = min(Lev(i,j), Lev(i,1) + Lev(1,j) + 1);
  end % for j
 end % for j
 BB = B(2:m,2:m);
 Lev = Lev(2:m,2:m);
 % Schur complement
 % do not take care of symmetry (faster)
 for i = ind
  BB(i,ind) = BB(i,ind) - B(1,1) * sl(i) * sl(ind)';
 end % for i
 B = BB;
end
L(n,n) = 1;
d(n) = B(1,1);
ind = find(d <= 0);
if length(ind) > 0
 error('Pb with the incomplete factorization, negative diagonal
     entries')
end
```

IC is one of the most popular preconditioners since it is relatively efficient on most problems

and easy to implement. We have seen in Chapter 3 that for Gaussian elimination the issue of how the unknowns are ordered is of great importance. Some orderings as the reverse Cuthill-McKee, the minimum degree or nested dissection can reduce the work and storage for the factorization of the matrix. The effect of the ordering of the unknowns for incomplete factorizations was studied experimentally for symmetric problems arising for finite difference matrices methods by I.S. Duff and G.M. [377].

Seventeen different orderings are considered in [377]. We simply report results from [377] for orderings we have studied in Chapter 3. They are listed in Table 7.1 together with the abbreviations used in the tables.

Table 7.1. *Orderings*

ordering	Abbreviation
row ordering	row
reverse Cuthill-McKee	rcm
minimum degree	mind
nested dissection	nest

In tables 7.2-7.3 we give the number of PCG iterations using IC(0) as a preconditioner to satisfy $\|r_k\| \leq 10^{-6}\|b\|$, the number of modifications that would occur in the complete factorization, the number of entries in the remainder R, the Frobenius norm of R and $\max_{i,j} |r_{i,j}|$. The number of operations per iteration is the same for all orderings, therefore the number of iterations is a good measure of the relative merits of the orderings. All examples use a 30×30 regular grid, that is, $n = 900$.

Table 7.2. *Results for the Poisson model problem, IC(0)*

ordering	# of iter.	# of modifs	# of elts in R	$\|R\|_F^2$	$\max_{i,j} \|r_{i,j}\|$
row	23	24389	841	142.5	0.293
rcm	23	16675	841	142.5	0.293
mind	39	7971	1582	467.3	0.541
nest	25	15228	1012	157.1	0.293

The second test problem is arising from the finite difference discretization of a 2D diffusion equation with a diffusion coefficient of 100 in the x direction and 1 in the y direction.

Table 7.3. *Results for the anisotropic problem, IC(0)*

ordering	# of iter.	# of modifs	# of elts in R	$\|R\|_F^2$	$\max_{i,j} \|r_{i,j}\|$
row	9	24389	841	$0.12\ 10^4$	0.87
rcm	9	16675	841	$0.12\ 10^4$	0.87
mind	48	7971	1582	$0.18\ 10^7$	49.51
nest	26	15228	1012	$0.43\ 10^6$	49.51

From these results, we see that the number of PCG iterations is not related to the number of fill-ins we are dropping, but it is almost directly related to the norm of the remainder matrix R. The number of fill-ins is related to the structure of the matrix, but the incomplete factorization is

dependent on the value of the entries. Some orderings like the minimum degree have very small number of fill-ins but a "large" matrix R and give a large number of iterations.

Table 7.4. *Results for the Poisson model problem, MIC(0)*

ordering	# of iter.	# of modifs	# of elts in R	$\|R\|_F^2$	$\max_{i,j} \|r_{i,j}\|$
row	18	24389	1741	1010.	0.979
rcm	18	16675	1741	1010.	0.979
mind	>200	7971	2482	3568.	3.000
nest	38	15228	1912	1107.	1.666

Let us now consider the modified incomplete decomposition MIC. As we can see in Table 7.4, the norm $\|R\|_F$ is much larger than for IC(0), although the number of iterations for the row ordering is smaller. Some of the orderings do not even converge when the modified factorization is used. Some other experiments were conducted keeping more fill-in. The first one uses a single level of fill-in. In tables 7.5 and 7.6, we add a column giving the number of entries in L.

Table 7.5. *Results for the Poisson model problem, IC with one level of fill-in*

ordering	# iter.	# modifs	# elts in R	$\|R\|_F^2$	$\max_{i,j} \|r_{i,j}\|$	# elts in L
row	17	24389	1653	2.43	0.087	3481
rcm	17	16675	1653	2.43	0.087	3481
mind	23	7971	2467	38.81	2.509	4282
nest	19	15228	2187	35.34	0.173	3652

Table 7.5 shows the results of IC using one level of fill-in. The second experiment with results in Table 7.6 uses a drop tolerance to keep or discard the fill-ins.

Table 7.6. *Results for the Poisson model problem, IC, tol=0.05*

ordering	# iter.	# modifs	# elts in R	$\|R\|_F^2$	$\max_{i,j} \|r_{i,j}\|$	# elts in L
row	12	24389	1595	4.26	0.039	4293
rcm	10	16675	1540	2.846	0.041	4293
mind	10	7971	1657	2.285	0.049	5531
nest	12	15228	2622	3.890	0.049	5574

For the factorizations using one level of fill-in, the reduction in the number of iterations does not compensate for the extra work done within each iteration. The reverse is true for the drop tolerance results where the greater reduction in iterations more than compensates the increased work. An interesting feature is that the relative performance of the different ordering schemes has changed. For example, the minimum degree algorithm does much better when some fill-in is allowed in L. An explanation is that many of the first level fill-ins for these orderings are quite large, unlike for the standard row ordering where the values of the fill-ins rapidly decrease.

These experimental results about the effect of changing the ordering before the incomplete factorization have been partly explained by V. Eijkhout [401] and S. Doi and A. Lichnewsky [345, 346] who also provided numerical experiments with nonsymmetric problems. Orderings

for nonsymmetric problems were also considered by M. Benzi, D.B. Szyld, and A. Van Duin [118]; see also L. Oliker, X. Li, P. Husbands, and R. Biswas [877], as well as D. Osei-Kuffuor, R. Li, and Y. Saad [885].

E.F. D'Azevedo, P.A. Forsyth and W.-P. Tang [319] proposed an algorithm for constructing an ordering for general sparse matrices that reduces the discarded fill-in during the incomplete factorization. This is called the Minimum Discarded Fill (MDF) algorithm. The main problem with MDF is that the construction of the preconditioner is quite costly.

A popular technique to introduce parallelism in the computation and use of an incomplete factorization is using a multicolor ordering. The nodes of the graph of the matrix A are colored with several different colors in such a way that the neighbours of a node of a given color all are of a different color. In earlier studies, like the one by E.L. Poole and J.M. Ortega [921], only a small number of colors was considered. Their conclusion was that the convergence of the iterative method was slowed down. This was also the conclusion in [377] where a red-black and a 4-color ordering were used. Orderings with a large number of colors were later considered by S. Doi and A. Hoshi [344] and S. Fujino and S. Doi [481] with some successes.

A completely different method for computing an ILU factorization was proposed by E. Chow and A. Patel [255]. They used the fact that

$$[LU]_{i,j} = a_{i,j}, \quad (i,j) \in G.$$

Taking L with a unit diagonal, this relation can be written for $(i,j) \in G$ as

$$\ell_{i,j} = \frac{1}{u_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} \ell_{i,k} u_{k,j} \right),$$
$$u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} \ell_{i,k} u_{k,j}.$$

This can be considered as a nonlinear system where the unknowns are the nonzero entries in the strictly lower triangular part of L and in U. This system can be written as $w = F(w)$ and solved by an iterative method. Since the formulas above for (i,j) depend only on other unknowns in row i of L to the left of j, and in column j of U above i, there is some structure in the mapping F. Iterative methods which are the nonlinear equivalents of the linear Jacobi and Gauss-Seidel methods and their convergence were considered in [255]. Different results are obtained by changing the ordering of the unknowns. In many cases it is only necessary to do a few iterations (called "sweeps" in [255]) to obtain a good ILU factorization. One of the interest of this method is that it has a high degree of parallelism.

A method which combine such a fixed point iteration for approximating the incomplete factors for a given sparsity pattern with a process that adaptively changes the sparsity pattern was proposed by H. Anzt, E. Chow, and J.J. Dongarra [38]. Nonzero entries are added and removed from the sparsity pattern in each adaptive step. Only one sweep of the fixed point method is used in between adjusting the sparsity pattern. To add nonzero entries to the sparsity pattern the remainder $R = A - LU$ is used. The candidate indices are the union of the nonzero entries in A and the matrix product LU that are not already in the current sparsity pattern. A candidate can be added to G if $|r_{i,j}|$ is larger than a given threshold, or all candidates can be added to G. Entries with a small absolute value in L and U are removed from the sparsity pattern. Additionally, the number of entries added or removed can be controlled by given parameters.

Of course, block incomplete factorizations have been developed. Let us first consider block

tridiagonal matrices. Let A be a symmetric matrix of order n written in block form as

$$A = \begin{pmatrix} D_1 & A_2^T & & & \\ A_2 & D_2 & A_3^T & & \\ & \ddots & \ddots & \ddots & \\ & & A_{m-1} & D_{m-1} & A_m^T \\ & & & A_m & D_m \end{pmatrix},$$

each block being square of order m, $n = m^2$. Let L the block lower triangular part of A. A block (complete) factorization of A can be written as

$$A = (\Sigma + L)\Sigma^{-1}(\Sigma + L^T),$$

where Σ is a block diagonal matrix whose diagonal blocks are denoted by Σ_i. By inspection, the diagonal blocks are given by

$$\Sigma_1 = D_1, \quad \Sigma_i = D_i - A_i(\Sigma_{i-1})^{-1}A_i^T,\ i = 2, \ldots, m$$

For instance, for finite difference discretizations of elliptic (or parabolic) partial differential equations, the matrices D_i are tridiagonal and the matrices A_i are diagonal. This implies that Σ_1 is tridiagonal, but all the other matrices $\Sigma_i, i = 2, \ldots, m$ are dense. The idea to obtain a block incomplete factorization is to replace the inverses by sparse approximations. Let

$$M = (\Delta + L)\Delta^{-1}(\Delta + L^T),$$

where Δ is a block diagonal matrix whose diagonal blocks are computed as

$$\begin{aligned} \Delta_1 &= D_1, \\ \Delta_i &= D_i - A_i\,\text{approx}(\Delta_{i-1}^{-1})A_i^T, \end{aligned}$$

where $\text{approx}(\Delta_{i-1}^{-1})$ is a sparse approximation of Δ_{i-1}^{-1}. This preconditioner was first proposed in a slightly different form by R. Underwood [1080] and then generalized by P. Concus, G.H. Golub, and G.M. [273]. There are many ways to define $\text{approx}(\Delta_{i-1}^{-1})$, see [273]. One of the most efficient method, named INV, just considers tridiagonal approximations of the inverses. We denote by $\text{tridiag}(B)$ the tridiagonal matrix whose nonzero entries are the same as the corresponding ones of B. Then, in INV Δ_i is defined as

$$\Delta_i = D_i - A_i\text{tridiag}(\Delta_{i-1}^{-1})A_i^T.$$

This choice was motivated by the case where the D_i's are tridiagonal. If A is a diagonally dominant L-matrix, then it is easy to show that the entries in a row of the inverses of the Δ_i's decay away from the diagonal. Therefore, the absolute values of the entries that are thrown away are smaller than the ones that are kept. One can also define a modified variant MINV for which the remainder R has zero rowsums.

The idea of block incomplete Cholesky or LU factorizations which was developed for block tridiagonal matrices can be straightforwardly generalized to any block structure with square diagonal blocks. This was proposed by O. Axelsson [61]. He also obtained bounds for the eigenvalues of $M^{-1}A$, see [64, 70].

7.4 ▪ Approximate inverses

Approximate inverse preconditioners directly construct the matrix M^{-1}, the goal being to have $M^{-1}A$ close to the identity matrix in a certain sense. Having M^{-1}, solving $Mz = r$ is done by

a matrix-vector product which is parallelizable. As for incomplete factorizations, there are many ways to construct approximate inverses.

We may try to compute $C = M^{-1}$ such that some norm of $AC - I$ or $CA - I$ is minimized. The Frobenius norm is particularly appealing because we have

$$\|AC - I\|_F^2 = \sum_{k=1}^{n} \|(AC - I)e_k\|^2.$$

We just have to minimize the ℓ_2 norms $\|Ac_k - e_k\|$, $k = 1, \ldots, n$ where the c_k's are the columns of C. We observe that the n least squares problems are independent of each other. Since we want to compute a sparse C, we have to choose a given sparsity pattern for the vectors c_k. We can just consider a vector $\hat{c}_k$ which contains the nonzero components of c_k. Let $\hat{A}_k$ be the matrix whose columns are the columns of A corresponding to the set of indices $G_k = \{j | (c_k)_j \neq 0\}$ and whose rows i are such that there exists $a_{i,j} \neq 0, j \in G_k$. Everything reduces to a set of (small) least squares problems,

$$\min_{\hat{c}_k} \|\hat{A}_k \hat{c}_k - e_k\|, \quad k = 1, \ldots, n.$$

Since $\hat{A}$ is of full rank, we could solve the least squares problems with the QR factorization, see, for instance, G.H. Golub and C.F. Van Loan [547, 548].

Different approaches can be used to choose the set of indices G of the nonzero entries. T. Huckle and M. Grote [657] proposed an incremental method for choosing the sparsity patterns. Starting from a set of indices G_k^0, usually corresponding to a diagonal matrix C or to the nonzero pattern of A, one solves the least squares problems and then, iteratively, enlarges the sets of indices and solve again the least squares problems until some criteria are satisfied.

Assume we are at iteration p. To extend the set of indices, we consider the residual $r = Ac_k^p - e_k$. We would like to reduce $\|r\|$. Let $\mathcal{L} = \{j | (r)_j \neq 0\}$ and $\forall \ell \in \mathcal{L}$ let $\mathcal{N}_\ell = \{j | a_{\ell,j} \neq 0, j \notin G_k^p\}$. The candidates for indices of new nonzero entries in the solution vector are chosen in $\cup_{\ell \in \mathcal{L}} \mathcal{N}_\ell$. For j in this set of indices, we consider solving the problem

$$\min_{\mu_j \in \mathbb{R}} \|r + \mu_j A e_j\| \quad \Longrightarrow \quad \mu_j = -\frac{(r, Ae_j)}{\|Ae_j\|^2}.$$

The norm of the new residual is

$$\|r\|^2 - \frac{(r, Ae_j)^2}{\|Ae_j\|^2}.$$

There exist indices such that $(r, Ae_j) \neq 0$. We choose those which give the smallest residuals. This process is repeated until the norm of the residual is smaller than a prescribed criterion or until we have reached the maximum storage we have allowed for that column. Adding new indices to the solution vector will add some columns and some rows to the matrices of the least squares problems and we have to update the QR factorization. There are efficient techniques for doing that, see [547, 548].

Bounds for some norms of $AC - I$ and also sufficient conditions for C being nonsingular were given by T. Huckle and M. Grote. This implementation is known as SPAI; for details, see [657, 658, 656, 338] and also [254]. A drawback of this algorithm is that there is no guarantee that M^{-1} is symmetric when A is symmetric.

Some improvements to the preceding algorithm were proposed by N.I. Gould and J.A. Scott [553]. Let $\hat{A}$ be the matrix with the compressed set of column indices. If we add a new column c, the new least squares problem is

$$\sigma_{+c} = \min_{z,\xi} \|\hat{A}z + \xi c - d\|^2.$$

In [657] the choice of the new column is based on the solution of the minimization of $\|\xi c - r\|^2$ where r is the residual $\hat{A}z - d$. But the solution σ_{+c} is known as

$$\sigma_{+c} = \sigma - \frac{(c, P_{\hat{A}} d)^2}{\|P_{\hat{A}} c\|^2} = \sigma - \frac{(c, r)^2}{\|P_{\hat{A}} c\|^2},$$

where σ is the previous solution and $P_{\hat{A}} = I - \hat{A}(\hat{A}^T \hat{A})^{-1} \hat{A}^T$. This can be computed from the QR factorization of $\hat{A}$. It turns out that the value of $\|P_{\hat{A}} c\|^2$ can be easily updated. Then, the choice of the updating column is based on σ_{+c}. Numerical experiments and numerical comparisons with ILU(0) are given in [553].

To address the problem of nonsymmetry of the approximate inverses, preconditioners of the form $M^{-1} = GG^T$ were described by L.Y. Kolotilina and A.Y. Yeremin [710]. This preconditioner is known as FSAI. Factored approximate inverse preconditioners and their relations with ILU factorizations were also considered by M. Bollhöfer and Y. Saad [139, 140].

E. Chow and Y. Saad proposed a few methods for computing sparse inverses for general sparse matrices [256]. One method is to compute a crude approximation of the solution of $Ac_j = e_j$ with an iterative method. Some of the entries of this crude solution are then dropped to preserve a given sparsity pattern of the approximate inverse.

As we said in Chapter 2, an inverse factorization algorithm was used by M. Benzi, C.D. Meyer, and M. Tůma [117] to construct preconditioners for SPD matrices. If $Z = [z_1, z_2, \ldots, z_n]$ is a set of conjugate directions for A, $D = Z^T AZ$ is a diagonal matrix with diagonal entries $d_i = (z_i, Az_i)$. It gives an expression for the inverse $A^{-1} = ZD^{-1}Z^T$. A set of conjugate directions z_i can be constructed by a Gram-Schmidt orthogonalization algorithm applied to a set of linearly independent vectors $v_1, v_2, \ldots, v_n$. If we choose $V = [v_1, v_2, \ldots, v_n] = I$, Z is upper triangular. The algorithm in [117] is the following,

1) $z_i^{(0)} = e_i, \quad i = 1, \ldots, n$

2) for $i = 1, \ldots, n \quad d_j^{(i-1)} = (a_i, z_j^{(i-1)}), \quad j = i, \ldots, n$ where a_i is the ith column (or row) of A

if $j \neq n$, $z_j^{(i)} = z_j^{(i-1)} - \left(\frac{d_j^{(i-1)}}{d_i^{(i-1)}} \right) z_i^{(i-1)}, \quad j = i+1, \ldots, n$

3) $z_i = z_i^{(i-1)}, d_i = d_i^{(i-1)}, \quad i = 1, \ldots, n$

To obtain a sparse approximate inverse we have to drop some fill-ins (in Z) outside some prescribed positions or the fill-ins whose absolute values are below a given drop tolerance. It can be also necessary to control the amount of storage used. This method is known as AINV.

It has been shown that AINV is feasible for H-matrices. Numerical examples are given in [117] comparing AINV, dropping fill-ins by values, to IC factorization using the same strategy. A variant, named SAINV, which is applicable to any SPD matrix, was developed by M. Benzi, J.K. Cullum, and M. Tůma [114].

A code for computing the AINV preconditioner for a symmetric matrix is following. It discards entries according to their absolute values and keep only the q largest in a row.

```
function [Z,d] = AINV_temp(A,epss,q)
n = size(A,1);
Z = speye(n);
d = zeros(n,1);
anorm = epss * max(abs(A'))'; % for dropping

for i=1:n-1
 xold = Z(:,i);
```

```
 xold(i) = 0;
 x = abs(Z(1:i-1,i));
 if size(x,1) ~= 0
  ind = find(x(1:i-1) < anorm(1:i-1) & x(1:i-1) > 0);
  Z(ind,i) = 0;
  % keep only the q largest entries
  [x,indx] = sort(abs(Z(1:i-1,i)));
  qq = min(q,i-1);
  indl = indx(1:i-1-qq);
  Z(indl,i) = 0;
  x = abs(Z(:,i));
  if nnz(x) == 1
   % there is just a diagonal entry
   % keep also the largest non diagonal entry
   [~,iold] = max(abs(xold));
   Z(iold(1),i) = xold(iold(1));
  end
 end
 d(i:n) = A(i,:) * Z(:,i:n);
 d1 = 1 / d(i);
 indp = find(abs(d(i+1:n)) > 0) + i;
 for j=indp'
  Z(:,j) = Z(:,j) - d(j) * d1 * Z(:,i);
 end % for j
end % for i
% last column
xold = Z(:,n);
xold(n) = 0;
for j=1:n-1
 x = abs(Z(j,n));
 if x < anorm(j) && x > 0
  Z(j,n) = 0;
 end % if
end % for j
[~,indx] = sort(abs(Z(1:n-1,n)));
qq = min(q,n-1);
indl = indx(1:n-1-qq);
Z(indl,n) = 0;
x = abs(Z(:,n));
if nnz(x) == 1
 % keep the largest non diagonal entries
 [~,iold] = max(abs(xold));
 Z(iold(1),n) = xold(iold(1));
end % if
d(n) = A(n,:) * Z(:,n);
```

AINV was generalized to nonsymmetric matrices by M. Benzi and M. Tůma [120]. Two sets of vectors $Z = [z_1, \ldots, z_n]$ and $W = [w_1, \ldots, w_n]$ are constructed such that the matrix $D = W^T AZ$ is diagonal. The matrices W and Z are constructed by a biconjugation process applied to $W^{(0)} = Z^{(0)} = I$. The formulas are quite similar to the ones for the symmetric case.

Numerical experiments comparing SPAI and AINV are described in [121]; see also [119, 122, 711, 712, 1002]. A multilevel AINV algorithm was proposed by G.M. [813, 814].

7.5 ▪ Polynomial preconditioners

Another idea to directly construct M^{-1} is to use a polynomial,

$$M^{-1} = P_k(A) = \sum_{j=0}^{k} \alpha_j A^j.$$

In fact, this was the first ever constructed preconditioner to accelerate the convergence of an iterative method by L. Cesari [211, 212, 213, 214] in 1937. We observe that, by the Cayley-Hamilton theorem, the inverse A^{-1} is a polynomial in A. But, the degree is generally large and its coefficients are unknown.

If the matrix A is SPD, it may be strange to use a polynomial preconditioner with PCG. We have seen in Chapter 5 that the polynomial implicity generated by CG is optimal in a certain sense. Applying m CG iterations to $P_k(A)A$ will generate a polynomial of degree $k+m$ that will be less efficient than $k+m$ iterations of CG without preconditioning for reducing the A-norm of the error. However, using a polynomial preconditoner, there will be less dot products and this could be beneficial on parallel computers.

One of the simplest way of constructing a polynomial preconditioner is to use Neumann series. Let A be a SPD matrix that we write as $A = D - L - L^T$ where D is a diagonal matrix, the minus signs being just a technical convenience. We symmetrically scale A by its diagonal,

$$A = D^{\frac{1}{2}}(I - D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}})D^{\frac{1}{2}},$$

$$A^{-1} = D^{-\frac{1}{2}}(I - D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}})^{-1}D^{-\frac{1}{2}}.$$

We observe that

$$D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}},$$

and

$$\rho(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) = \rho(I - D^{-1}A).$$

If the spectral radius satisfies $\rho(I - D^{-1}A) < 1$, we can expand $I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ in a Neumann series and only use a few terms of the series for the preconditioner. Generally, a small number of terms is used. For instance, the first order Neumann polynomial is

$$M^{-1} = D^{-1} + D^{-1}(L + L^T)D^{-1} = 2D^{-1} - D^{-1}AD^{-1}.$$

Generally, polynomials of even degree are not used because it was shown by P.F. Dubois, A. Greenbaum, and G.H. Rodrigue [367] that a Neumann polynomial of odd degree k is more efficient than the Neumann polynomial of degree $k+1$.

Since we have $M^{-1}A = P_k(A)A$, the polynomial q_k which is involved is $q_{k+1}(\lambda) = \lambda P_k(\lambda)$. It satisfies $q_k(0) = 0$. The ideal situation for PCG would be to have $q_k(\lambda) \equiv 1$, $\forall \lambda$. However, this is not possible because of the value at the origin.

O.G. Johnson, C.A. Michelli, and G. Paul [684] defined a generalized condition number. Let a and b such that the eigenvalues λ_i of A are in $[a, b], \forall i$. Let

$$cond(q) = \frac{\max_{\lambda \in [a,b]} q(\lambda)}{\min_{\lambda \in [a,b]} q(\lambda)},$$

with $\mathcal{Q}_k = \{\text{polynomials } q_k | \forall \lambda \in [a,b], q_k(\lambda) > 0, q_k(0) = 0\}$. The first constraint gives a positive definite preconditioner. We look for the solution of the problem

$$\text{Find } q_k \in \mathcal{Q}_k \text{ such that } \forall q \in \mathcal{Q}_k, cond(q_k) \leq cond(q).$$

The solution of this problem was given in [684], see also S.F. Ashby [46].

Theorem 7.9. *The solution of the previous minimization problem is*

$$q_k(\lambda) = 1 - \frac{C_k(\mu(\lambda))}{C_k(\mu(0))},$$

where C_k is the Chebyshev polynomial of order k and

$$\mu(\lambda) = \frac{2\lambda - b - a}{b - a}.$$

Proof. Let $\theta = |C_k(\mu(0))|$. Then,

$$cond(q_k) = \frac{\theta + 1}{\theta - 1},$$

since we have

$$cond(q_k) = \frac{\max_{\lambda \in [a,b]} \{C_k(\mu(0)) - C_k(\mu(\lambda))\}}{\min_{\lambda \in [a,b]} \{C_k(\mu(0)) - C_k(\mu(\lambda))\}} = \frac{C_k(\mu(0)) + 1}{C_k(\mu(0)) - 1},$$

because $\min C_k = -1$ and $\max C_k = 1$. Moreover, $|C_k(\mu(\lambda))| \leq 1$ on $[a,b]$ and $C_k(\mu(0)) > 1$. Let $q \in \mathcal{Q}_k$ and $v = \min_{\lambda \in [a,b]} q(\lambda)$, $V = \max_{\lambda \in [a,b]} q(\lambda)$. We would like to show that

$$\frac{\theta + 1}{\theta - 1} \leq \frac{V}{v},$$

or equivalently,

$$\frac{V + v}{V - v} \leq \theta.$$

Outside $[-1, 1]$ the Chebyshev polynomials are those with the fastest increase. Let u_k be a polynomial of degree k. Then, for $\mu(\lambda) \geq 1$,

$$\frac{|u_k(\mu(\lambda))|}{\max_{|t| \leq 1} |u(t)|} \leq |C_k(\mu(\lambda))|.$$

We use this result with $u_k(\mu(\lambda)) = 1 - \frac{2q(\lambda)}{V+v}$. Clearly,

$$\max_{|t| \leq 1} |u_k(\lambda)| = \frac{V - v}{V + v}.$$

Therefore,

$$\frac{V + v}{V - v} \leq |C_k(\mu(0))|.$$

□

Theorem 7.10. *Let $\kappa = b/a$ and $\nu = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$. Then, $cond(q_k) = \left(\frac{1+\nu^k}{1-\nu^k}\right)^2$.*

Proof. See O. Perlot [914]. □

A frequently used technique is to use a least squares polynomial, that is, to look for the polynomial p_k of degree k that minimizes

$$\int_a^b (1 - \lambda q(\lambda))^2 w(\lambda)\, d\lambda, \quad q \in \mathcal{Q}_k, \tag{7.7}$$

where $w(\lambda)$ is a positive weight, see Y. Saad [973]. Usually, one chooses the Jacobi weights,

$$w(\lambda) = (b - \lambda)^\alpha (\lambda - a)^\beta, \quad \alpha \geq \beta \geq -\frac{1}{2},$$

because the orthogonal polynomials associated with these weights are known. Let $s_i(\lambda)$ be this normalized orthogonal polynomial and

$$J_k(\sigma, \lambda) = \sum_{j=0}^{k} s_j(\sigma) s_j(\lambda).$$

To derive the solution of this problem, let us prove two technical results.

Lemma 7.11. *Let $r(\lambda)$ be a polynomial of degree $\leq k$. Then,*

$$1 - r(\sigma) = \int_a^b J_k(\sigma, \lambda)(1 - r(\lambda)) w(\lambda)\, d\lambda.$$

Proof. We can write $1 - r(\lambda) = \sum_l \alpha_l s_l(\lambda)$. Then,

$$\begin{aligned}
&\int_a^b J_k(\sigma, \lambda)(1 - r(\lambda)) w(\lambda)\, d\lambda \\
&\qquad = \sum_j s_j(\sigma) \sum_l \int_a^b s_j(\lambda) s_l(\lambda) \alpha_l w(\lambda)\, d\lambda, \\
&\qquad = \sum_j s_j(\sigma) \sum_l \alpha_l \int_a^b s_j(\lambda) s_l(\lambda) w(\lambda)\, d\lambda, \\
&\qquad = \sum_j \alpha_j s_j(\sigma) = 1 - r(\sigma).
\end{aligned}$$

□

Lemma 7.11 explains why $J_k(\sigma, \lambda)$ is called a reproducing kernel. The next result gives a lower bound for $J_k(0, 0)$.

Lemma 7.12. *Let $r(\lambda)$ be a polynomial of degree $\leq k$ such that $r(0) = 0$. Then*

$$1 \leq J_k(0, 0) \int_a^b (1 - r(\lambda))^2 w(\lambda)\, d\lambda.$$

Proof. Using the Cauchy-Schwarz inequality,

$$\begin{aligned}
1 = (1 - r(0))^2 &= \int_a^b J_k(0, \lambda)(1 - r(\lambda)) w(\lambda)\, d\lambda, \\
&\leq \int_a^b J_k(0, \lambda)^2 w(\lambda)\, d\lambda \int_a^b (1 - r(\lambda))^2 w(\lambda)\, d\lambda.
\end{aligned}$$

But,

$$\int_a^b J_k(0,\lambda)^2 w(\lambda)\,d\lambda = \sum_j s_j(0)^2 = J_k(0,0).$$

□

Theorem 7.13. *The solution of the least squares minimization problem (7.7) is*

$$\lambda q(\lambda) = r_k(\lambda) = 1 - \frac{J_k(0,\lambda)}{J_k(0,0)}.$$

Proof. The polynomial $1 - r_k(\lambda)$ is collinear to $J_k(0,\lambda)$ and equal to 1 in $\lambda = 0$. Therefore, we have equality in the Cauchy-Schwarz inequality,

$$\frac{1}{J_k(0,0)} = \int_a^b (1 - r_k(\lambda))^2 w(\lambda)\,d\lambda,$$

which is the minimum value as Lemma 7.12 shows. □

The solution of (7.7) can be rewritten as

$$q_k(\lambda) = \sum_{j=0}^{k+1} b_j t_j(\lambda),$$

where

$$b_j = \frac{s_j(0)}{\sum_{l=0}^{k+1} s_l(0)^2}, \quad t_j(\lambda) = \frac{s_j(0) - s_j(\lambda)}{\lambda}.$$

Generally, one uses the Chebyshev weight corresponding to $\alpha = \beta = -\frac{1}{2}$.

Evaluating the matrix polynomial applied to a given vector can be done with the Horner's scheme. However, a more specific method can be developed using a generalization of Clenshaw's formula.

Lemma 7.14. *Let*

$$\mu(\lambda) = \frac{2\lambda - (a+b)}{b-a},$$

and

$$s_{j+1}(\lambda) = \alpha_j \mu(\lambda) s_j(\lambda) - \gamma_{j-1} s_{j-1}(\lambda), \quad t_j(\lambda) = \frac{s_j(\lambda) - s_j(0)}{\lambda}.$$

Then,

$$t_{j+1}(\lambda) = \delta_{j+1} + \alpha_j \mu(\lambda) t_j(\lambda) - \gamma_{j-1} t_{j-1}(\lambda),\ \forall j \geq 1,$$

with $\delta_{j+1} = K\alpha_j s_j(0),\ K = \frac{\mu(0)-\nu(\lambda)}{\lambda}$.

Proof. We have

$$\begin{aligned} t_{j+1}(\lambda) &= \frac{s_{j+1}(0) - s_{j+1}(\lambda)}{\lambda}, \\ &= \frac{1}{\lambda}(\alpha_j \mu(0) s_j(0) - \gamma_{j-1} s_{j-1}(0) \\ &\quad -\alpha_j \mu(\lambda) s_j(\lambda) + \gamma_{j-1} s_{j-1}(\lambda)), \end{aligned}$$

$$
\begin{aligned}
&= \alpha_j \frac{\mu(0) - \mu(\lambda)}{\lambda} + \alpha_j \mu(\lambda) \frac{s_j(0) - s_j(\lambda)}{\lambda} \\
&\quad -\gamma_{j-1} \frac{s_{j-1}(0) - s_{j-1}(\lambda)}{\lambda}, \\
&= K\alpha_j s_j(0) + \alpha_j \mu(\lambda) t_j(\lambda) - \gamma_{j-1} t_{j-1}(\lambda).
\end{aligned}
$$

□

Theorem 7.15. *For the least squares polynomial we have*

$$p_k(\lambda) = t_1(\lambda)\eta_1(\lambda) + \omega_2(\lambda),$$

where the η_j's are defined by

$$\eta_j(\lambda) = b_j + \alpha_j \mu(\lambda)\eta_{j+1}(\lambda) - \gamma_j \eta_{j+2}(\lambda), \quad j = k+1, \ldots, 1$$

$\eta_{k+3} = \eta_{k+2} \equiv 0$ *and ω_j is defined by*

$$\omega_j(\lambda) = \delta_j \eta_j(\lambda) + \omega_{j+1}(\lambda), \quad j = k+1, \ldots, 2, \quad \omega_{k+2} \equiv 0.$$

Proof. Let $A_k(\lambda) = \sum_{j=0}^{k} b_j t_j(\lambda)$ and let H_ℓ denote the hypothesis that

$$p_k(\lambda) = A_{k-\ell}(\lambda) + \omega_{k-\ell+2}(\lambda) + \eta_{k-\ell+1}(\lambda) t_{k-\ell+1}(\lambda) - \gamma_{k-\ell}\eta_{k-\ell+2}(\lambda) t_{k-\ell}(\lambda).$$

We know that $p_k(\lambda) = A_{k+1}(\lambda)$, therefore H_{-1} holds. Let us assume that H_ℓ is true, then let us show that $H_{\ell+1}$ holds.

$$
\begin{aligned}
p_k(\lambda) &= A_{k-\ell-1}(\lambda) + b_{k-\ell} t_{k-\ell}(\lambda) + \omega_{k-\ell+2}(\lambda) \\
&\quad + \eta_{k-\ell+1}(\lambda) t_{k-\ell+1}(\lambda) - \gamma_{k-\ell}\eta_{k-\ell+2}(\lambda) t_{k-\ell}(\lambda), \\
&= A_{k-\ell-1}(\lambda) + b_{k-\ell} t_{k-\ell}(\lambda) + \omega_{k-\ell+2}(\lambda) - \gamma_{k-\ell}\eta_{k-\ell+2}(\lambda) \\
&\quad + \eta_{k-\ell+1}(\lambda)(\delta_{k-\ell+1}\alpha_{k-\ell}\mu(\lambda) t_{k-\ell}(\lambda) - \gamma_{k-\ell-1} t_{k-\ell-1}(\lambda)), \\
&= A_{k-\ell-1}(\lambda) + (b_{k-\ell} + \alpha_{k-\ell}\mu(\lambda)\eta_{k-\ell+1}(\lambda) \\
&\quad - \gamma_{k-\ell}\eta_{k-\ell+2}(\lambda)) t_{k-\ell}(\lambda) + \omega_{k-\ell+2}(\lambda) + \eta_{k-\ell+1}(\lambda)\delta_{k-\ell+1} \\
&\quad - \gamma_{k-\ell-1}\eta_{k-\ell+1}(\lambda) t_{k-\ell-1}(\lambda) \\
&\quad\quad A_{k-\ell-1}(\lambda) + \eta_{k-\ell}(\lambda) t_{k-\ell}(\lambda) + \omega_{k-\ell+1}(\lambda) \\
&\quad - \gamma_{k-\ell-1}\eta_{k-\ell+1}(\lambda) t_{k-\ell-1}(\lambda).
\end{aligned}
$$

Finally, using $\ell = k$, we have

$$p_k(\lambda) = A_0(\lambda) + \omega_2(\lambda) + \eta_1(\lambda) t_1(\lambda) - \gamma_0 \eta_2(\lambda) t_0(\lambda) = \omega_2(\lambda) + \eta_1(\lambda) t_1(\lambda).$$

□

Theorem 7.16. *For the least squares polynomial, $z = P_k(A)r$ can be computed as follows. Let*

$$s_0(0) = \frac{1}{\sqrt{\pi}}, \quad s_1(0) = \sqrt{\frac{2}{\pi}} \frac{a+b}{a-b}, \quad s_2(0) = \sqrt{\frac{2}{\pi}} \left[2 \left(\frac{a+b}{a-b} \right)^2 - 1 \right],$$

and

$$s_j(0) = 2\mu(0) s_{j-1}(0) - s_{j-2}(0), \quad j = 3, \ldots, k+1$$

$$b_j = \frac{s_j(0)}{\sum_{i=0}^{k+1} s_i^2(0)}, \quad j = 1, \ldots, k+1.$$

Then,

$$z_{k+1} = b_{k+1} r, \quad z_k = b_k r + \frac{2}{b-a}(2A - (a+b)I) z_{k+1},$$

$$z_j = b_j r + \frac{2}{b-a}(2A - (a+b)I) z_{j+1} - z_{j+2}, \quad j = k-1, \ldots, 1$$

and

$$u_{k+1} = \frac{4}{a-b} s_k(0) z_{k+1},$$

$$u_{j+1} = \frac{4}{a-b} s_j(0) z_{j+1} + u_{j+2}, \quad j = k-1, \ldots, 1$$

Finally,

$$z = \sqrt{\frac{2}{\pi}} \frac{2}{a-b} z_1 + u_2.$$

□

The two following functions compute the coefficients $s_j(0)$ and evaluate $P_k(A)r$ for A symmetric positive definite. If they are not given $\lambda_{\min}$ and $\lambda_{\max}$ (denoted as a and b above) are computed using Gerschgorin bounds.

```
[s,lmin,lmax] = poly_ls(A,k,lmin,lmax)
k = abs(k);
if nargin == 1
 k = 1;
end % if
if nargin < 3
 % compute the Gerschgorin bounds
 n = size(A,1);
 un = ones(n,1);
 AA = abs(A);
 DA = spdiags(diag(AA),0,n,n);
 % put zeros on the diagonal
 AA = spdiags(zeros(n,1),0,AA);
 b = (DA - AA) * un;
 bb = (DA + AA) * un;
 lmin = max(min(b),0);
 lmax = max(bb);
end % if
s = zeros(k+2,1);
mu0 = -(lmin + lmax) / (lmax - lmin);
s(1) = 1 / sqrt(pi);
s(2) = sqrt(2 / pi) * (lmin + lmax) / (lmin - lmax);
s(3) = sqrt(2 / pi) * ((2 * (lmin + lmax)^2 / (lmin - lmax)^2)
    - 1);
for j = 4:k+2
 s(j) = 2 * mu0 * s(j-1) - s(j-2);
end % for j
```

```
function z = eval_poly_ls(r,A,s,lmin,lmax);
k = length(s) - 2;
ss = sum(s(1:k+2).^2);
b = s / ss; % normalization
z(:,k+1) = b(k+2) * r;
z(:,k) = b(k+1) * r + (2 / (lmax - lmin)) * (2 * A * z(:,k+1) -
    (lmax + lmin) * z(:,k+1));
for j = k-1:-1:1
 z(:,j) = b(j+1) * r + (2 / (lmax - lmin)) * (2 * A * z(:,j+1)
    - (lmax + lmin) * z(:,j+1)) - z(:,j+2);
end
u = (4 / (lmin - lmax)) * s(k+1) * z(:,k+1);
for j = k-1:-1:1
 u = (4 / (lmin - lmax)) * s(j+1) * z(:,j+1) + u;
end
z = sqrt(2 / pi) * (2 /(lmin - lmax)) * z(:,1) + u;
```

A drawback of the min-max and least squares polynomials is that they need estimates of the smallest and largest eigenvalues of A. The least squares polynomial is rather insensitive to the choice of a and $a = 0$ can also be chosen since the polynomial stays positive definite. When $a = 0$, it can be computed in a simpler way. Let $d_0 = 1, d_1 = \frac{3}{2}$ and

$$d_{j+1} = \frac{2j+1}{j+1} d_j - \frac{(j+1/2)(j-1/2)}{j(j+1)} d_{j-1}, \quad j = 1, \ldots, k.$$

The solution of $z = M^{-1}r$ is computed as

$$z_0 = \frac{2}{3} r, \quad z_1 = -\frac{4}{5} Ar + 2r,$$

$$z_j = \frac{2j+1}{j+1} \frac{d_j}{d_{j+1}} (r + (I - A) z_{j-1}) - \frac{(j^2 - 1/4)}{j(j+1)} \frac{d_{j-1}}{d_{j+1}} z_{j-2}, \quad j = 2, \ldots, k.$$

Finally, $z = z_k$.

If the matrix A is symmetric but indefinite, it is more difficult to find a "good" polynomial. There are cases where the optimal polynomial is explicitly known, see B. Fischer [455]. For the general case, C. de Boor and J.R. Rice [322] formulated a Remez-type algorithm for the numerical computation of the optimal polynomial. The classical Remez algorithm is an iterative technique for computing the minimax polynomial approximation to a real function f on a compact set S. It can also be modified to solve a constrained minimization problem. A detailed description is given by S.F. Ashby [45].

The problem of constructing a polynomial preconditioner is even more difficult when the matrix A is nonsymmetric. Since the eigenvalues of A are complex (conjugate) numbers, we have first to locate these eigenvalues and to find a region S of the complex plane that contains them. This problem was considered by T.A. Manteuffel [779, 780] who suggested computing an ellipse (not containing the origin) that contains the convex hull of the spectrum. Then, the min-max polynomial can (sometimes) be expressed in terms of Chebyshev polynomials.

H.C. Elman, Y. Saad, and P.E. Saylor [419] also used an ellipse enclosing the Ritz values obtained from the Arnoldi process and Chebyshev polynomials. H.C. Elman and R.L. Streit [420] considered the union of convex hulls of subsets of the Ritz values and an L_∞ optimal approximation polynomial. D.C. Smolarski and P.E. Saylor [992] enclosed the Ritz values in a polygon and

used a least squares polynomial. This was previously used by Y. Saad [974], but his least squares polynomial was computed in a different and more stable way using modified moments. On that topic, see also the Ph.D. thesis of S.F. Ashby [45], as well as [47] by S.F. Ashby, T.A. Manteuffel, and J.S. Otto.

7.6 ▪ Domain decomposition

Domain decomposition (DD) is a natural framework to develop solution methods for solving problems on parallel computers. Although the idea is quite old and it has been used for many years, mainly in structural mechanics (see [490]), the interest in domain decomposition was renewed at the end of the 1980s. DD methods were initially proposed for solving partial differential equations on a domain which is partitioned into subdomains. Subproblems are solved on each subdomain and their solutions are glued together to obtain the global solution. DD methods can be considered for continuous problems or for algebraic linear systems.

It is generally admitted that the first ever DD method was due to Hermann A. Schwarz (1843-1921) in August 1870. It was described in [1000]. As explained in the historical paper [491] by M.J. Gander and G. Wanner, Schwarz's goal was to prove the existence of a solution to Laplace's equation $\Delta u = 0$ with Dirichlet boundary conditions $u = g$ on complicated domains. Schwarz considered a two-dimensional domain which was the union of a disk and an overlapping rectangle, see Figure 7.1. It was known that solutions of Laplace's equation existed for the disk and the rectangle and Schwarz constructed an iterative method by solving alternatively subproblems in the disk using a Dirichlet boundary condition from the solution in the rectangle and then, in the rectangle using a Dirichlet boundary condition from the previous solution in the disk, and so on by iterating the process. He proved the convergence of the method using the maximum principle, even though his argument lacked some rigor.

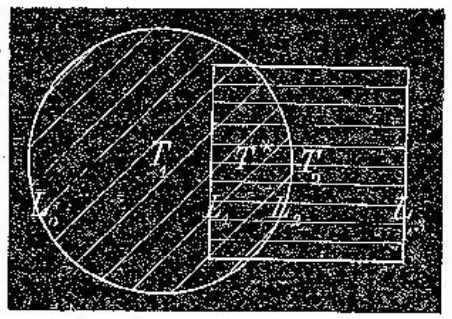


Figure 7.1. *Schwarz domain decomposition in [1000]*

Since the 1980s, thousands of papers have been written about DD. We will just describe the main ideas. We can divide DD methods into two main categories: with and without overlapping. Another important distinction between methods is the algorithm which is used for solving the subproblems. One can use either a direct method (like Gaussian elimination) or iterative methods or simply a preconditioner for the matrix of the subproblem. Combining all these possibilities gives rise to a very large number of algorithms. A good source of information on DD methods are the proceedings of the DD conferences that started in 1987 in Paris, see www.ddm.org.

We first study some methods with overlapping. They are generically known as Schwarz methods. An interesting paper about Schwarz methods and their history is [487] by M.J. Gander.

7.6.1 ▪ Schwarz methods

Let us assume that we would like to solve a second order elliptic PDE in a bounded two-dimensional domain Ω. For simplicity, we consider the domain Ω split into two overlapping subdomains Ω_1 and Ω_2. Let Γ_i, $i = 1, 2$, be the part of the boundary of Ω_i enclosed in Ω, see Figure 7.2. Let us consider an iterative method similar to what was used by Schwarz.

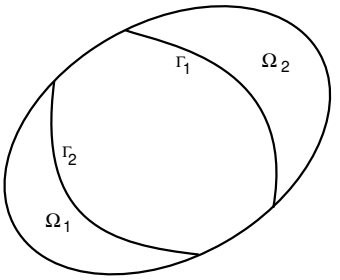


Figure 7.2. *Overlapping subdomains*

The rate of convergence of the Schwarz method depends on the extent of overlapping. The larger the overlapping, the faster the convergence. If $\Omega_1 = \Omega_2 = \Omega$, the method converges in one iteration. However, when the overlapping is larger, the cost of solving the subproblems is higher. It is assumed that the subproblems are solved exactly. Therefore, there is a trade-off between the number of iterations and the cost of solving the subproblems. A general convergence theory for PDE problems was provided by P.L. Lions [751, 752, 753] in terms of projections in Hilbert spaces. Let us follow [751]. We consider the Poisson model problem, but the results hold true for other bilinear forms. Let the problem be

$$-\Delta u = f \ \text{ in } \Omega, \quad u|_{\partial\Omega} = 0.$$

The Schwarz alternating method can be formulated at the continuous level, when u_1 is given, as

$$-\Delta u_{2k} = f \ \text{ in } \Omega_1, \quad u_{2k}|_{\Gamma_1} = u_{2k-1}|_{\Gamma_1},$$

$$-\Delta u_{2k+1} = f \ \text{ in } \Omega_2, \quad u_{2k+1}|_{\Gamma_2} = u_{2k}|_{\Gamma_2},$$

for $k = 1, 2, \ldots$ with the given boundary conditions on the other parts of the boundary. The bilinear form a of the problem is defined as

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dx.$$

The Poisson model problem can be written in variational form,

$$a(u, v) = (f, v), \quad \forall v \in H_0^1(\Omega).$$

Let $V_1 = H_0^1(\Omega_1)$ and $V_2 = H_0^1(\Omega_2)$ and the projectors P_1 and P_2 defined by

$$a(P_i v, w) = a(v, w), \ \forall w \in V_i, \ i = 1, 2.$$

The functions defined only on a subdomain are extended by 0 to $H_0^1(\Omega)$. Then, we have

$$a(u_{2k} - u, v_1) = 0, \ \ \forall v_1 \in V_1, \ \ u_{2k} - u_{2k-1} \in V_1,$$

$$a(u_{2k+1} - u, v_2) = 0, \ \ \forall v_2 \in V_2, \ \ u_{2k+1} - u_{2k} \in V_2.$$

It yields

$$u - u_{2k} = (I - P_1)(u - u_{2k-1}),$$
$$u - u_{2k+1} = (I - P_2)(u - u_{2k}).$$

Therefore, by eliminating u_{2k}, we obtain

$$u - u_{2k+1} = (I - P_2)(I - P_1)(u - u_{2k-1}).$$

This relation explains why this type of algorithm is more generally known as a multiplicative Schwarz method. The mathematical problem for studying convergence is

$$v_0 \in V, \quad v_{2k} = (I - P_1)v_{2k-1}, \quad v_{2k+1} = (I - P_2)v_{2k}.$$

We have to study the convergence of iterated projections. The following result shows that the method converges.

Theorem 7.17. *If $V = \overline{V_1 + V_2}$, where the bar denotes the closure of the set, then $v^{(k)} \longrightarrow 0$. Moreover, if $V = V_1 + V_2$ then,*

$$\|(I - P_2)(I - P_1)\| \leq c < 1.$$

Proof. See P.L. Lions [751]. □

We observe that the original Schwarz method is not parallel since the subproblems are solved in sequence. It is a block Gauss-Seidel-like method with overlapping. P.L. Lions [752] proposed a parallel variant by using $u_{2k+1}|_{\Gamma_2} = u_{2k-1}|_{\Gamma_2}$. This is similar to a block Jacobi-like algorithm.

The multiplicative Schwarz method can also be formulated at the discrete level for a linear system $Ax = b$. We have to partition the unknowns in (overlapping) subsets corresponding to the subdomains. Let us assume that we have two such subsets and restriction operators R_1 and R_2. The transposes of R_1 and R_2 are prolongation (extension) operators. We define

$$A^{(1)} = R_1 A R_1^T, \quad A^{(2)} = R_2 A R_2^T.$$

In the first half step of the iteration, we restrict the residual with R_1, we apply the inverse of $R_1 A R_1^T$, and extend the result by R_1^T. This is finally written as

$$x_{2k} = x_{2k-1} + R_1^T (A^{(1)})^{-1} R_1 (b - A x_{2k-1}).$$

Similarly, the second half step is

$$x_{2k+1} = x_{2k} + R_2^T (A^{(2)})^{-1} R_2 (b - A x_{2k}).$$

Proposition 7.18. *The matrix $P_i = R_i^T (R_i A R_i^T)^{-1} R_i A, i = 1, 2$ is an orthogonal projection in the dot product defined by A.*

Proof. We have

$$P_i P_i = R_i^T (R_i A R_i^T)^{-1} R_i A R_i^T (R_i A R_i^T)^{-1} R_i A = P_i.$$

Moreover,

$$AP_i = AR_i^T(R_iAR_i^T)^{-1}R_iA = (AP_i)^T.$$

□

If ε_k is the error vector, we have

$$\varepsilon_{2k} = (I - P_1)\varepsilon_{2k-1}, \quad \varepsilon_{2k+1} = (I - P_2)\varepsilon_{2k}.$$

This could appear as a discretization of what we have seen at the continuous level. For a discussion of this point, see [487]. Without overlap, the Schwarz alternating method is just a block Gauss-Seidel method. The method can be generalized to S subdomains,

$$x_{k+\frac{s}{S}} = x_{k+\frac{s-1}{S}} + R_s^T(A^{(s)})^{-1}R_s(b - Ax_{k+\frac{s-1}{S}}), \quad s = 1, \ldots, S.$$

Let us study the convergence of the method for a one-dimensional Poisson model problem. The matrices which are involved are

$$A^{(1)} = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

of order $p - 1$ and $A^{(2)}$ which is the same matrix but of order $n - \ell$. We explicitly know the inverses of these matrices.

Proposition 7.19. *For the one-dimensional Poisson problem, we have*

$$[\varepsilon_{2k}]_i = \frac{i}{p}[\varepsilon_{2k}]_p, \ i = 1, \ldots, p-1,$$

$$[\varepsilon_{2k+1}]_i = \frac{n-i+1}{n-\ell+1}[\varepsilon_{2k+1}]_\ell, \ i = \ell+1, \ldots, n.$$

Proof. Because we use exact solves for the subproblems, the equations corresponding to the unknowns inside the subdomains are exactly satisfied and we have

$$A^{(1)} \begin{pmatrix} [\varepsilon_{2k}]_1 \\ \vdots \\ [\varepsilon_{2k}]_{p-1} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ [\varepsilon_{2k-1}]_p \end{pmatrix}.$$

The components 1 to $p - 1$ of the error vector at iteration $2k$ are the components of the last column of the inverse of $A^{(1)}$ times $[\varepsilon_{2k-1}]_p$. Since the inverse is known explicitly, we obtain the result. The proof is the same for the other relation. We observe that $[\varepsilon_{2k}]_p = [\varepsilon_{2k-1}]_p$ and $[\varepsilon_{2k+1}]_\ell = [\varepsilon_{2k}]_\ell$. □

Then, it is clear how the error is reduced during the iterations. At the end of the first half step, the error is maximum for the node p and linear (being 0 at the ends of the interval). At the end of the second half step, the error is maximum for the node ℓ and linear, see Figure 7.3 where the errors for three half steps are shown for a problem of order 30. The left subdomain uses mesh points 1 to 20 and the right one points 10 to 30. The error in the first half step is the solid line.

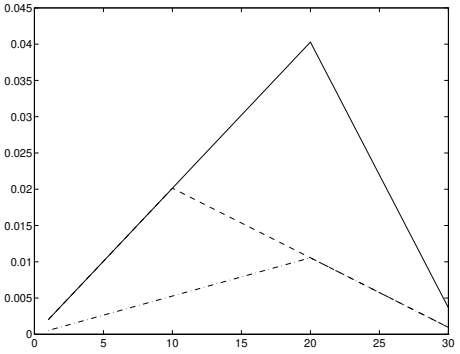


Figure 7.3. *Errors for three half steps of the Schwarz method*

The error in the second half step is the dashed line (partially hidden by the others). The error in the third half step is the dash-dotted line.

Theorem 7.20. *At odd steps, the maximum of (the absolute value of) the error is obtained for node ℓ and*

$$\|\varepsilon_{2k+1}\|_\infty = \frac{\ell}{p}\frac{n-p+1}{n-\ell+1}\|\varepsilon_{2k-1}\|_\infty.$$

Proof. The result is obvious from the previous discussion. We note that both multiplying factors are less than 1 since

$$\frac{n-p+1}{n-\ell+1} = 1 - \frac{p-\ell}{n-\ell+1}.$$

□

The previous theorem shows that the larger the overlap ($p-\ell$), the faster the convergence. The same analysis can be done for this problem with a larger number of subdomains since the error is still linear on each subdomain. Unfortunately, the rate of convergence is slower when we have a large number of subdomains as shown by Figure 7.4 where we plot the number of iterations for two and three subdomains as a function of the extent of overlap (number of points in the overlapping region) for the one-dimensional Poisson model problem. When we increase the extent of overlapping we have a large decrease in the number of iterations at first and then the numbers of iterations level off meaning that it is only marginally better to have a large overlap.

Like Gauss-Seidel, the multiplicative Schwarz method is sequential and the iteration matrix is nonsymmetric, even when A is symmetric. A way to introduce parallelism when there is more than two subdomains is to color them in such a way that two neighboring subdomains do not have the same color. Another possibility is to use the additive Schwarz method introduced by M. Dryja and O.B. Widlund [1118, 364, 365]. A similar method was introduced earlier at the continuous level by A.M. Matsokin and S.V. Nepomnyaschikh [791]. With two subdomains and the same notation as above, the system to be solved is written as

$$M_{AS}^{-1}Ax = M^{-1}b, \quad M_{AS}^{-1} = R_1^T(R_1AR_1^T)^{-1}R_1 + R_2^T(R_2AR_2^T)^{-1}R_2.$$

However, used as a stationary iterative method $x_{k+1} = x_k + M_{AS}^{-1}(b - Ax_k)$, it does not converge in the overlap, see [487]. Nevertheless, it can be used as a preconditioner for Krylov iterative methods. Since the convergence is not always fast enough, a coarse grid correction is sometimes

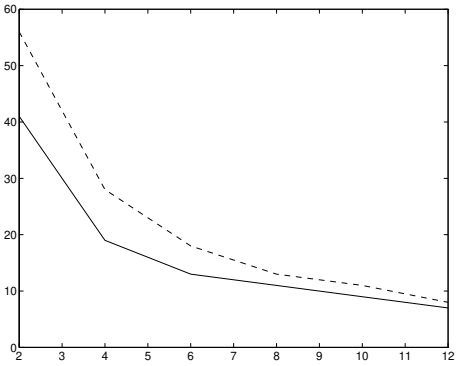


Figure 7.4. *Number of iterations as a function of the length of the overlap, two subdomains (solid), three subdomains (dashed)*

added,

$$M_{AS}^{-1} = \sum_{i=1}^{S} R_i^T [A^{(i)}]^{-1} R_i + R_0^T [A^{(0)}]^{-1} R_0.$$

Another method was introduced by X.-C. Cai and M. Sarkis [194] under the name restricted additive Schwarz method. For two subdomains,

$$M_{RAS}^{-1} = \tilde{R}_1^T (R_1 A R_1^T)^{-1} R_1 + \tilde{R}_2^T (R_2 A R_2^T)^{-1} R_2,$$

where $\tilde{R}_1$ and $\tilde{R}_2$ correspond to a non-overlapping decomposition of the domain. For a study of this method, see [480] by A. Frommer and D.B. Szyld. Unfortunately, the matrix M_{RAS}^{-1} is nonsymmetric.

A way to enhance the convergence of Schwarz methods is to change the boundary (transmission) conditions on the interfaces between subdomains. At the continuous level, P.L. Lions [753] proposed to use Robin-like boundary conditions,

$$\left(\frac{\partial}{\partial n_1} + p_1\right) u_{2k}^1 = \left(\frac{\partial}{\partial n_1} + p_1\right) u_{2k-1}^2 \text{ on } \Gamma_1,$$

$$\left(\frac{\partial}{\partial n_2} + p_2\right) u_{2k+1}^2 = \left(\frac{\partial}{\partial n_2} + p_2\right) u_{2k}^1 \text{ on } \Gamma_2,$$

where $n_j, j = 1, 2$ denotes the unit outward normal, and $p_j, j = 1, 2$ can be constants or functions defined along the interface.

This type of transmission conditions was also advocated by W.-P. Tang [1057]. More general operators can be used for the transmission conditions and a lot of research has been done on that topic, see, for instance, the work of F. Nataf and his collaborators [347]; this is described at length in the survey paper [493] by M.J. Gander and H. Zhang.

7.6.2 ▪ DD without overlapping

In DD without overlapping and when using exact solvers for the subdomains, one generally eliminates the unknowns strictly inside the subdomains, obtaining a linear system for the interface unknowns.

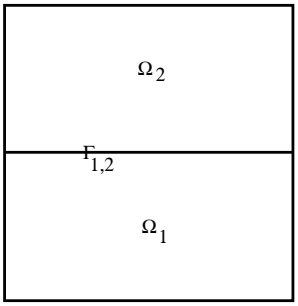


Figure 7.5. *Partitioning of the rectangle with non-overlapping subdomains*

For the sake of simplicity we first consider a square domain Ω decomposed into two subdomains. Let Ω_1 and Ω_2 be the two subdomains and $\Gamma_{1,2}$ the interface which is a mesh line, see Figure 7.5.

Let x_1 (resp. x_2) be the vector of unknowns in Ω_1 (resp. in Ω_2) and $x_{1,2}$ be the vector of the unknowns on the interface. Within each subdomain we order the unknowns with the usual rowwise ordering. With this numbering of the unknowns, the linear system can be rewritten blockwise as

$$\begin{pmatrix} A_1 & 0 & E_1 \\ 0 & A_2 & E_2 \\ E_1^T & E_2^T & A_{12} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_{1,2} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_{1,2} \end{pmatrix}. \tag{7.8}$$

The matrix A_1 (resp. A_2) represents the coupling of the unknowns within Ω_1 (resp. Ω_2). The matrix A_{12} represents the coupling of the unknowns on the interface. The matrix E_1 (resp. E_2) represents the coupling of Ω_1 (resp. Ω_2) with the interface. We eliminate the unknowns x_1 and x_2 corresponding to the subdomains. This gives a reduced system for the interface unknowns

$$Sx_{1,2} = \overline{b_{1,2}}, \tag{7.9}$$

with

$$\overline{b_{1,2}} = b_{1,2} - E_1^T A_1^{-1} b_1 - E_2^T A_2^{-1} b_2,$$

and

$$S = A_{12} - E_1^T A_1^{-1} E_1 - E_2^T A_2^{-1} E_2.$$

The matrix S is the Schur complement of A_{12} in A. Of course, A_1^{-1} and A_2^{-1} are generally dense matrices. Therefore, it is too costly to construct and factor S. A cheaper solution is to solve the reduced system with matrix S on the interface with an iterative method. The choice of the iterative method depends on the properties of the Schur complement. If the matrix A is a symmetric positive definite M-matrix, S is also a symmetric positive definite M-matrix and we can use the preconditioned conjugate gradient method.

One of the operations that has to be done for performing an iteration of PCG is the product of the matrix S by a given vector, say p. This may seem a costly operation. However the product, Sp can be computed easily as

$$Sp = A_{1,2}p - E_1^T A_1^{-1} E_1 p - E_2^T A_2^{-1} E_2 p,$$

p being a vector defined on the interface. Then, $w_1 = A_1^{-1} E_1 p$ is computed by solving

$$A_1 w_1 = E_1 p, \tag{7.10}$$

This is solving a linear system corresponding to a problem in Ω_1. Similarly, $w_2 = A_2^{-1}E_2p$ is computed by solving

$$A_2w_2 = E_2p, \tag{7.11}$$

a problem in Ω_2. Finally, we have

$$Sp = A_{1,2}p - E_1^Tw_1 - E_2^Tw_2.$$

Generally, we do not need all the components of w_1 and w_2, but this depends on the nonzero pattern of E_1 and E_2. We have just seen that we do not need to explicitly construct S to be able to compute Sp since this can be done through subdomain solves. Regarding parallel computing, the important fact is that the linear systems (7.10) and (7.11) are independent and therefore can be solved in parallel.

To improve the convergence rate of CG on the reduced system, a preconditioner M is needed. We would ideally like to find a preconditioner M such that

$$\kappa(M^{-1}S) = O(1).$$

This will imply that when the order of the matrix increases, the number of iterations stays approximately constant to reach a given precision. Then, the total number of operations will be proportional to the number of operations for one iteration. The main problem with this approach is to find an approximation of the Schur complement S.

Let us now assume that the matrix A is symmetric positive definite and that we do not want to use direct solvers for solving the problems with A_1 and A_2. We must construct a global preconditioner for the matrix A. Let us look for M in the form

$$M = L \begin{pmatrix} M_1^{-1} & & \\ & M_2^1 & \\ & & M_{1,2}^{-1} \end{pmatrix} L^T,$$

where M_1 (resp. M_2) is of the same order as A_1 (resp. A_2) and $M_{1,2}$ is of the same order as $A_{1,2}$. L is a block lower triangular matrix

$$L = \begin{pmatrix} M_1 & & \\ 0 & M_2 & \\ E_1^T & E_2^T & M_{1,2} \end{pmatrix}.$$

At each PCG iteration, we have to solve a linear system like

$$Mz = M \begin{pmatrix} z_1 \\ z_2 \\ z_{1,2} \end{pmatrix} = r = \begin{pmatrix} r_1 \\ r_2 \\ r_{1,2} \end{pmatrix}.$$

This is done by first solving $Ly = r$, where the first two steps are

$$M_1y_1 = r_1, \quad M_2y_2 = r_2.$$

The two solves can be done in parallel. Then, we solve for the interface

$$M_{1,2}y_{1,2} = r_{1,2} - E_1^Ty_1 - E_2^Ty_2.$$

The backward solve to obtain the solution is

$$\begin{pmatrix} I & 0 & M_1^{-1}E_1 \\ & I & M_2^{-1}E_2 \\ & & I \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_{1,2} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_{1,2} \end{pmatrix}.$$

It yields $z_{1,2} = y_{1,2}$ and

$$M_1 w_1 = E_1 z_{1,2}, \quad z_1 = y_1 - w_1,$$

$$M_2 w_2 = E_2 z_{1,2}, \quad z_2 = y_2 - w_2.$$

The last two steps can be done in parallel. Therefore, we have to perform two independent solves on the subdomains, a solve on the interface and two other independent solves on the subdomains. The problem we are facing is to choose the approximations M_1, M_2 and $M_{1,2}$. If we multiply together the three matrices whose product defines M, we obtain

$$M = \begin{pmatrix} M_1 & 0 & E_1 \\ 0 & M_2 & E_2 \\ E_1^T & E_1^T & M_{1,2}^* \end{pmatrix},$$

where

$$M_{1,2}^* = M_{1,2} + E_1^T M_1^{-1} E_1 + E_2^T M_2^{-1} E_2.$$

Therefore, since we would like M to be an approximation of A, it makes sense to choose

$$M_1 \approx A_1, \quad M_2 \approx A_2,$$

and

$$M_{1,2}^* \approx A_{1,2} \Longrightarrow M_{1,2} \approx A_{12} - E_1^T M_1^{-1} E_1 - E_2^T M_2^{-1} E_2.$$

That is, if the inverse of M_1 (resp. M_2) is a good approximation of the inverse of A_1 (resp. A_2), we are back to the same problem as before, $M_{1,2}$ must be an approximation of the Schur complement S.

The two different classes of methods we studied above give rise to the same fundamental problem of constructing an approximate Schur complement. Let us review a few preconditioners for a model problem arising from the finite difference discretization of a second order partial differential equation in a rectangle divided in two subdomains like in Figure 7.5 with m_1 and m_2 mesh lines in the subdomains and the mesh size $h = 1/(m+1)$.

Dryja's preconditioner [361] is defined as follows. Let T_2 be the matrix corresponding to the finite difference discretization of the one-dimensional Laplacian. We have

$$T_2 = Q_2 \Sigma_2 Q_2^T,$$

where Σ_2 is the diagonal matrix of the eigenvalues whose diagonal entries are

$$\sigma_i = 2 - 2\cos(i\pi h), \quad i = 1, \ldots, m$$

and Q_2 is the orthogonal matrix of the (normalized) eigenvectors whose components are

$$q_{i,j} = \sqrt{\frac{2}{m+1}} \sin(ij\pi h), \quad i, j = 1, \ldots, m.$$

Let $\sqrt{\Sigma_2}$ be the diagonal matrix whose diagonal entries are the square roots of the corresponding diagonal entries of Σ_2. The preconditioner M_D is

$$M_D = Q_2 \sqrt{\Sigma_2} Q_2^T.$$

Symbolically, we denote this by $\sqrt{-\Delta_{1D}}$.

Theorem 7.21. *Let $\lambda_\ell = 2 + \sigma_\ell$ be the eigenvalues of A and $\gamma_\ell = \left(1 + \frac{\sigma_\ell}{2} - \sqrt{\sigma_\ell + \frac{\sigma_\ell^2}{4}}\right)^2$, then the eigenvalues of S are*

$$\theta_\ell = \left(\frac{1+\gamma_\ell^{m_1+1}}{1-\gamma_\ell^{m_1+1}} + \frac{1+\gamma_\ell^{m_2+1}}{1-\gamma_\ell^{m_2+1}}\right)\sqrt{\sigma_\ell + \frac{\sigma_\ell^2}{4}}, \quad \forall \ell = 1, \ldots, m. \tag{7.12}$$

Moreover,

$$\lambda_\ell(M_D^{-1}S) = \left(\frac{1+\gamma_\ell^{m_1+1}}{1-\gamma_\ell^{m_1+1}} + \frac{1+\gamma_\ell^{m_2+1}}{1-\gamma_\ell^{m_2+1}}\right)\sqrt{1 + \frac{\sigma_\ell}{4}}, \quad \ell = 1, \ldots, m.$$

Proof. See [811]. □

For the Poisson model problem, $\kappa(M_D^{-1}S) = O(1)$.

The Golub and Mayers' preconditioner [540] is an improvement upon Dryja's preconditioner. It retains the term under the square root which was replaced by $\sqrt{\sigma_\ell}$ in the Dryja's method. It is defined as

$$M_{GM} = Q_2\sqrt{\Sigma_2 + \frac{\Sigma_2^2}{4}}Q_2^T.$$

Symbolically, we denote this by $\sqrt{-\Delta_{1D} + \frac{\Delta_{1D}^2}{4}}$.

Theorem 7.22. *The eigenvalues of $M_{GM}^{-1}S$ are*

$$\lambda_\ell(M_{GM}^{-1}S) = \frac{1+\gamma_\ell^{m_1+1}}{1-\gamma_\ell^{m_1+1}} + \frac{1+\gamma_\ell^{m_2+1}}{1-\gamma_\ell^{m_2+1}}, \quad \ell = 1, \ldots, m$$

where γ_ℓ is defined as in Theorem 7.21.

For the Poisson model problem, $\kappa(M_{GM}^{-1}S) = O(1)$.

The Neumann-Dirichlet preconditioner was introduced by P. Bjørstad and O.B. Widlund [131]. Since in the two subdomains case, A is written as

$$\begin{pmatrix} A_1 & 0 & E_1 \\ 0 & A_2 & E_2 \\ E_1^T & E_2^T & A_{1,2} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_{1,2} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_{1,2} \end{pmatrix},$$

we can distinguish what in $A_{1,2}$ comes from subdomain Ω_1 and what comes from Ω_2. Let

$$A_{1,2} = A_{1,2}^{(1)} + A_{1,2}^{(2)}.$$

This is easy to do for finite element methods. For finite difference methods, it is a little more difficult. However, for instance, for the one dimensional Poisson model problem, we have

$$A_{1,2}^{(1)} = A_{1,2}^{(2)} = \begin{pmatrix} 2 & -\frac{1}{2} & & & \\ -\frac{1}{2} & 2 & -\frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 2 & -\frac{1}{2} \\ & & & -\frac{1}{2} & 2 \end{pmatrix} = \frac{1}{2}A_{1,2}.$$

Since

$$S = A_{1,2} - E_1^T A_1^{-1} E_1 - E_2^T A_2^{-1} E_2,$$

we can define

$$S^{(1)} = A_{1,2}^{(1)} - E_1^T A_1^{-1} E_1, \quad S^{(2)} = A_{1,2}^{(2)} - E_2^T A_2^{-1} E_2.$$

With this notation, $S = S^{(1)} + S^{(2)}$. The Neumann-Dirichlet preconditioner is defined as

$$M_{ND} = S^{(1)}. \tag{7.13}$$

We observe that we could also have chosen $S^{(2)}$ instead of $S^{(1)}$.

Theorem 7.23.
The eigenvalues of M_{ND} are

$$\lambda_\ell(M_{ND}) = \left(\frac{1+\gamma_\ell^{m_1+1}}{1-\gamma_\ell^{m_1+1}}\right)\sqrt{\sigma_\ell + \frac{\sigma_\ell^2}{4}}, \quad \ell = 1, \ldots, m,$$

and the eigenvalues of $M_{ND}^{-1}S$ are

$$\lambda_\ell(M_{ND}^{-1}S) = 1 + \left(\frac{1-\gamma_\ell^{m_1+1}}{1+\gamma_\ell^{m_1+1}}\right)\left(\frac{1+\gamma_\ell^{m_2+1}}{1-\gamma_\ell^{m_2+1}}\right), \tag{7.14}$$

where γ_ℓ is defined as in Theorem 7.21.

For the Poisson model problem, $\kappa(M_{ND}^{-1}S) = O(1)$. This preconditioner is called "Neumann-Dirichlet" because solving a linear system

$$S^{(1)} y_{12} = (A_{1,2}^{(1)} - E_1^T A_1^{-1} E_1) y_{1,2} = c_{1,2},$$

is equivalent to solving

$$\begin{pmatrix} A_1 & E_1 \\ E_1^T & A_{1,2}^{(1)} \end{pmatrix} \begin{pmatrix} y_1 \\ y_{1,2} \end{pmatrix} = \begin{pmatrix} 0 \\ c_{1,2} \end{pmatrix}.$$

For second order elliptic PDEs, it is easy to see that this is simply solving a problem in Ω_1 with given Neumann boundary conditions on the interface. When the solution is known on the interface, it is enough to solve a Dirichlet problem in Ω_2.

The Neumann-Neumann preconditioner was introduced by P. Le Tallec [727]. With the same notation as above, it is defined as

$$M_{NN}^{-1} = \frac{1}{2}\left[(S^{(1)})^{-1} + (S^{(2)})^{-1}\right]. \tag{7.15}$$

We observe that we directly define the inverse of the preconditioner as an average of inverses of "local" inverses of Schur complements.

Theorem 7.24. *The eigenvalues of M_{NN}^{-1} are given by*

$$\lambda_\ell(M_{NN}^{-1}) = \frac{1}{2}\left(\frac{1-\gamma_\ell^{m_1+1}}{1+\gamma_\ell^{m_1+1}} + \frac{1-\gamma_\ell^{m_2+1}}{1+\gamma_\ell^{m_2+1}}\right)\frac{1}{\sqrt{\sigma_\ell + \frac{\sigma_\ell^2}{4}}},$$

and the eigenvalues of $M_{NN}^{-1}S$ *are*

$$\lambda_\ell(M_{NN}^{-1}S) = 1 + \frac{1}{2}\left(\frac{1-\gamma_\ell^{m_1+1}}{1+\gamma_\ell^{m_1+1}}\right)\left(\frac{1+\gamma_\ell^{m_2+1}}{1-\gamma_\ell^{m_2+1}}\right) + \frac{1}{2}\left(\frac{1-\gamma_\ell^{m_2+1}}{1+\gamma_\ell^{m_2+1}}\right)\left(\frac{1+\gamma_\ell^{m_1+1}}{1-\gamma_\ell^{m_1+1}}\right).$$

For the Poisson model problem, $\kappa(M_{NN}^{-1}S) = O(1)$.

There are other ways to construct preconditioners for the Schur complement S. One idea is to use probing. We compute the entries of M by requiring that

$$Mv_i = Sv_i,\ i = 1, \ldots, q$$

for a given set of vectors $v_i, i = 1, \ldots, q$. This idea was introduced in the preconditioning area by T.F. Chan [223] and O. Axelsson and B. Polman [72, 73]. Assume, for instance, that we would like to compute a tridiagonal approximation M of S. We set $q = 3$ and a possible choice of probing vectors is

$$\begin{aligned} v_1 &= (1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \ldots)^T, \\ v_2 &= (0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \ldots)^T, \\ v_3 &= (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad \ldots)^T. \end{aligned}$$

If we denote $y_i = Mv_i, i = 1, 2, 3$, we have

$$\begin{aligned} y_1 &= (m_{1,1} \quad m_{2,1} \quad m_{3,4} \quad m_{4,4} \quad m_{5,4} \quad \ldots)^T, \\ y_2 &= (m_{1,2} \quad m_{2,2} \quad m_{3,2} \quad m_{4,5} \quad m_{5,5} \quad \ldots)^T, \\ y_3 &= (0 \quad m_{2,3} \quad m_{3,3} \quad m_{4,3} \quad m_{5,6} \quad \ldots)^T. \end{aligned}$$

We see that if we have computed Sv_1, Sv_2, Sv_3 by solving six subproblems, we obtain the nonzero entries of M right away. This can be generalized in a straightforward way to construct a banded matrix M=Probe(S, d) with $2d + 1$ diagonals. However, constructing the approximation in this way leads to a nonsymmetric M even for a symmetric S. Nevertheless, this approximation possesses nice properties since T.F. Chan has shown that

$$\text{Probe}(\alpha S_1 + S_2, d) = \alpha\text{Probe}(S_1, d) + \text{Probe}(S_2, d)$$

and if a row of S is strictly diagonally dominant, the corresponding row of Probe(S, d) is also strictly diagonally dominant. A way to solve the symmetry problem is to use

$$M = \frac{1}{2}(\text{Probe}(S, d) + \text{Probe}(S, d)^T),$$

but then, the diagonal dominance property is not preserved. A better way to obtain a symmetric preconditioner is to define MProbe(S, d) by computing the i, j entry from M=Probe(S, d) as $m_{i,j}$ if $|m_{i,j}| = \min(|m_{i,j}|, |m_{j,i}|)$ and $m_{j,i}$ otherwise. This preserves symmetry and strict diagonal dominance.

By the way it is constructed, we cannot expect the tridiagonal probing approximation to give a condition number independent of the mesh size. T.F. Chan and T. Mathew [224] proved that, for the Poisson model problem

$$\kappa(M^{-1}S) = O\left(\frac{1}{\sqrt{h}}\right).$$

With two subdomains, let

$$B = \begin{pmatrix} B_1 & 0 & F_1 \\ 0 & B_2 & F_2 \\ F_1^T & F_2^T & B_{1,2} \end{pmatrix}$$

and let A be another matrix

$$A = \begin{pmatrix} A_1 & 0 & E_1 \\ 0 & A_2 & E_2 \\ E_1^T & E_2^T & A_{1,2} \end{pmatrix}$$

spectrally equivalent to B that is, there exist constants c_0 and c_1 such that

$$c_0(Bx, x) \leq (Ax, x) \leq c_1(Bx, x), \ \forall x.$$

It implies that

$$\kappa(B^{-1}A) \leq \frac{c_1}{c_0}.$$

Let $x = (\ x_1 \quad x_2 \quad x_{1,2}\ )^T$ and $y = (\ -A_1^{-1}E_1x_{1,2} \quad -A_2^{-1}E_2x_{1,2} \quad x_{1,2}\ )^T$. Then

$$Ay = \begin{pmatrix} 0 \\ 0 \\ (A_{1,2} - E_1^T A_1^{-1}E_1 - E_2^T A_2^{-1}E_2)x_{1,2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ Sx_{1,2} \end{pmatrix}.$$

With this choice of y we have

$$(Ay, y) = (Sx_{1,2}, x_{1,2}),$$

and

$$By = \begin{pmatrix} (-B_1A_1^{-1}E_1 + F_1)x_{1,2} \\ (-B_2A_2^{-1}E_2 + F_2)x_{1,2} \\ (B_{1,2} - F_1^T A_1^{-1}E_1 - F_2^T A_2^{-1}E_2)x_{1,2} \end{pmatrix}.$$

Let us compute (By, y),

$$\begin{aligned}(By, y) = &(y_1, (-B_1A_1^{-1}E_1 + F_1)x_{1,2}) \\ &+ (y_2, (-B_2A_2^{-1}E_2 + F_2)x_{1,2}) \\ &+ (x_{1,2}, (B_{1,2} - F_1^T A_1^{-1}E_1 - F_2^T A_2^{-1}E_2)x_{1,2}).\end{aligned}$$

Let us assume that there exist nonsingular matrices D_1 and D_2 such that

$$A_i = D_iB_i, \quad E_i = D_iF_i, \ i = 1, 2.$$

Then, the first and second terms in (By, y) are zero and the third one is $(S_Bx_{1,2}, x_{1,2})$, S_B being the Schur complement of B. This shows that we have

$$c_0(S_Bx_{1,2}, x_{1,2}) \leq (Sx_{1,2}, x_{1,2}) \leq c_1(S_Bx_{1,2}, x_{1,2}),$$

and S is spectrally equivalent to S_B. Hence, we can use the same preconditioners for S as for S_B. This happens, for instance, for the case $A_i = \rho_iB_i, E_i = \rho_iF_i, i = 1, 2$.

Of course, we have to consider what to do when we have more than two subdomains. If the problem arises from the discretization of a partial differential equation, we have to partition the mesh, whence for more general problems we have to partition the graph of the matrix. There are at least two ways to partition the mesh or the graph. The first one is to partition into strips (like in Figure 7.6 for a rectangle) and the second one is to allow more general subdomains like in

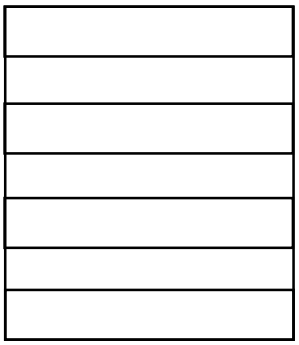


Figure 7.6. *Multi strips partitioning*

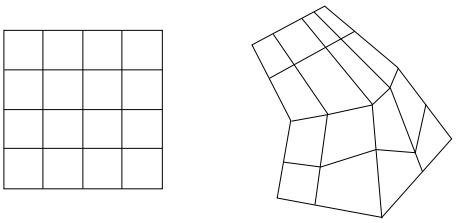


Figure 7.7. *Partitioning with boxes*

Figure 7.7. Most of the automatic mesh or graph partitioners, like parMETIS [694, 695], provide irregular subdomains.

A domain decomposition with strips can be done for general domains by finding pseudo-peripheral nodes and constructing the level structure corresponding to one of these nodes, see Chapter 3. This is similar to one-way dissection. If we partition the domain or graph into strips, when eliminating exactly the subdomain unknowns, the Schur complement S is block tridiagonal since an interface can only be linked to two other interfaces. In some simple cases the eigenvalues of S are known analytically, see [811].

The problem is to define preconditioners for S. A first idea is to use block diagonal preconditioners, the diagonal blocks being the preconditioners we define for the two subdomains case. Dryja's preconditioner removes the mesh dependency but not that on the number k of subdomains and, in fact, we have $\kappa(M_D^{-1}S) = O(k^2)$ for model problems. The same result holds for the Golub and Mayers' preconditioner although the condition number may be a little smaller. It is more difficult to generalize the Neumann-Dirichlet preconditioner to many subdomains. One obvious way is to use as preconditioner a block diagonal matrix whose blocks are the two subdomains Neumann-Dirichlet preconditioner. Another way to derive a preconditioner is to apply the same principle as with two subdomains, that is, to decompose the Schur complement into $S^{(1)} + S^{(2)}$. M. Dryja and W. Proskurowski [362] proved that the condition number is of order k^2.

The Neumann-Neumann preconditioner can be easily extended to many subdomains, the inverses of partial Schur complements have to be weighted by the inverse of the number of subdomains which share a given node.

If we cannot solve exactly for the subproblems, we are not able to use an iterative method with S since we cannot compute the matrix-vector product Sv. Hence, we have to iterate on all the unknowns and we need a global parallel preconditioner. For a partitioning into k strips we

can define the preconditioner as

$$M = L \begin{pmatrix} M_1^{-1} & & & & & & & & \\ & M_2^{-1} & & & & & & & \\ & & \ddots & & & & & & \\ & & & M_k^{-1} & & & & & \\ & & & & M_{1,2}^{-1} & & & & \\ & & & & & \ddots & & & \\ & & & & & & M_{k-1,k}^{-1} & & \end{pmatrix} L^T,$$

$$L = \begin{pmatrix} M_1 & & & & & & & & \\ & M_2 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \ddots & & & & & \\ & & & & M_k & & & & \\ C_1^T & E_2^T & & & & M_{1,2} & & & \\ & C_2^T & E_3^T & & & H_2 & M_{2,3} & & \\ & & \ddots & \ddots & & & \ddots & \ddots & \\ & & & C_{k-1}^T & E_k^T & & & H_{k-1} & M_{k-1,k} \end{pmatrix}.$$

The matrices M_i can be chosen as in the two subdomains case. For the matrices $M_{i,i+1}$ and H_i, we have many possibilities since there are many different ways to construct an incomplete factorization of a block tridiagonal matrix.

Let us now consider domain decomposition with boxes, see the right part of Figure 7.7. With this decomposition interfaces are made of two kinds of sets, the edges as before and the points where the interfaces cross which are known as vertices or cross points. We use an index E_i for the edges, $E = \cup E_i$ and V for the vertices. We observe that with the most standard finite element or finite difference approximations the cross points are not linked to points inside the subdomains but only to points on the edges. Moreover, there is no direct connection between the cross points since we require at least one mesh point per edge.

The nonzero structure of the Schur complement S depends on the choice of numbering of the points. Let us consider an example for the Poisson model problem with 121 nodes and 9 subdomains arranged as 3×3, each subdomain having 3×3 points with the following ordering of the points,

61	62	63	114	70	71	72	117	79	80	81
58	59	60	113	67	68	69	116	76	77	78
55	56	57	112	64	65	66	115	73	74	75
103	104	105	120	106	107	108	121	109	110	111
34	35	36	99	43	44	45	102	52	53	54
31	32	33	98	40	41	42	101	49	50	51
28	29	30	97	37	38	39	100	46	47	48
88	89	90	118	91	92	93	119	94	95	96
7	8	9	84	16	17	18	87	25	26	27
4	5	6	83	13	14	15	86	22	23	24
1	2	3	82	10	11	12	85	19	20	21

The points within the subdomains are numbered first, then the points on the edges, and, finally, the four vertices. With this ordering the nonzero structure of S is shown in Figure 7.8 with 3×3 blocks.

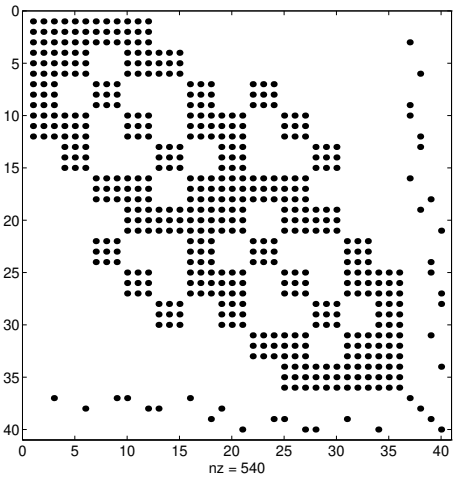


Figure 7.8. *Nonzero structure of S with boxes*

Because this is a small problem, one block is only linked to a maximum of 5 blocks. Moreover, the entries of S outside a tridiagonal nonzero structure are quite small except for the last 4 rows and 4 columns corresponding to the links with the cross points, see Figure 7.9. For the Poisson model problem, the magnitude of the entries depends only on the distance between points. However, this is not true for more general problems. We observe that there is a coupling between the points on different neighboring edges and this must be taken into account when constructing preconditioners.

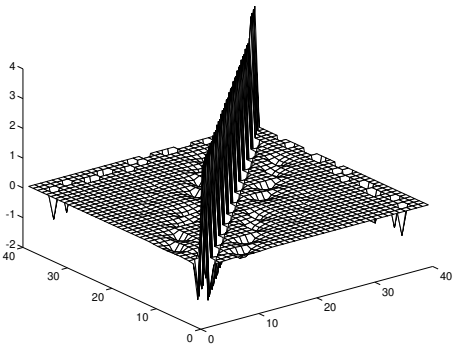


Figure 7.9. *Poisson model problem, S with boxes*

Let us describe a preconditioner that was proposed by J.H. Bramble, J.E. Pasciak, and A.H. Schatz in [144] and the subsequent papers [145, 146, 147] for finite element problems in two dimensions. This is a preconditioner, denoted as BPS, that had an important influence on the development of domain decomposition methods. It relies on deriving a simpler finite element approximation of the problem. Assume that we have to solve the problem

$$a(u, v) = (f, v), \ \forall v \in V$$

where a is a coercive bilinear form arising from a second order elliptic partial differential equation, V being a Hilbert space, say $H_0^1(\Omega)$ for homogeneous Dirichlet boundary conditions. Let

us construct another spectrally equivalent bilinear form $b(u,v)$ that is, satisfying

$$c_0 b(v,v) \leq a(v,v) \leq c_1 b(v,v),\ \forall v \in V.$$

Then, we use b as a preconditioner for a. The domain Ω is divided into non-overlapping subdomains Ω_k, the edges between two subdomains being denoted as $E_{\ell,m}$. Another intermediate form is introduced to allow for some averaging of the coefficients,

$$\tilde{a}(u,v) = \sum_k \sum_{i,j} \int_{\Omega_k} a_{i,j}^k \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_j}\, dx = \sum_k \tilde{a}_k(u,v).$$

The method separates interior, edges and vertices unknowns in the following way,

$$u = u_P + u_H,$$

where u_P is in $\sum_{\oplus} V^0(\Omega_k)$ with functions in $V^0(\Omega_k)$ having homogeneous Dirichlet boundary conditions and

$$u_P = 0 \text{ on } E_{\ell,m}.$$

The function u_P is defined as the solution of

$$\tilde{a}_k(u_P,\phi) = \tilde{a}_k(u,\phi),\ \forall \phi \in V^0(\Omega_k).$$

The function u_H is defined by

$$\tilde{a}_k(u_H,\phi) = 0,\ \forall \phi \in V^0(\Omega_k).$$

The method goes one step further and split u_H on the interfaces as

$$u_H = u_E + u_V,$$

where u_E stands for edge unknowns, u_V for vertices unknowns, $u_V(v_j) = u(v_j)$ and $u_V|_{E_{ij}}$ is linear, $u_E(v_j) = 0$. In [144] an operator ℓ_0 on the edges is defined, $V^0(E_{i,j}) \to V^0(E_{i,j})$ by

$$\int_{E_{i,j}} c^{-1} \ell_0(w)\phi = \int_{E_{i,j}} cw'\phi', \quad \forall \phi \in V^0(E_{i,j}),$$

where c is piecewise constant. This defines something which behaves like the one-dimensional Laplace operator. The bilinear form b is defined as

$$\begin{aligned} b(w,\phi) = {} & \tilde{a}(u_P,\phi_P) \\ & + \sum_{E_{i,j}} \int_{E_{i,j}} \alpha_{i,j} c^{-1} \ell_0^{1/2}(u_E)\phi_E \\ & + \sum_{E_{i,j}} (u_V(v_i) - u_V(v_j))(\phi_V(v_i) - \phi_V(v_j)). \end{aligned}$$

The basis functions are the usual polynomial ones for the interior nodes, one-dimensional hat functions vanishing at the vertices for the edges, and functions which are linear on each edge, being 1 at one vertex, 0 at the other ones for the vertices. The BPS preconditioner perform the following steps:

1) solve Dirichlet problems on each subdomain in parallel to obtain u_P,
2) solve one dimensional edge equations in parallel for u_E

3) solve a coarse mesh linear system on the set of vertices for u_V. From u_E and u_V we obtain the boundary values of u_H. The last step is

4) solve Dirichlet problems on each subdomain in parallel to obtain u_H.

The solution is $u_P + u_H$. For possible choices for the averaged coefficients, see [144].

Theorem 7.25. *Under suitable hypotheses, the condition number for the preconditioned system given by BPS in two-dimensional problems is*

$$\kappa \leq C \left(1 + \log^2 \left(\frac{H}{h}\right)\right),$$

where H is the coarse mesh size.

Proof. See J.H. Bramble, J.E. Pasciak, and A.H. Schatz [144]. □

There is only a slight h-dependency in the condition number and this gives a fast convergence when solving with CG. This dependency arises from the fact that the vertices are not directly linked to the neighboring edge nodes. Variants of the BPS preconditioner can also be defined as

$$M^{-1}v = \sum_{edges} R_{E_i}^T (\alpha_i M_i)^{-1} R_{E_i} v + R_H^T A_H^{-1} R_H v,$$

where R_{E_i} denotes the restriction to the edge E_i and R_H is a weighted restriction onto the coarse mesh, M_i being one of the preconditioners for the two subdomains case, either Dryja's or Golub-Mayers'.

A way to improve on BPS was proposed by B. Smith [1023, 1024, 1025, 1026]. It allows for a coupling between the vertices and the edge points. Let V_k be a set of points around each vertex on each of the edges. The preconditioner is defined as

$$M^{-1}v = R_H^T A_H^{-1} R_H v + \sum_{edges} R_{E_i}^T (M_{E_i})^{-1} R_{E_i} v + \sum_{vertices} R_{V_k}^T (M_{V_k})^{-1} R_{V_k} v.$$

It includes some coupling between neighboring edges. The edge preconditioner can be chosen as a weighting of Dryja's or Golub-Mayers' preconditioners. This is known as the vertex space preconditioner.

The use of probing to define M_{V_k} was proposed by T.F. Chan and T. Mathew [223]. The restriction to the edges is tridiagonal and an edge is only linked to the cross point and to the two points adjacent to the cross point on the neighboring edges. Five probing vectors are chosen to construct this approximation. If enough points are used around each vertex, the condition number is independent of the mesh size h and of the number of subdomains.

The finite element tearing and interconnecting (FETI) method was introduced by C. Farhat and F.X. Roux [443], initially for structural mechanics. FETI is a domain decomposition method where Lagrange multipliers are introduced at the subdomain interfaces to enforce the continuity of the solution. Its convergence was studied by J. Mandel and R. Tezaur [778]. Optimized interface preconditioners for FETI were proposed by M.J. Gander and H. Zhang [492]. An enhanced algorithm named FETI-DP (DP standing for "dual-primal") was described by C. Farhat, M. Lesoinne, P. Le Tallec, K. Pierson, and D. Rixen [441], see also [442].

In these algorithms the interface unknowns are partitioned into corners (cross points) which can have different definitions, and the remainder of the interface unknowns. FETI-DP insures the continuity of the corner unknowns, whereas all other unknowns are defined at the subdomain level. The continuity is obtained by using Lagrange multipliers. The problem is formulated as a

dual-primal problem relating the dual Lagrange multiplier unknowns λ to the primal unknowns. These latter ones are eliminated giving a symmetric positive definite linear system for the Lagrange multipliers which is solved by PCG. In doing so, one has to solve a subproblem for the corner unknowns that can be interpreted as a coarse problem.

Hundred of papers have been written over the years about variants of FETI-DP or about applications to particular problems.

The balancing domain decomposition (BDD) preconditioner was introduced by J. Mandel [774] by adding a coarse component to the Neumann-Neumann preconditioner. The algorithm is well explained in [775] by J. Mandel and M. Brezina. For problems arising from finite element discretization of two- and three-dimensional elliptic problems, the condition number of the preconditioned matrix is bounded from above by

$$c\left(1+\log^2\left(\frac{H}{h}\right)\right),$$

where the constant c does not depend on H, h, or the coefficients of the PDE.

A balancing domain decomposition with constraints (BDDC) was proposed in [341] by C.R. Dohrmann, mainly for problems in solid mechanics. This method minimizes the energy norm $x^T Ax/2$ under some constraints $C_i x_i = e_j$ where $x_i = R_i x$ for the ith subdomain and e_j is the jth column of the identity matrix. Each row of C_i is associated with a coarse unknown (cross points) common to two or more subdomains. Implementation details and numerical experiments are given in [341]. The BDDC algorithm involves the choice of a set of primal unknowns and the choice of an averaging operator which restores the continuity of the approximate solution across the interfaces between the subdomains. The choice of this last component of the algorithm was further discussed by O.B. Widlund and C.R. Dohrmann [1117] with an algorithm named "BDDC deluxe". Inexact solvers for the coarse problem were considered by C.R. Dohrmann, K.H. Pierson, and O.B. Widlund [342].

FETI-DP and BDDC are two methods which are very closely related, see J. Mandel and C.R. Dohrmann [776], as well as J. Mandel and B. Sousedík [777] where the equivalence of the two methods is discussed.

In [208] L. Carvalho considered a few preconditioners whose spirit is close to the vertex space preconditioner. They involve some overlapping between the edge and vertex parts. This is why they are denoted as algebraic additive Schwarz (AAS). He studied several local block preconditioners for the subdomains and several coarse space preconditioners. For the local preconditioners, the main difference with the vertex space preconditioner is that the edge and the adjacent vertices are considered together, see Figure 7.10.

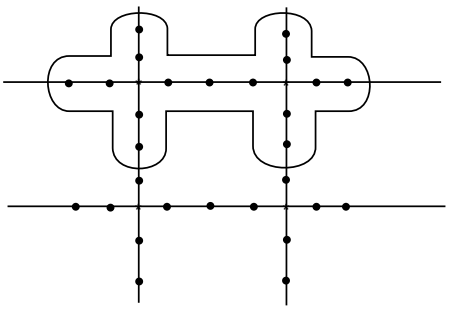


Figure 7.10. *The points in the vertex space*

Another proposal was to consider the complete boundary of one subdomain, to be able to

take into account all the couplings between the edge points and the vertices when the interior points are eliminated. The local preconditioners are obtained by probing [208]. A set of numerical experiments on elliptic problems shows that there is a $1/H^2$ dependency in the number of iterations. Therefore, a coarse space component is added to the algorithm. A restriction operator R_0 is defined and the coarse component of the preconditioner is defined as $R_0^T A_0^{-1} R_0$ where A_0 is the Galerkin coarse space operator $A_0 = R_0 S R_0^T$. Several possibilities were considered,

i) a subdomain-based coarse space where all the boundary points of a subdomain are considered. The coarse space is spanned by vectors which have nonzero components for the points around a subdomain, for all subdomains.
ii) a vertex-based coarse space where the vertices and some few adjacent edge points are considered.
iii) an edge-based coarse space where the points of an edge and the adjacent vertices are considered.

The combination of one of these coarse space preconditioners with the local parts gives a preconditioner for which the condition number is insensitive to the mesh size or the number of subdomains, except for very highly anisotropic problems. For more details and numerical experiments, see L.M. Carvalho, L. Giraud, and G.M. [209].

Domain decomposition is still an active area of research, see the proceedings of the DD conferences. The developments are now more focused on the application of these ideas to many more difficult problems in science and engineering.

There are several books about domain decomposition: B.F. Smith, P.E. Bjørstad, and W.D. Gropp [1026] in 1996, A. Quarteroni and A. Valli [934] in 1999, A. Toselli and O. Widlund [1069] in 2004, C. Pechstein [909] in 2013, and V. Dolean, P. Jolivet, and F. Nataf [347] in 2015.

7.7 ▪ Algebraic multigrid and multilevel methods

The multigrid method was originally devised as an iterative method for solving problems arising from the discretization of elliptic partial differential equations on simple domains. This method is now known as *geometric multigrid*. Let us briefly describe the ingredients of the method on a simple example.

Let us assume that we would like to compute an approximate solution of a second order linear elliptic partial differential equation $Lu = f$ in a square domain Ω with Dirichlet boundary conditions. The domain Ω is covered with a regular mesh of stepsize $h = \frac{1}{m+1}$, m being odd, giving a grid Ω_h. The PDE is discretized using the classical five-point finite difference scheme. An example of such a problem is given by the Poisson model problem we have already seen in preceding chapters. It yields a linear system $Au = b$. This change of notation from x to u means we are solving a system arising from a continuous problem and it is motivated by using a notation which is almost standard in the multigrid literature.

A two-grid method uses a coarse grid whose stepsize is $H = 2h$. Let u_k be an approximation of the solution u and $\varepsilon_k = u - u_k$ be the error. We have already seen that

$$A\varepsilon_k = Au - Au_k = b - Au_k = r_k.$$

Solving this equation is as difficult as the original problem. The idea of multigrid is to solve this problem on the coarse grid. The error being defined on Ω_h, we need a method to go from Ω_h to Ω_H. Therefore we define a linear restriction operator R

$$R : \Omega_h \to \Omega_H, \quad (\mathbb{R}^{m^2} \to \mathbb{R}^{p^2})$$

The representation of R is a $p^2 \times m^2$ rectangular matrix. When we have solved the problem on Ω_H, we need to go back to Ω_h. A linear prolongation (interpolation) operator P is defined,

$$P : \Omega_H \to \Omega_h, \quad (\mathbb{R}^{p^2} \to \mathbb{R}^{m^2}).$$

We also have to define the problem to be solved on the coarse grid. The coarse grid matrix is most often defined as $A_H = RAP$. The coarse grid linear system can be solved by a direct method. The iteration matrix of such a two-grid method would be $I - PA_H^{-1}RA$. Unfortunately, when A is symmetric positive definite, the eigenvalues of that matrix are 0 and 1, and the method does not converge. The eigenvectors of A span a basis of $\mathbb{R}^n$. It corresponds to (or converges towards when $h \to 0$) the eigenfunctions of L. Some of these eigenfunctions vary rapidly (high frequencies), some others are smooth (low frequencies). The components of any vector on the high frequency eigenvectors cannot be well approximated on the coarse grid. Hence, to have a method that converges, we need smooth residuals such that $r_k^H = Rr_k$ is a good approximation of r_k.

It is well known that classical iterative methods such as relaxed Jacobi or Gauss-Seidel give smoother and smoother errors and residuals as the methods proceed. Therefore, a few iterations of these methods can be used as a smoother for the residual vectors. We denote by S the iteration matrix of the chosen smoothing method. The two-grid algorithm is the following,

1) Starting from u_k, do ν_1 iterations of the smoothing method. Let $\bar{u}_k$ be the resulting vector.
2) $\bar{r}_k = b - A\bar{u}_k$,
3) $\bar{r}_k^H = R\bar{r}_k$,
4) solve exactly $A_H \varepsilon_k^H = \bar{r}_k^H$,
5) $v_k = P\varepsilon_k^H$,
6) starting from $\bar{u}_k + v_k$, do ν_2 iterations of the smoothing method. Let u_{k+1} be the result.
7) if no convergence go to 1.

The iteration matrix of the two-grid method is

$$M = S^{\nu_2}(I - PA_H^{-1}RA)S^{\nu_1} = S^{\nu_2}KS^{\nu_1}.$$

What we have just described is only a general framework. By combining the different possibilities for each component of the method, many variants can be generated. We have to choose

- the smoothing method S,
- the integers ν_1 and ν_2,
- how to construct the coarse grid,
- the restriction operator R,
- the prolongation operator P,
- how to define the coarse matrix A_H.

The multigrid method is defined by using the two-grid method recursively. The method uses a sequence of grids Ω_ℓ whose mesh sizes are h_ℓ, $\ell = 0$ corresponding to the coarsest grid and $\ell = L$ to the finest grid.

We denote

- A_ℓ the approximation of A on Ω_ℓ,
- R_ℓ the restriction operator: $\Omega_\ell \to \Omega_{\ell-1}$,

- $P_{\ell-1}$ the interpolation operator: $\Omega_{\ell-1} \to \Omega_\ell$,
- S_ℓ the iteration matrix of the smoothing operator on Ω_ℓ.

Let $\bar{w}_\ell = smooth^\nu(w_\ell, A_\ell, b_\ell)$ be the result of ν smoothing iterations for the problem $A_\ell u_\ell = b_\ell$ starting from w_ℓ. The multigrid algorithm is the following,

If $\ell = 1$ apply the two-grid algorithm.
If $\ell > 1$
1) $\bar{u}_k^\ell = smooth^{\nu_1}(u_k^\ell, A_\ell, b_\ell)$,
2) $\bar{r}_k^\ell = b_\ell - A_\ell \bar{u}_k \ell$,
3) $\bar{r}_k^{\ell-1} = R_\ell \bar{r}_k^\ell$,
4) compute $\bar{v}_k^{\ell-1}$ as the approximate solution of

$$A_{\ell-1} v_k^{\ell-1} = \bar{r}_k^{\ell-1}$$

on $\Omega_{\ell-1}$ by doing γ iterations of the ℓ-grid algorithm $(\Omega_{\ell-1}, \ldots, \Omega_0)$ starting from 0.
5) $\bar{v}_k^\ell = P_{\ell-1} \bar{v}_k^{\ell-1}$,
6) $u_{k+1}^\ell = smooth^{\nu_2}(\bar{u}_k^\ell + \bar{v}_k^\ell, A_\ell, b_\ell)$.

In this algorithm there is a new parameter γ which indicates how we solve the problem on the grid $\Omega_{\ell-1}$. If $\gamma = 1$, we have what is called a V-cycle since we visit the grids from the finest one to the coarsest one and back to the finest one. The value $\gamma = 2$ gives what is called a W-cycle. Another way of navigating through the levels that is not described by the previous algorithm is the F-cycle. Starting from the finest level $\ell = L$, one goes down to the coarsest level $\ell = 0$ as in the V-cycle, then up to $\ell = 1$, down to $\ell = 0$, up to $\ell = 2$, down to $\ell = 0$, up to $\ell = 3$, down to $\ell = 0$, and so on until back to $\ell = L$.

For one step, the V-cycle is the fastest method, the W-cycle is the more expensive, and the F-cycle is in between. However, the convergence of the V-cycle is usually the worst, and that of the W-cycle the best.

Let us denote by $M_\ell^{\ell-1}$ the iteration matrix of the two-grid method with grids Ω_ℓ and $\Omega_{\ell-1}$. As we have seen above,

$$M_\ell^{\ell-1} = S_\ell^{\nu_2}(I - P_{\ell-1} A_{\ell-1}^{-1} R_\ell A_\ell) S_\ell^{\nu_1}.$$

Using the multigrid method is replacing $A_{\ell-1}^{-1}$ by γ iterations of the ℓ-grid method starting from 0, that is,

$$A_{\ell-1}^{-1} \to (I - M_{\ell-1}^\gamma) A_{\ell-1}^{-1},$$

Let $M_{\ell-1}$ be the iteration matrix of the ℓ-grid method. Then,

$$M_\ell = S_\ell^{\nu_2}(I - P_{\ell-1}(I - M_{\ell-1}^\gamma) A_{\ell-1}^{-1} R_\ell A_\ell) S_\ell^{\nu_1},$$

for $\ell > 2$ and

$$M_1 = S_1^{\nu_2}(I - P_0 A_0^{-1} R_1 A_1) S_1^{\nu_1}.$$

By straightforward algebra, we obtain a recursive relation for the iteration matrix,

$$M_\ell = M_\ell^{\ell-1} + (S_\ell^{\nu_2} P_{\ell-1}) M_{\ell-1}^\gamma (A_{\ell-1}^{-1} R_\ell A_\ell S_\ell^{\nu_1}).$$

Many papers have been written about geometric multigrid convergence. Most of these papers made some assumptions on the smoother and the restriction and prolongation (interpolation) operators. Let us just cite a result by W. Hackbusch [600]. The first assumption is known as the

smoothing property[4], one assumes that there are functions $\eta(\nu)$ and $\bar{\nu}(h)$ independent of ℓ such that

$$\|A_\ell S_\ell^\nu\| \leq \eta(\nu)\|A_\ell\| \text{ for all } 0 \leq \nu \leq \bar{\nu}(h_\ell).$$

These functions are such that $\lim_{\nu\to\infty} \eta(\nu) = 0$ and $\lim_{h\to 0} \bar{\nu}(h) = \infty$ or $\bar{\nu}(h) = \infty$. The second assumption is known as the approximation property,

$$\|A_\ell^{-1} - PA_{\ell-1}^{-1}R\| \leq \frac{C}{\|A_\ell\|}, \forall \ell \geq 1.$$

Using these properties and assuming that $\|S_\ell^\nu\| \leq C$, $C_1\|x_{\ell-1}\| \leq \|Px_{\ell-1}\| \leq C_2\|x_{\ell-1}\|$, $\gamma \geq 2$, W. Hackbusch proved that

$$\|M_\ell\| < 1, \text{ for } \nu \leq \min \bar{\nu}(h_\ell),$$

provided the mesh size h is small enough. If $\bar{\nu}(h) = \infty$ the choice of the mesh size is not restricted.

The theory for geometric multigrid is described in several books: W. Hackbusch [599, 600], P. Wesseling [1114], J.H. Bramble [143], W.L. Briggs, V.E. Henson, and S.F. McCormick [173], and U. Trottenberg, C.W. Oosterlee, and A. Schüller [1074]

Algebraic multigrid (AMG) was designed to be able to use multigrid ideas for linear systems not necessarily arising from the discretization of a PDE. The word "algebraic" refers to the fact that these methods use only the matrix A of the linear system. But, in the multigrid framework, we have to define coarse problems. In geometric multigrid the finest levels are often obtained by refinements of a coarse mesh. AMG proceeds in the opposite way, obtaining coarse levels by coarsening. This is done by using the graph of the matrix A. The n vertices (nodes) of the graph are split into two sets: fine nodes $\mathcal{F}$ and coarse nodes $\mathcal{C}$, with $\mathcal{F} \cup \mathcal{C} = \{1, \ldots, n\}$ and $\mathcal{F} \cap \mathcal{C} = \emptyset$. The coarsening operation is then applied recursively to the set $\mathcal{C}$ until obtaining a number of nodes for which the problem can be solved efficiently by a given linear solver. In the following we will still speak of coarse grids, even though we are just working with (sub)graphs.

As in geometric multigrid, we have to choose a smoother, a restriction R, and a prolongation P (often chosen as R^T). Many choices have been proposed over the years. Here we can just briefly describe a few of them.

Let us assume that the matrix A is symmetric and start with the coarsening. For each node i, we define a set of dependencies Σ_i and an influence matrix Σ whose rows are the Σ_i's. Again, this can be done in many ways. The goal is to find the strong connections between nodes using the matrix entries. What can be considered as the standard AMG algorithm [965, 173] for an M-matrix defines

$$\Sigma_i = \{j | -a_{i,j} > \tau \max_{k\neq i}(-a_{i,k}), \ \tau < 1\},$$

where τ is a given parameter that defines which entries are strongly connected to i. The set of points that i influences is $\Sigma_i^T = \{j | \ i \in \Sigma_j\}$. This definition can be generalized to any matrix as

$$\Sigma_i^A = \{j \mid |a_{i,j}| > \tau \max_{k\neq i} |a_{i,k}|, \ \tau < 1\}.$$

We observe that this influence matrix is local since we are looking only at the neighbours of i in the graph of A.

[4]Note that there are other definitions of the smoothing property in the literature, see [1131]

Rather than using an influence matrix given by the entries of A, it seems more natural to measure the influences of the points by the inverse of A since it describes how the unknowns are linked together. Therefore, the idea is to use an approximate inverse of A with entries $m_{i,j}$,

$$\Sigma_i^M = \{j \in [1, \ldots, n], j \neq i | \; m_{i,j} \neq 0\}.$$

We can use the approximate inverse given by AINV [117, 114] or that in [1058]. When using AINV the approximate inverse is only given in factored form $ZD^{-1}Z^T$. Thus we could also define the influence matrix as

$$\Sigma_i^Z = \{j \in [1, \ldots, n], j \neq i | \; n_{i,j} \neq 0\}.$$

with $N = \tilde{Z} + \tilde{Z}^T - Q$, where Q is a diagonal matrix whose diagonal entries are the square roots of those of D^{-1} and $\tilde{Z} = ZQ$.

When solving anisotropic PDE problems, using AINV to define the set of dependencies can lead to obtain too many connections and coarse grids with very few points. A way to avoid this is to compute the coarse grid and then to check if every $\mathcal{F}$ node has at least one $\mathcal{C}$ node in its neighbours in the graph of A. If this is not the case, we can choose one of the neighbours and change its status to a $\mathcal{C}$ node.

Once the matrix Σ is obtained by any method, there are different ways we can follow to decide which are the $\mathcal{F}$ and $\mathcal{C}$ nodes. What can be considered as the standard coarsening algorithm is based on two principles:

1. For each $i \in \mathcal{F}$, each node $j \in \Sigma_i$ should either be in $\mathcal{C}$ or should depend on at least one node in $\mathcal{C}_i$ which is the set of coarse nodes used for the interpolation at node i,

2. $\mathcal{C}$ should be (as most as possible) a maximal subset with the property that no $\mathcal{C}$ node depends on another $\mathcal{C}$ node.

The first principle tends to increase the number of $\mathcal{C}$ nodes. The second one is used to limit the number of nodes in the coarse grid. The coarsening algorithm is defined by two passes. The first one uses weights w_i which are the number of nodes that depend on i. One step of the algorithm is the following,

1. Choose the first node i with maximal weight as a $\mathcal{C}$ node.

2. Assign the nodes that i influences as $\mathcal{F}$ points.

3. Increment by 1 the weights of the nodes influencing these new $\mathcal{F}$ nodes.

4. Decrease by 1 the weights of nodes that depends on i.

This guarantees that each $\mathcal{F}$ node has at least one connection to a $\mathcal{C}$ node. This is needed for the standard interpolation scheme. It tends sometimes to produce too many $\mathcal{F}$ nodes. A second pass could be added in which some $\mathcal{F}$ nodes are changed into $\mathcal{C}$ nodes to enforce the first criterion and to minimize $\mathcal{C}$-$\mathcal{C}$ connections. The idea is to test each $\mathcal{F}$ node to see if the first criterion is satisfied. The neighbours of i are split into the coarse nodes $\mathcal{C}_i$, the strongly connected non interpolatory nodes D_i^S (those which belong to Σ_i) and the weakly connected non interpolatory nodes D_i^W. If there is a node in D_i^S which is not connected to $\mathcal{C}_i$, it is tentatively flag as a $\mathcal{C}$ node. If the first criterion is satisfied with this new $\mathcal{C}$ node, it is definitely considered as a $\mathcal{C}$ node and testing on other $\mathcal{F}$ nodes continues. There are many cases for which this expensive second pass is not needed.

There are many others ways to generate the $\mathcal{F}$ and $\mathcal{C}$ nodes; see, for instance, the algorithm proposed by A.J. Cleary, R.D. Falgout, V.E. Henson, and J.E. Jones [269]. Their algorithm was designed to be used on parallel computers. Another possibility is to use a greedy aggregation algorithm. For example, we set $n_c = 0$, and for $i = 1, \ldots, n$ we do the following steps:

1. If i and all its neighbours have not been visited yet, we set $n_c = n_c + 1$, the subgraph whose nodes are i and its neighbours is labelled n_c, and we mark i and its neighbours as visited.

2. If at least one neighbour of i has been visited, we continue the loop over the nodes.

At the end of the loop on i it may happen that there are nodes which do not belong to any aggregate. They are added to the neighbouring aggregate having the least number of nodes.

For AMG algorithms based on aggregation, see P. Vaněk, J. Mandel, and M. Brezina [1094, 1095], and Y. Notay [864, 865], as well as A. Napov and Y. Notay [850].

We observe that the construction of the coarse grids depends on the matrices on the coarse levels and therefore also on the interpolation scheme which gives P and R. This implies that if we change the interpolation scheme, the number and location of the coarse nodes also change.

Let us now consider the interpolation scheme. The geometric multigrid algorithm uses bilinear interpolation. But, it is well known that this is not satisfactory for general problems. The standard AMG algorithm uses instead an interpolation based on the equations of the linear system. We know the values at the $\mathcal{C}$ nodes, arising from the solve on the next coarsest grid, and we have to compute interpolated values at the $\mathcal{F}$ nodes. For a node i in $\mathcal{F}$, the interpolation weight with a coarse node j is

$$\omega_{i,j} = -\frac{a_{i,j} + \sum_{k \in D_i^S} \frac{a_{i,k} a_{k,j}}{\sum_{m \in \mathcal{C}_i} a_{k,m}}}{a_{i,i} + \sum_{k \in D_i^W} a_{i,k}}.$$

This is obtained by writing the equations for $Az = 0$ and by doing some approximations; namely, writing that $z_j \approx z_i$ for weak connections and using a weighted average for $\mathcal{F}$ connections. Note that the given $\mathcal{F}$ node i needs to have at least one coarse point in its neighbourhood in the graph of A in order to be able to use this interpolation scheme.

We can also use an approximate inverse to compute the interpolation weights, see [814]. For an $\mathcal{F}$ point i, the interpolation weights $w_{i,j}$ are defined as

$$w_{i,j} = \frac{n_{i,j}}{\sum_{\ell \in \mathcal{C}_i} n_{i,\ell}} \quad j \in \mathcal{C}_i,$$

where $n_{i,j} = m_{i,j}$ or $N = \tilde{Z} + \tilde{Z}^T - Q$. The rationale behind this choice is that the points which are more important for interpolation are the ones with the strongest connections.

Another possibility is the energy minimization interpolation described by W.L. Wan, T.F. Chan, and B.F. Smith in the finite element framework, see [1109]. However, this algorithm can be used in a more general setting. The prolongation operator relates the coarse grid basis functions ϕ_i^H to the fine grid basis functions ϕ_i^h,

$$[\phi_1^H \cdots \phi_m^H] = [\phi_1^h \cdots \phi_n^h] P.$$

The coarse grid functions can be expressed in the fine grid basis,

$$\phi_i^H = \sum_{j=1}^{n} \varphi_j^i \phi_j^h,$$

and we are looking for coefficients φ_j^i that minimize the A-norm of the coarse grid basis functions satisfying the property that the interpolation of a constant value is exact. Let

$$\varphi^i = (\varphi_1^i \dots \varphi_n^i)^T, \quad \phi = (\varphi^1 \dots \varphi^m)^T.$$

Then, the minimization problem is

$$\min \frac{1}{2}\phi^T Q \phi, \quad B^T \phi = \mathbf{1},$$

where $\mathbf{1}$ is the vector of all ones, Q is a block diagonal matrix whose diagonal blocks Q_i are given by $(Q_i)_{k,\ell} = a_{k,\ell}$ if k and ℓ are neighbours of i in the graph of A and $\delta_{k,\ell}$ otherwise. The constraint matrix B is given as $B^T = (I_1^T \cdots I_m^T)$, with $(I_i)_{k,\ell} = 1$ if $k = \ell$, i being a neighbour of k and 0 otherwise. The minimization problem is solved using a Lagrange multiplier Λ. This gives a linear system

$$\begin{pmatrix} Q & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} \phi \\ \Lambda \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

This is solved by eliminating ϕ to get $(B^T Q^{-1} B)\Lambda = -1$. Fortunately, Q^{-1} is not difficult to obtain since Q is block diagonal with small blocks. Once, we have ϕ and therefore the φ^j's, we set the prolongation (interpolation) matrix as $P = (\varphi^1 \cdots \varphi^m)$.

More interpolation schemes are described in [813]. For a long-range interpolation scheme, see U. Meier Yang [794].

The coarse grid matrix is generally defined as $A_{\ell-1} = R_\ell A_\ell P_{\ell-1}$.

There exist many choices for the smoother in the literature. Let us describe a few. One of the most used smoother for symmetric problems is the symmetric Gauss-Seidel iteration. The relaxed Jacobi iteration is also popular because it is a parallel iteration. Overlapping Schwarz methods give another possibility.

Another smoother which has been proposed is the incomplete Cholesky (IC) factorization $LD^{-1}L^T$, where L is lower triangular and D diagonal, see Section 7.3. As we have seen above, there are many variants of this algorithm. The most popular one is to use a factorization for which the nonzero structure of L is the same as the structure of the lower triangular part of A. However, one can also keep a part of the fill-in either by looking at the size of the entries or by using the levels of fill-in.

One can also use an approximate inverse M^{-1} as a smoother in a Richardson iteration defined as

$$x_{k+1} = x_k + M^{-1}(b - Ax_k),$$

when solving $Ax = b$, see [814]. This can be used with the approximate inverses described for the coarsening algorithm. A few iterations of the preconditioned conjugate gradient has also been proposed as a smoother.

A least squares polynomial can also be used in a Richardson iteration, see [813] where numerical experiments are described.

The smoother and the coarse spaces are closely related. As stated in [1131], "any chosen smoother is expected to converge well only on certain components of the solution, which will be known as algebraic high frequencies with respect to the given smoother. With the smoother fixed, the main task of an AMG method is then to identify a sequence of coarse spaces that would complement this smoother well".

We must add that, nowadays, algebraic multigrid is most often used as a preconditioner in Krylov iterative methods and not as a stand-alone algorithm.

Since the multigrid methods became popular, the idea of using several levels has been used in different areas of numerical linear algebra.

The use of multilevel orderings for incomplete Cholesky and LU preconditioners has been considered starting in the 1990s. Research in this direction was done by O. Axelsson [65, 66], O. Axelsson and P.S. Vassilevski [74, 75, 76, 77], and O. Axelsson and V. Eijkhout [68]. We can also refer to the work of Y. Notay [860, 862, 863].

Let us assume that A is an M-matrix. We first consider a two-level algorithm, the matrix A being partitioned as

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}.$$

The block $(1,1)$ refers to unknowns corresponding the "fine" grid and $(2,2)$ to unknowns on the "coarse" grid. A two-level preconditioner is defined as

$$M = \begin{pmatrix} M_{1,1} & \\ A_{2,1} & S \end{pmatrix} \begin{pmatrix} I & M_{1,1}^{-1} A_{1,2} \\ & I \end{pmatrix},$$

where $M_{1,1}$ is a (modified) incomplete LU factorization of $A_{1,1}$ without fill-in, and $S = A_{2,2} - A_{2,1} K A_{1,2}$ is an approximation of the Schur complement of A, K being a diagonal matrix to be defined. By choosing a diagonal approximation, the reduced system has the same structure as the original problem. When $M_{1,1}$ is a modified ILU preconditioner, we have $M_{1,1}e = A_{1,1}e$, where $e = (1, 1, \ldots, 1)^T$. Let D be a diagonal matrix such that $De = A_{1,1}e$. Then, $k_{i,i}$ can be set equal to $d_{i,i}^{-1}$ if the diagonal entry is nonzero and zero otherwise. When A is symmetric, S is a symmetric M-matrix.

Bounds for the condition number of $M^{-1}A$ are given in [862]. The multilevel method is obtained by a recursive application of the two-level preconditioner. Let M_S be the preconditioner for the next coarsest level. A solve with S is replaced by a multiplication by $P_2(M_S^{-1}S)S^{-1}$ where P_2 is a second order polynomial such that $P_2(0) = 0$. It was proposed to use shifted Chebyshev polynomials. Using a second order polynomial leads to a kind of W-cycle algorithm. Numerical results in [862] show that this method seems quite robust. In [860] Y. Notay proposed using a V-cycle with a similar algorithm by replacing the use of the second order polynomial by some smoothing (for instance, a relaxed Jacobi iteration).

The incomplete factorization multigraph algorithm was proposed by R.E. Bank and R.K. Smith [95] and the multilevel ILU decomposition by R.E. Bank and C. Wagner [96].

For multilevel incomplete factorizations we can also cite (in alphabetical order), O. Axelsson and S. Margenov [71], Z.Z. Bai [81], M. Bollhöfer and V. Mehrmann [138], M. Bollhöfer and Y. Saad [141], T.F. Chan, C.C.J. Kuo, and C. Tong [222], W. Dahmen and A. Kunoth [305], H.C. Elman and X.-Z. Guo [418], Y.A. Erlangga and R. Nabben [427], T. Grauschopf, M. Griebel, and H. Regler [555], T. Huckle and J. Staudacher [659], R. Kehl, R. Nabben, and D.B. Szyld [696], C.C.J. Kuo and T.F. Chan [669, 670], Y. Kuznetsov and A. Prokopenko [717], Z. Li, Y. Saad, and M. Sosonkina [744], Y. Notay [861, 864], Y. Saad [978], Y. Saad and B. Suchomel [981], Y. Saad and J. Zhang [983], P.S. Vassilevski [1101], A. van der Ploeg, E.F.F. Botta, and F.W. Wubs [1082], H. Yserentant [1147], Y. Xi, R. Li, and Y. Saad [1126], T. Xu, V. Kalantzis, R. Li, Y. Xi, G. Dillon, and Y. Saad [1132], J. Zhang [1154], and X. Zhang [1156]. Many other papers were devoted to applications of the multilevel techniques.

We have seen above that it is useful to add a coarse space component to the additive Schwarz preconditioners. It is relatively easy to generalize these two-level methods to a multilevel algorithm. Assume that we have $L + 1$ levels, the mesh or graph of each level ℓ being decomposed into $N^{(\ell)}$ subdomains denoted as Ω_i^ℓ. Then, the multilevel additive Schwarz preconditioner is

defined as

$$M^{-1} = \sum_{\ell=0}^{L} \sum_{i=1}^{N^{(\ell)}} (R_i^\ell)^T (A_i^\ell)^{-1} R_i^\ell.$$

The index $\ell = 0$ corresponds to the coarsest grid. Note that the subdomains Ω_i^ℓ overlap each other as in the one-level case. A particularly simple algorithm is the multilevel diagonal scaling preconditioner. Then, if the coarsest grid has only a single subdomain

$$M^{-1} = (R^0)^T (A^0)^{-1} R^0 + \sum_{\ell=1}^{L-1} (R^\ell)^T (D^\ell)^{-1} R^\ell + (D^L)^{-1},$$

where D^ℓ is the diagonal of A^ℓ. A closely related preconditioner (known as BPX) was proposed by J.H. Bramble, J.E. Pasciak, and J. Xu [149]. They observed that in finite element methods with linear approximations, the diagonal entries of the matrix at level ℓ must be of order $(h^\ell)^{d-2}$ where h^ℓ is the mesh size and d is the dimension (1, 2 or 3). The BPX preconditioner is defined as

$$M^{-1} = (R^0)^T (A^0)^{-1} R^0 + \sum_{\ell=1}^{L-1} (h^\ell)^{2-d} (R^\ell)^T R^\ell + (h^L)^{2-d} I.$$

The BPX preconditioner discards all information about the coefficients of the problem except for the coarsest mesh. However, it has been proved that it is theoretically optimal for the problems it was designed for, the condition number being $O(1)$.

Additive Schwarz methods can be mixed with multiplicative methods in different ways. One can define as before fully additive methods which are additive among subdomains and between levels. Another possibility is to be multiplicative between subdomains on one level and additive between levels. A third kind of algorithm is being multiplicative between both subdomains and levels.

For multilevel Schwarz methods we can also cite (in alphabetic order), H. Al Daas, L. Grigori, P. Jolivet, and P.H. Tournier [7], H. Al Daas and P. Jolivet [8], H. Al Daas, P. Jolivet, and T. Rees [9], X.-C. Cai and O.B. Widlund [195], T.F. Chan and J. Zou [226], M. Dryja, M.V. Sarkis, and O.B. Widlund [363], G. Migliorati and A. Quarteroni [832], E.E. Prudencio and X.-C. Cai [931], and X. Zhang [1156].

7.8 ▪ Other ideas

Many other ideas have been proposed to construct preconditioners. Let us consider just a few of them. Note that there exist block variants of many of the methods we have described above.

M. Bollhöfer [136, 137] constructed ILU factorizations with pivoting by monitoring the growth of the inverse factors.

In the context of finite element methods, element by element preconditioners were introduced by T.J. Hughes, I. Levit, and J. Winget [660] in 1983.

T.F. Chan and K. Chen [219, 220], T.F. Chan, W.-P. Tang, and W.L. Wan [225], and P.S. Vassilevski and J. Wang [1102] proposed preconditioner based on the use of wavelets.

Preconditioners using low-rank approximations were devised by R. Li and Y. Saad [739], Y. Xi, R. Li, and Y. Saad [1126], N.J. Higham and T. Mary [637], Q. Zheng, Y. Xi, and Y. Saad [1157], and T. Xu, V. Kalantzis, R. Li, Y. Xi, G. Dillon, and Y. Saad [1132].

For the construction of preconditioners, we can cite the following interesting works (in alphabetical order),

O. Axelsson [58, 63], M. Benzi [112], M. Benzi, J.C. Haws, and M. Tůma [115], D. Bertaccini and F. Durastante [126], C.W. Brand [150], S.C. Brenner [156], R. Bru, J. Cerdán, J. Marín, and J. Mas [176], R. Bru, J. Marín, J. Mas and M. Tůma [177], B. Carpentieri, I.S. Duff, and L. Giraud [200], T.F. Chan, C.C.J. Kuo, and C. Tong [222], D. Chen and S. Toledo [241], K. Chen [242], E. Chow and Y. Saad [257], G. Ciaramella and M.J. Gander [263], J.D.F. Cosgrove, J.C. Diaz, and A. Griewank [285], E.F. D'Azevedo, P.A. Forsyth, and W.-P. Tang [320], S. Doi and A. Hoshi [344], Z. Dostál [356], H.C. Elman [416, 417], Y.A. Erlangga and R. Nabben [426, 427], B. Fischer and R.W. Freund [456], J. Frank and C. Vuik [466], R.W. Freund [467], M.J. Gander [488], M.J. Gander, S. Loisel, and D.B. Szyld [489], T. Gergelits, K.A. Mardal, B.F. Nielsen, and Z. Strakoš [519], L. Giraud and S. Gratton [532], N.I. Gould and J.A. Scott [553], L. Grigori and S. Moufawad [577], P. Guillaume, Y. Saad, and M. Sosonkina [579], R. Guo and R.D. Skeel [580], I. Gustafsson and G. Lindskog [586], J. Hrnčíř, I. Pultarová, and Z. Strakoš [652], D. Hysom and A. Pothen [661, 662], I.E. Kaporin [691], J. Kopal, M. Rozložník, and M. Tůma [712], H.-C. Lee and A.J. Wathen [596], J. Màlek and Z. Strakoš [772], J. Mandel [773], T.A. Manteuffel and J.S. Otto [783], G. Meurant [799, 803, 805, 804, 806, 808], M.M. Monga Made and H.A. van der Vorst [835, 836], R. Nabben and C. Vuik [848], B. Nour-Omid and B.N. Parlett [866], P. Novati, M. Redivo-Zaglia, and M.R. Russo [867], D.P. O'Leary [875], M.E. Ong [880], D.V. Perevozkin and G.A. Omarova [913], I. Pultarová and M. Ladecký [932], P. Raghavan and K. Teranishi [935], G.H. Rodrigue and D. Wolitzer [947], Y. Saad [973], V. Simoncini [1010], V. Simoncini and M. Benzi [1012], E. de Sturler and J. Liesen [327], J. Duintjer Tebbens and M. Tůma [389, 390], M. Tismenetsky and I. Efrat [1063, 1064], K. Wang and J. Zhang [1111], J. Zhang [1154, 1153].

7.9 ▪ Numerical experiments

Let us do some numerical experiments with four symmetric matrices and PCG with different preconditioners (listed in Table 7.7) to solve $Ax = b$. In Table 7.8 the first row gives the maximum attainable accuracy of the true residual norm $\|b - Ax_k\|$, the second row gives the number of iterations to obtain $\|r_k\| \leq 10^{-10}\|b\|$, the third row gives the norm of the residual at convergence, and the fourth row gives the number of dot products. The number of matrix-vector products is the number of iterations plus one. We do not give computing times since they are not very reliable in Matlab and depend very much on the coding.

Table 7.7. *Preconditioners*

no	without preconditioning
d	diagonal preconditioner
ic	IC(0)
lv 2	IC with dropping if level > 2
lv 4	IC with dropping if level > 4
ss	SSOR preconditioner with $\omega = 1$
ai 0.025	AINV with parameter = 0.025
ai 0.01	AINV with parameter = 0.01
po 2	least squares polynomial of degree 2
po 4	least squares polynomial of degree 4
ml	standard AMG

Lap 2500 is a linear system which is relatively easy to solve and all the preconditioners work well. The diagonal preconditioner does not give any improvement since the diagonal entries of A are all equal to 4. With IC(0) the number of iterations is divided by three, but the total computing

time, that is, the time for computing the preconditioner plus the time for the PCG iterations, is larger than the time without preconditioning. Keeping some fill-ins in the incomplete factorization decrease the number of iterations, but the time for computing the preconditioner is larger than for IC(0). AMG is very efficient at reducing the number of iterations, but the preconditioner is expensive to compute.

For Pb26 the diagonal preconditioner reduces significantly the number of iterations, which means that the matrix A is badly scaled. We obtain an even better result using IC(0). The polynomial preconditioners do not work well, probably because the estimates of the extreme eigenvalues given by the Gerschgorin disks are not accurate enough.

The same conclusions applied for 1138bus and nos3 except for AINV for which the parameters we use are not well suited for that problem.

For all problems there must be a balance between the time spent for the computation of the preconditioner and the time spent for the iterations. Usually, preconditioners giving a large decrease of the number of iterations are expensive to compute and this can be the bottleneck of the computation when solving a single linear system.

7.10 ▪ Historical and bibliographical comments

Some people believe that preconditioning goes back to the works of C.F. Gauss and C.G.J. Jacobi in the 19th century. In fact, Jacobi used rotations in 1845 to improve the diagonal dominance of the linear system he was considering. In a way this can be interpreted as preconditioning. The first scholar to use preconditioning purposely was L. Cesari in Italy in 1937. He left multiplied the matrix of his system by a polynomial of small order.

The word "preconditioning" appeared in the paper [1078] by A.M. Turing in 1948. We may also consider that preconditioning was already implicitly included in the 1952 paper [629] of M.R. Hestenes and E. Stiefel.

The real development of preconditioners started in the 1960s. One of the pioneers of incomplete factorization was the Russian mathematician N.I. Buleev with the paper [181] published in 1960. Another important early paper was [1097] by R.S. Varga. As noted by Varga, a similar method was proposed by T.A. Oliphant in 1958, with a paper [878] published in 1962. In 1968, T.F. Dupont, R.P. Kendall, and H.H. Rachford proposed an incomplete factorization where the diagonal is modified with a parameter [391]; see also the work of H.L. Stone [1048] that same year.

In 1973, A.D. Tuff and A. Jennings described a method called "partial elimination" in which they discarded some entries based on their values [1076]. J.A. Meijerink and H.A. van der Vorst described in 1977 a general incomplete factorization method [795] for M-matrices. It was the starting point of the popularity of this type of preconditioners. This paper has been widely cited; see also the papers [698, 699] by D.S. Kershaw.

T.A. Manteuffel considered an incomplete Cholesky preconditioner for symmetric positive definite matrices [781, 782] in 1980 by using a shift of the diagonal of the matrix.

R.S. Varga, E.B. Saff, and V. Mehrmann [1100] characterized the matrices that are incompletely factorizable in 1980. A clever way of implementing an LDL^T incomplete factorization in the conjugate gradient algorithm allowing to decrease the number of operations was described by S.C. Eisenstat [402] in 1981. It is known as Eisenstat's trick.

Y. Robert [946] defined a general factorization for general symmetric positive definite matrices A in 1982.

The effect of the ordering of the unknowns on incomplete factorizations used with the conjugate gradient algorithm was investigated experimentally by I.S. Duff and G.M. [377] in 1989.

Table 7.8. *Results for different preconditioners*

precond	Lap 2500	Pb26	1138 bus	nos3
no	$4.7783\ 10^{-13}$ 177 $4.0073\ 10^{-9}$ 534	$1.5668\ 10^{-12}$ 1691 $6.8080\ 10^{-8}$ 5076	$1.0248\ 10^{-10}$ 3484 $1.4620\ 10^{-10}$ 10455	$5.5754\ 10^{-12}$ 283 $2.8382\ 10^{-8}$ 852
d	$4.7783\ 10^{-13}$ 177 $4.0073\ 10^{-9}$ 534	$6.1101\ 10^{-13}$ 226 $7.1218\ 10^{-8}$ 681	$4.6898\ 10^{-11}$ 1084 $8.9962\ 10^{-11}$ 3255	$5.2458\ 10^{-12}$ 246 $2.6049\ 10^{-8}$ 741
ic	$2.9297\ 10^{-13}$ 60 $3.8025\ 10^{-9}$ 183	$5.0460\ 10^{-13}$ 73 $6.0867\ 10^{-8}$ 222	$1.3888\ 10^{-11}$ 161 $6.4047\ 10^{-11}$ 486	$2.5525\ 10^{-12}$ 53 $1.0105\ 10^{-8}$ 162
lv 2	$5.0460\ 10^{-13}$ 39 $4.1487\ 10^{-9}$ 120	$3.1321\ 10^{-13}$ 45 $7.1720\ 10^{-8}$ 138	$1.2253\ 10^{-11}$ 75 $8.1401\ 10^{-11}$ 228	$2.6000\ 10^{-12}$ 50 $1.4711\ 10^{-8}$ 153
lv 4	$2.1969\ 10^{-13}$ 34 $2.3780\ 10^{-9}$ 105	$2.6674\ 10^{-13}$ 39 $5.1162\ 10^{-8}$ 120	$1.0102\ 10^{-11}$ 40 $3.1397\ 10^{-11}$ 123	$2.4162\ 10^{-12}$ 41 $1.3875\ 10^{-8}$ 126
ss	$3.1628\ 10^{-13}$ 71 $3.5797\ 10^{-9}$ 216	$3.8608\ 10^{-13}$ 85 $6.7531\ 10^{-8}$ 258	$2.8148\ 10^{-11}$ 535 $6.1641\ 10^{-11}$ 1608	$3.2202\ 10^{-12}$ 81 $3.0500\ 10^{-8}$ 246
ai 0.025	$3.0860\ 10^{-13}$ 63 $2.6316\ 10^{-9}$ 192	$3.7243\ 10^{-13}$ 123 $6.4265\ 10^{-8}$ 372	$5.4836\ 10^{-11}$ 1151 $1.0066\ 10^{-10}$ 3456	$5.1062\ 10^{-12}$ 246 $2.5788\ 10^{-8}$ 741
ai 0.01	$2.2472\ 10^{-13}$ 39 $2.6839\ 10^{-9}$ 120	$3.0726\ 10^{-13}$ 75 $6.2959\ 10^{-8}$ 228	$5.2953\ 10^{-11}$ 1169 $8.2241\ 10^{-11}$ 3510	$5.3603\ 10^{-12}$ 246 $2.5917\ 10^{-8}$ 741
po 2	$2.8945\ 10^{-13}$ 69 $4.3581\ 10^{-9}$ 210	$1.5208\ 10^{-12}$ 946 $7.2717\ 10^{-8}$ 2841	$6.3778\ 10^{-11}$ 2212 $9.2382\ 10^{-11}$ 6639	$1.9568\ 10^{-11}$ 2478 $3.0597\ 10^{-8}$ 7437
po 4	$2.4864\ 10^{-13}$ 43 $4.7076\ 10^{-9}$ 132	$1.2171\ 10^{-12}$ 612 $6.5057\ 10^{-8}$ 1839	$5.7113\ 10^{-11}$ 1595 $9.7010\ 10^{-11}$ 4788	$1.7234\ 10^{-11}$ 1512 $3.1617\ 10^{-8}$ 4539
ml	$1.0696\ 10^{-13}$ 6 $2.0735\ 10^{-9}$ 21	$1.8497\ 10^{-13}$ 11 $1.2687\ 10^{-8}$ 36	$8.7625\ 10^{-12}$ 37 $5.6231\ 10^{-11}$ 114	$1.8959\ 10^{-12}$ 19 $1.5651\ 10^{-8}$ 60

These experimental results have been explained theoretically to some extent by V. Eijkhout [401], S. Doi [343], and S. Doi and A. Lichnewsky [345, 346] in 1991.

In 1994, Y. Saad considered several techniques for incomplete factorizations in [976]. One was to keep fill-ins according to their level. Another possibility was to drop fill-ins according to their value, but to keep only a given number of the largest ones in modulus to control the storage for the preconditioner.

That same year, O. Axelsson published a book about iterative methods [64] in which he described and studied several preconditioners.

Incomplete factorizations were still a topic of research in the 2000s, even though the basic principles were known since the 1960s. For instance, M. Bollhöfer presented in 2001 an ILU factorization with a refined dropping strategy that monitored the growth of the inverse factors of L and U [136, 137] and R. Bru, J. Marín, J. Mas, and M. Tůma [177] introduced balanced incomplete factorizations in 2008. They computed the triangular factors and their inverses at the same time. In 2015, E. Chow and A. Patel [255] described a parallel ILU algorithm which was very different from existing approaches.

When the matrix has a block structure, efficient incomplete factorizations can be developed taking advantage of that structure. This was done by several authors in the 1980s and 1990s. In 1985, following the work of R.R. Underwood, P. Concus, G.H. Golub, and G.M. [273] proposed several block incomplete factorization methods for block tridiagonal matrices; see also [274]. Another proponent of block factorizations was O. Axelsson, who published many papers on that topic; see [67, 60, 61, 63, 72, 73]. Approximate block factorizations were also proposed by R. Beauwens and M. Ben Bouzid [106, 107] in 1987-1988.

Using the SSOR iteration matrix as a preconditioner for symmetric positive definite matrices was proposed by D.J. Evans [430] in 1967; see also the book [431]. SSOR was also considered by O. Axelsson [56] in 1974.

As we wrote in this chapter, a polynomial preconditioner of small degree was used in 1937 by L. Cesari [211, 212, 213, 214]. In 1979, P.F. Dubois, A. Greenbaum, and G.H. Rodrigue [367] used a Neumann series as a polynomial preconditioner. O.G. Johnson, C.A. Micchelli, and G. Paul [684] defined a generalized condition number in 1983, and used the polynomial minimizing this condition number. Least squares polynomial preconditioners for symmetric matrices were mainly studied by Y. Saad [973]. Another polynomial preconditioner for CG was proposed by D.P. O'Leary [875] in 1991. She used the polynomial defined by the conjugate gradient iteration in an adaptive recursive procedure on an already preconditioned system.

About polynomial preconditioners for nonsymmetric matrices we can cite the works of T.A Manteuffel [779, 780] in 1977, H.C. Elman, Y. Saad, and P.E. Saylor [419], H.C. Elman and Streit [420] in 1986, Y. Saad [974] in 1987, and D.C. Smolarski and P.E. Saylor [992] in 1991. We can also refer to the Ph.D. thesis of S.F. Ashby [45].

The computation of approximate inverses as precontioners started in the 1990s. Norm minimization was used by M.J. Grote and H.D. Simon [578] in 1993, T. Huckle and M.J. Grote [657, 658] in 1994, and L.Y. Kolotilina and A.Yu. Yeremin [710]. Improvements of the method of Huckle and Grote were described by N.I.M. Gould and J.A. Scott [553] in 1995-1998.

Another approach using A-conjugate vectors was developed by M. Benzi, C.D. Meyer, and M. Tůma [117] in 1996; see also [120, 119, 121, 122, 114].

Interesting documents about preconditioning are the review article by M. Benzi [112] in 2002 and the book [242] by K. Chen in 2005.

As we wrote above, it is generally admitted that the first ever domain decomposition method was due to H.A. Schwarz in August 1870. Domain decomposition methods came back to life in the 1980s. The first Domain Decomposition Conference was organized in Paris in 1987. The first paper in the proceedings [536] was by P.L. Lions who gave a nice proof of convergence of

the alternating Schwarz method; see also [752, 753]. For the history of Schwarz methods and a summary of some variants, see [487] by M.J. Gander.

Since the 1980s thousands of papers have been written about domain decomposition methods. Besides the conference proceedings, a few books were devoted to DD methods, [1026] by B.F. Smith, P. Bjørstad, and W.D. Gropp in 1996, [934] by A. Quarteroni and A. Valli in 1999, [1069] by A. Toselli and O.B. Widlund in 2004, [347] by V. Dolean, P. Jolivet, and F. Nataf in 2015, and [263] by G. Ciaramella and M.J. Gander.

Multigrid methods were developed in the USSR in the 1960s for solving linear systems arising from discretization of partial differential equations. Such a method was proposed by R.P. Fedorenko who published three papers on this topic [445, 446, 447] in 1961, 1964, and 1973. Multigrid methods were popularized by A. Brandt who was one of the first to recognize their potential in the 1970s by solving many different and difficult problems; see [151, 152, 153, 154, 155].

Many papers were written on multigrid in the 1980s and 1990s. Multigrid convergence for indefinite problems [857] was studied by R.A Nicolaides in 1978; see also [856, 858]. Convergence of multigrid iterations applied to difference equations was the topic of a paper [598] by W. Hackbusch in 1980. K. Stüben and U. Trottenberg published in 1982 a study of multigrid methods [602]. Convergence estimates for multigrid algorithms without regularity assumptions were given by J.H. Bramble, J.E. Pasciak, J.P. Wang, and J. Xu [148] in 1991. A unified convergence theory for multigrid or multilevel algorithms [359] was proposed by C.C. Douglas in 1993; see also [357, 358].

An important step was the introduction of algebraic multigrid in 1985 by J.W. Ruge and K. Stüben [964, 965].

A few books were devoted to multigrid in the 1980s and 1990s: [599] by W. Hackbusch in 1985, [172] by W.L. Briggs in 1987, an updated and extended version [173] was published by W.L. Briggs, V.E. Henson, and S.F. McCormick in 2000, [793] edited by S.F. McCormick in 1987, [1114] by P. Wesseling in 1992, [143] by J.H. Bramble in 1993, and [1074] by U. Trottenberg, C.W. Oosterlee, and A. Schüller in 2000.

Algebraic multilevel preconditioning methods were studied in 1989 and 1990 by O. Axelsson and P.S. Vassilevski [74, 76]. A multilevel block incomplete factorization preconditioning [861] was introduced by Y. Notay in 1999. He published a robust parameter-free algebraic multilevel preconditioner [863] in 2002. That same year an algebraic recursive multilevel solver (ARMS) for general sparse linear systems [981] was proposed by Y. Saad and B. Suchomel. A parallel version (pARMS) [744] was published by Z. Li, Y. Saad, and M. Sosonkina in 2003; see also [978] by Y. Saad in 2005.

Aggregation-based algebraic multilevel preconditioners [864] were studied by Y. Notay in 2006. Multilevel preconditioners constructed from inverse-based ILU factorizations [141] were proposed by M. Bollhöfer and Y. Saad in 2006.

Appendix: Test matrices

The matrices are ordered by increasing condition number; *nnz* is the number of nonzero entries, $\kappa(A)$ is the condition number (computed by Matlab), *min svd* is the smallest singular value of A and *fov* tells that 0 is inside the field of values when it is equal to 1. The last column tells if the matrix is normal. In fact, all the normal matrices in the list are symmetric and denoted by an "s". All these matrices can be obtained from the SuiteSparse matrix collection `https://sparse.tamu.edu`.

Table 9. *Properties of test matrices*

Matrix	n	nnz	$\kappa(A)$	min svd	fov	normal?
pde225	225	1065	3.90638e+01	2.50580e-01	0	-
gre 343	343	1310	1.11976e+02	9.01565e-03	1	-
jpwh 991	991	6027	1.42045e+02	1.14696e-01	0	-
pde2961	2961	14585	6.42493e+02	1.61532e-02	1	-
jagmesh1	936	6264	1.22765e+03	5.62507e-03	1	s
bfwa782	782	7514	1.74061e+03	7.22416e-03	1	-
dw2048	2048	10114	2.09321e+03	4.67610e-04	1	-
jagmesh2	1009	6865	2.36649e+03	2.94509e-03	1	s
lshp1009	1009	6865	2.36649e+03	2.94509e-03	1	s
raefsky2	3242	293551	4.25194e+03	8.76027e-04	1	-
fs 680 1c	680	2184	8.69443e+03	4.38996e-04	1	-
add20	2395	13151	1.20471e+04	5.99434e-05	0	-
raefsky1	3242	293409	1.28851e+04	2.87892e-04	1	-
jagmesh4	1440	9504	1.51630e+04	4.52739e-04	1	s
fs 680 1	680	2184	1.54052e+04	4.66822e+09	1	-
sherman1	1000	3750	1.55953e+04	3.23487e-04	0	-
nos3	960	15844	3.77236e+04	1.82884e-02	0	s
sherman5	3312	20793	1.87941e+05	2.41965e-02	1	-
cavity05	1182	32632	5.77065e+05	2.25403e-05	0	-
e05r0500	236	5846	1.15887e+06	4.93622e-05	1	-
comsol	1500	97645	1.41522e+06	4.47174e-07	0	-
olm1000	1000	3996	1.48722e+06	6.19384e-02	1	-
cavity10	2597	76171	2.95507e+06	4.41707e-06	0	-
steam2	600	5660	3.78313e+06	1.23855e+03	0	-
1138bus	1138	4054	8.57265e+06	3.51686e-03	0	s
steam1	240	2248	2.82750e+07	7.67886e-01	0	-
bcsstk26	1922	30336	1.65934e+08	9.53833e+02	0	s
nos7	729	4617	2.37451e+09	4.15413e-03	0	s
watt1	1856	11360	4.35964e+09	2.29377e-10	1	-
bcsstk14	1806	63454	1.19232e+10	9.99998e-01	0	-
fs 183 6	183	1000	1.73678e+11	6.79901e-03	1	-
bcsstk20	485	3135	3.89279e+12	3.24066e+03	0	s
mcfe	765	24382	4.24182e+13	3.17699e+03	1	-
nnc	261	1500	2.90962e+14	3.57651e-12	1	-
lnsp	511	2796	3.32894e+15	1.26332e-05	1	-

Table 10. *Symmetric test matrices*

Name	n	nnz	$\lambda_{\min}$	$\lambda_{\max}$	$\kappa(A)$	Rhs
Lap2500	2500	12300	$7.586685051\ 10^{-3}$	7.992413314	$1.053478991\ 10^{3}$	random
ash292	292	2208	2.9944579613	9.6312493108	3.2163581641	random
bcsstk01	48	400	3417.267562666	$3.0151790899\ 10^{9}$	$8.823362627\ 10^{5}$	$\frac{\text{ones}(48,1)}{\sqrt{48}}$
bcsstk01I	48	2304	$3.316552583\ 10^{-10}$	$2.9263146115\ 10^{-4}$	$8.823362627\ 10^{5}$	x_{ex}
bcsstk09	1083	18437	$7.102229057\ 10^{3}$	$6.760303645\ 10^{7}$	$9.51856606\ 10^{3}$	random
Pb26	6400	31680	$1.189952486\ 10^{-3}$	167.2150583486	$1.405224665\ 10^{5}$	$A*$random
1138_bus	1138	4054	$3.516860\ 10^{-3}$	$3.014879\ 10^{4}$	$8.572646\ 10^{6}$	random

Bibliography

[1] J. O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971. (Cited on page 107.)

[2] K. Abe. On convergence speed of parallel variants of BiCGSTAB for solving linear equations. In *Asian Simulation Conference*, pages 401–413, Singapore, 2018. Springer. (Cited on page 344.)

[3] L. M. Adams and H. F. Jordan. Is SOR color blind? *SIAM J. Sci. Statist. Comput.*, 7(2):490–506, 1986. (Cited on page 238.)

[4] G. Advelas and A. Hadjidimos. Optimum accelerated overrelaxation method in a special case. *Math. Comput.*, 36(153):183–187, 1981. (Cited on page 238.)

[5] A. A. Ahac, J. J. Buoni, and D. D. Olesky. Stable LU factorization of H-matrices. *Linear Algebra Appl.*, 99:97–110, 1988. (Cited on page 109.)

[6] A. A. Ahac and D. D. Olesky. A stable method for the LU factorization of M-matrices. *SIAM J. Alg. Disc. Meth.*, 7(3):368–378, 1986. (Cited on page 109.)

[7] H. Al Daas, L. Grigori, P. Jolivet, and P. H. Tournier. A multilevel Schwarz preconditioner based on a hierarchy of robust coarse spaces. *SIAM J. Sci. Comput.*, 43(3):A1907–A1928, 2021. (Cited on page 416.)

[8] H. Al Daas and P. Jolivet. A robust algebraic multilevel domain decomposition preconditioner for sparse symmetric positive definite matrices. *SIAM J. Sci. Comput.*, 44(4):A2582–A2598, 2022. (Cited on page 416.)

[9] H. Al Daas, P. Jolivet, and T. Rees. Efficient algebraic two-level Schwarz preconditioner for sparse matrices. *SIAM J. Sci. Comput.*, 45(3):A1199–A1213, 2023. (Cited on page 416.)

[10] G. Alefeld. Zur konvergenz des Peaceman-Rachford verfahrens. *Numer. Math.*, 26:409–419, 1976. (Cited on page 243.)

[11] G. Alefeld and R. S. Varga. Zur konvergenz des symmetrischen relaxationsverfahrens. *Numer. Math.*, 25:291–295, 1976. (Cited on page 238.)

[12] J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, E. Dufrechou, P. Ezzatti, and E. S. Quintana-Ortí. Exploiting task and data parallelism in ILUPACK's preconditioned CG solver on NUMA architectures and many-core accelerators. *Parallel Comput.*, 54:97–107, 2016. (Cited on page 373.)

[13] Z. Allen-Zhu, Y. Li, R. Oliveira, and A. Wigderson. Much faster algorithms for matrix scaling. In *58th Annual Symposium on Foundations of Computer Science*, pages 890–901. IEEE, 2017. (Cited on page 128.)

[14] R. Alomairy, M. Gates, S. Cayrols, D. Sukkari, K. Akbudak, A. YarKhan, P. Bagwell, and J. J. Dongarra. Communication avoiding LU with tournament pivoting in SLATE. Technical Report SLATE Working Note 18, University of Tennessee, 2022. (Cited on page 145.)

[15] H. Alqahtani and L. Reichel. Simplified anti-Gauss quadrature rules with applications in linear algebra. *Numer. Algorithms*, 77:577–602, 2018. (Cited on page 284.)

[16] H. Alqahtani and L. Reichel. Generalized block anti-Gauss quadrature rules. *Numer. Math.*, 143:605–648, 2019. (Cited on page 284.)

[17] S. C. Althoen and R. McLaughlin. Gauss-Jordan reduction: A brief history. *Am. Math. Mon.*, 94(2):130–142, 1987. (Cited on pages 146 and 182.)

[18] F. L. Alvarado, A. Pothen, and R. S. Schreiber. Highly parallel sparse triangular solution. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph theory and sparse matrix computation*, pages 141–157. Springer, New York, 1993. (Cited on page 186.)

[19] F. L. Alvarado and R. S. Schreiber. Optimal parallel solution of sparse triangular systems. *SIAM J. Sci. Comput.*, 14(2):446–460, 1993. (Cited on page 186.)

[20] F. L. Alvarado, D. C. Yu, and R. Betancourt. Partitioned sparse A^{-1} methods. *IEEE Trans. Power Systems*, 5(2):452–459, 1990. (Cited on page 186.)

[21] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J. Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.*, 37(3):A1451–A1474, 2015. (Cited on page 223.)

[22] P. Amestoy, O. Boiteau, A. Buttari, M. Gerest, F. Jézéquel, J. Y. L'Excellent, and T. Mary. Mixed precision low rank approximations and their application to block low rank LU factorization. HAL-03251738v3, 2022. (Cited on page 223.)

[23] P. R. Amestoy. *Factorisation de grandes matrices creuses non symétriques basée sur une méthode multifrontale dans un environnement multiprocesseur*. PhD thesis, ENSEEIHT, 1990. CERFACS Report TH/PA/91/2. (Cited on page 216.)

[24] P. R. Amestoy, A. Buttari, N. J. Higham, J. Y. L'Excellent, T. Mary, and B. Vieublé. Five-precision GMRES-based iterative refinement. Technical Report MIMS Eprint 2021.5, University of Manchester, 2021. (Cited on page 129.)

[25] P. R. Amestoy, A. Buttari, J. Y. L'Excellent, and T. Mary. Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format. *SIAM J. Sci. Comput.*, 41(3):A1414–A1442, 2019. (Cited on page 223.)

[26] P. R. Amestoy, A. Buttari, J. Y. L'Excellent, and T. Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Trans. Math. Soft. (TOMS)*, 45(1):article 2, 2019. (Cited on page 223.)

[27] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996. (Cited on page 205.)

[28] P. R. Amestoy, I. S. Duff, A. Guermouche, and T. Slavova. Analysis of the solution phase of a parallel multifrontal solver. *Parallel Comput.*, 36:3–15, 2010. (Cited on page 186.)

[29] P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Engrg.*, 184(2-4):501–520, 2000. (Cited on page 221.)

[30] P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, 2001. (Cited on page 221.)

[31] P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Trans. Math. Soft. (TOMS)*, 27(4):388–421, 2001. (Cited on page 222.)

[32] P. R. Amestoy, I. S. Duff, D. Ruiz, and B. Uçar. A parallel matrix scaling algorithm. In *International Conference on High Performance Computing for Computational Science*, pages 301–313, Berlin, Heidelberg, 2008. Springer. (Cited on page 128.)

[33] P. R. Amestoy, A. Guermouche, J. Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Comput.*, 32(2):136–156, 2006. (Cited on page 222.)

[34] P. R. Amestoy and C. Puglisi. An unsymmetrized multifrontal LU factorization. *SIAM J. Matrix Anal. Appl.*, 24(2):553–569, 2002. (Cited on page 219.)

[35] E. Anderson and Y. Saad. Solving sparse triangular linear systems on parallel computers. *Intl. J. High Speed Computing*, 1(1):73–95, 1989. (Cited on page 186.)

[36] R. S. Anderssen and G. H. Golub. Richardson's non-stationary matrix iterative procedure. Technical Report STAN-CS-72-304, Computer Science Dept., Stanford University, 1972. (Cited on page 248.)

[37] M. Angelaccio and M. Clajanni. Subcube matrix decomposition, a unifying view for LU factorization on multicomputers. *Parallel Comput.*, 20(2):257–270, 1994. (Cited on page 141.)

[38] H. Anzt, E. Chow, and J. J. Dongarra. ParILUT-a new parallel threshold ILU factorization. *SIAM J. Sci. Comput.*, 40(4):C503–C519, 2018. (Cited on page 377.)

[39] H. Anzt, J. J. Dongarra, G. Flegar, and E. S. Quintana-Ortí. Variable-size batched Gauss-Jordan elimination for block-Jacobi preconditioning on graphics processors. *Parallel Comput.*, 81:131–146, 2019. (Cited on page 150.)

[40] M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.*, 10(2):165–190, 1989. (Cited on pages 217, 219, and 220.)

[41] M. Arioli, V. Pták, and Z. Strakoš. Krylov sequences of maximal length and convergence of GMRES. *BIT Numerical Mathematics*, 38(4):636–643, 1998. (Cited on page 359.)

[42] M. Arioli and J. Scott. Chebyshev acceleration of iterative refinement. *Numer. Algorithms*, 66(3):591–608, 2014. (Cited on page 129.)

[43] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951. (Cited on page 358.)

[44] E. Asenjo, M. Ujaldón, and E. L. Zapata. Parallel WZ factorization on mesh multiprocessors. *Microprocess. Microprogram.*, 38(1-5):319–326, 1993. (Cited on page 167.)

[45] S. F. Ashby. *Polynomial preconditioning for conjugate gradient methods*. PhD thesis, Dept. of Computer Science, University of Illinois, 1987. (Cited on pages 388, 389, and 420.)

[46] S. F. Ashby. Minimax polynomial preconditioning for Hermitian linear systems. *SIAM J. Matrix Anal. Appl.*, 12(4):766–789, 1991. (Cited on page 383.)

[47] S. F. Ashby, T. A. Manteuffel, and J. S. Otto. A comparison of adaptive Chebyshev and least squares polynomial preconditioning for Hermitian positive definite linear systems. *SIAM J. Sci. Statist. Comput.*, 13(1):1–29, 1991. (Cited on page 389.)

[48] C. C. Ashcraft. A taxonomy of distributed dense LU factorization methods. Technical Report ECA-TR-161, Boeing Computer Services, 1991. (Cited on page 142.)

[49] C. C. Ashcraft. *A taxonomy of column-based Cholesky factorizations*. PhD thesis, Yale University, 1995. (Cited on page 142.)

[50] C. C. Ashcraft, I. S. Duff, J. Hogg, J. A. Scott, and S. Thorne. Nested dissection revisited. Technical Report RAL-TR-2016-004, Rutherford Appleton Lboratory, 2016. (Cited on page 207.)

[51] C. C. Ashcraft, S. C. Eisenstat, and J. W. H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM J. Sci. Statist. Comput.*, 11(3):593–599, 1990. (Cited on page 221.)

[52] C. C. Ashcraft, R. G. Grimes, J. G. Lewis, B. W. Peyton, and H. D. Simon. Progress in sparse matrix methods for large linear systems on vector supercomputers. *Intl. J. Supercomp. Appl.*, 1(4):10–30, 1987. (Cited on pages 211 and 224.)

[53] C. C. Ashcraft and J. W. H. Liu. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. *SIAM J. Matrix Anal. Appl.*, 19(2):325–354, 1998. (Cited on pages 209 and 210.)

[54] C. C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Appl.*, 19(3):816–832, 1998. (Cited on pages 210 and 211.)

[55] E. Asplund. Inverses of matrices $\{a_{ij}\}$ which satisfy $a_{ij} = 0$ for $j > i+p$. *Math. Scand.*, 7:57–60, 1959. (Cited on pages 68 and 79.)

[56] O. Axelsson. On preconditioning and convergence acceleration for sparse matrix problems. Technical Report Report 74-10, CERN, Geneva, Switzerland, 1974. (Cited on pages 365 and 420.)

[57] O. Axelsson. A class of iterative methods for finite element equations. *Comput. Methods Appl. Mech. Engrg.*, 9:123–137, 1976. (Cited on pages 273, 303, 365, and 366.)

[58] O. Axelsson. Preconditioning of indefinite problems by regularization. *SIAM J. Numer. Anal.*, 16(1):58–69, 1979. (Cited on page 417.)

[59] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations. *Linear Algebra Appl.*, 29:1–16, 1980. (Cited on page 329.)

[60] O. Axelsson. Incomplete block matrix factorization preconditioning methods. The ultimate answer? *J. Comput. Appl. Math.*, 12:3–18, 1985. (Cited on page 420.)

[61] O. Axelsson. A general incomplete block matrix factorization method. *Linear Algebra Appl.*, 74:179–190, 1986. (Cited on pages 378 and 420.)

[62] O. Axelsson. A generalized conjugate gradient, least square method. *Numer. Math.*, 51(2):209–227, 1987. (Cited on page 329.)

[63] O. Axelsson. Preconditoning methods for block H-matrices. In E. Spedicato, editor, *Computer Algorithms for Solving Linear Algebraic Equations, the State of the Art*, pages 169–184. Springer, 1991. NATO ASI F, volume 77. (Cited on pages 417 and 420.)

[64] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994. (Cited on pages 3, 371, 378, and 420.)

[65] O. Axelsson. Stabilization of algebraic multilevel iteration; additive methods. In O. Axelsson and B. Polman, editors, *AMLI'96: Proceedings of the Conference on Algebraic Multilevel Iteration Methods with Applications*, pages 49–62. University of Nijmegen, 1996. (Cited on page 415.)

[66] O. Axelsson. A survey of algebraic multilevel iteration (AMLI) methods. *BIT Numerical Mathematics*, 43(5):863–879, 2003. (Cited on page 415.)

[67] O. Axelsson, S. Brinkkemper, and V. P. Il'in. On some versions of incomplete block matrix factorization iterative methods. *Linear Algebra Appl.*, 58:3–15, 1984. (Cited on page 420.)

[68] O. Axelsson and V. Eijkhout. The nested recursive two level factorization method for nine point difference matrices. *SIAM J. Sci. Statist. Comput.*, 12:1373–1400, 1991. (Cited on page 415.)

[69] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned conjugate gradient method. *Numer. Math.*, 48:499–523, 1986. (Cited on pages 273 and 276.)

[70] O. Axelsson and H. Lu. A survey of some estimates of eigenvalues and condition numbers for certain preconditioned matrices. *J. Comput. Appl. Math.*, 80:241–264, 1994. (Cited on page 378.)

[71] O. Axelsson and S. Margenov. On multilevel preconditioners which are optimal with respect to both problem and discretization parameters. *Comput. Methods Appl. Math.*, 3(1):6–22, 2003. (Cited on page 415.)

[72] O. Axelsson and B. Polman. On approximate factorization methods for block matrices suitable for vector and parallel processors. *Linear Algebra Appl.*, 77:3–26, 1986. (Cited on pages 400 and 420.)

[73] O. Axelsson and B. Polman. Block preconditioning and domain decomposition methods, II. *J. Comput. Appl. Math.*, 24:55–72, 1988. (Cited on pages 400 and 420.)

[74] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods, I. *Numer. Math.*, 56:157–177, 1989. (Cited on pages 415 and 421.)

[75] O. Axelsson and P. S. Vassilevski. A survey of multilevel preconditioned iterative methods. *BIT Numerical Mathematics*, 29(4):769–793, 1989. (Cited on page 415.)

[76] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods, II. *SIAM J. Numer. Anal.*, 27(6):1569–1590, 1990. (Cited on pages 415 and 421.)

[77] O. Axelsson and P. S. Vassilevski. Algebraic multilevel preconditioning methods III. In R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and O. B. Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 163–177. SIAM, 1991. (Cited on page 415.)

[78] M. Baboulin, S. Donfack, J. J. Dongarra, L. Grigori, A. Rémy, and S. Tomov. A class of communication-avoiding algorithms for solving general dense linear systems on CPU/GPU parallel machines. *Procedia Comput. Sci.*, 17-26:17–26, 2012. (Cited on page 144.)

[79] M. Baboulin, J. J. Dongarra, J. Herrmann, and S. Tomov. Accelerating linear system solutions using randomization techniques. *ACM Trans. Math. Soft. (TOMS)*, 39(2):1–13, 2013. (Cited on page 96.)

[80] Z. Bai. Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem. *Math. Comput.*, 62(205):209–226, 1994. (Cited on page 337.)

[81] Z. Z. Bai. A class of hybrid algebraic multilevel preconditioning methods. *Appl. Numer. Math.*, 19:389–399, 1996. (Cited on page 415.)

[82] Z.-Z. Bai. Optimal parameters in the HSS-like methods for saddle-point problems. *Numer. Linear Algebra Appl.*, 16(6):447–479, 2009. (Cited on page 244.)

[83] Z.-Z. Bai, G. H. Golub, and C.-K. Li. Optimal parameter in Hermitian and skew-Hermitian splitting method for certain two-by-two block matrices. *SIAM J. Sci. Comput.*, 28(2):583–603, 2006. (Cited on page 244.)

[84] Z.-Z. Bai, G. H. Golub, L. Z. Lu, and J. F. Yin. Block triangular and skew-Hermitian splitting methods for positive-definite linear systems. *SIAM J. Sci. Comput.*, 26(3):844–863, 2005. (Cited on page 244.)

[85] Z.-Z. Bai, G. H. Golub, and M. K. Ng. Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems. *SIAM J. Matrix Anal. Appl.*, 24(3):603–626, 2003. (Cited on page 244.)

[86] Z.-Z. Bai, G. H. Golub, and M. K. Ng. On successive-overrelaxation acceleration of the Hermitian and skew-Hermitian splitting iterations. *Numer. Linear Algebra Appl.*, 14(4):319–335, 2007. (Cited on page 244.)

[87] A. H. Baker, J. M. Dennis, and E. R. Jessup. An efficient block variant of GMRES. Technical Report CU-CS-957-03, University of Colorado, Boulder (USA), 2003. (Cited on page 360.)

[88] A. H. Baker, J. M. Dennis, and E. R. Jessup. On improving linear solver performance: A block variant of GMRES. *SIAM J. Sci. Comput.*, 27(5):1608–1626, 2006. (Cited on page 360.)

[89] A. H. Baker, E. R. Jessup, and T. V. Kolev. A simple strategy for varying the restart parameter in GMRES(m). *J. Comput. Appl. Math.*, 230(2):751–761, 2009. (Cited on page 324.)

[90] A. H. Baker, E. R. Jessup, and T. A. Manteuffel. A technique for accelerating the convergence of restarted GMRES. *SIAM J. Matrix Anal. Appl.*, 26(4):962–984, 2005. (Cited on pages 324 and 359.)

[91] G. Ballard, E. Carson, J. W. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 2014. (Cited on page 338.)

[92] G. Ballard, J. W. Demmel, B. Lipshitz, O. Schwartz, and S. Toledo. Communication efficient Gaussian elimination with partial pivoting using a shape morphing data layout. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 232–240. ACM, 2013. (Cited on page 136.)

[93] T. Banachiewicz. Méthode de résolution numérique des équations linéaires, du calcul des déterminants et des inverses, et de réduction des formes quadratiques. *Bulletin international de l'Académie polonaise des sciences et des lettres, Classe des sciences mathématiques et naturelles. Série A*, pages 393–404, 1938. (Cited on page 183.)

[94] R. E. Bank and R. K. Smith. General sparse elimination requires no permanent integer storage. *SIAM J. Sci. Statist. Comput.*, 8(4):574–584, 1987. (Cited on page 193.)

[95] R. E. Bank and R. K. Smith. The incomplete factorization multigraph algorithm. *SIAM J. Sci. Comput.*, 20(4):1349–1364, 1999. (Cited on page 415.)

[96] R. E. Bank and C. Wagner. Multilevel ILU decomposition. *Numer. Math.*, 82(4):543–576, 1999. (Cited on page 415.)

[97] J. Baranger and M. Duc-Jacquet. Matrices tridiagonales symétriques et matrices factorisables. *RIRO*, 3:61–66, 1971. (Cited on pages 70, 79, and 111.)

[98] S. T. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numer. Linear Algebra Appl.*, 2(4):317–334, 1995. (Cited on pages 202 and 224.)

[99] W. W. Barrett and P. J. Feinsilver. Inverses of banded matrices. *Linear Algebra Appl.*, 41:111–130, 1981. (Cited on page 68.)

[100] M. S. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *Ann. Math. Stat.*, 22:107–111, 1951. (Cited on page 73.)

[101] D. Bashir, H. Kamarulhailia, and O. Babarinsa. A review on Quadrant Interlocking factorization: WZ and WH factorization. *J. Nig. Soc. Phys. Sci.*, 5:1112, 2023. (Cited on page 168.)

[102] F. L. Bauer. Optimally scaled matrices. *Numer. Math.*, 5:73–87, 1963. (Cited on page 128.)

[103] F. L. Bauer. Remarks on optimally scaled matrices. *Numer. Math.*, 13(1):1–3, 1969. (Cited on page 128.)

[104] F. L. Bauer and C. T. Fike. Norms and exclusion theorems. *Numer. Math.*, 2(1):137–141, 1960. (Cited on page 78.)

[105] F. L. Bauer and A. S. Householder. Some inequalities involving the Euclidean condition of a matrix. *Numer. Math.*, 2(1):308–311, 1960. (Cited on page 78.)

[106] R. Beauwens and M. Ben Bouzid. On sparse block factorization iterative methods. *SIAM J. Numer. Anal.*, 24(5):1066–1076, 1987. (Cited on page 420.)

[107] R. Beauwens and M. Ben Bouzid. Existence and conditioning properties of sparse approximate block factorizations. *SIAM J. Numer. Anal.*, 25(4):941–956, 1988. (Cited on page 420.)

[108] M. Bellalij, K. Jbilou, and H. Sadok. New convergence results on the global GMRES method for diagonalizable matrices. *J. Comput. Appl. Math.*, 219:350–358, 2008. (Cited on page 311.)

[109] P. Benner, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. Revisiting the Gauss-Huard algorithm for the solution of linear systems on graphics accelerators. In *Parallel Processing and Applied Mathematics: 11th International Conference*, pages 505–514. Springer, 2016. (Cited on page 153.)

[110] Cdt. E. Benoît. Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés à un système d'équations linéaires en nombre inférieur à celui des inconnues, (Procédé du commandant Cholesky). *Bull. Géodésique*, 2:67–77, 1924. (Cited on page 183.)

[111] M. Benzi. *Row-projection method for sparse linear systems*. PhD thesis, Department of Mathematics, North Carolina State University, 1993. (Cited on page 158.)

[112] M. Benzi. Preconditioning techniques for large linear systems: A survey. *J. Comp. Phys.*, 182(2):418–477, 2002. (Cited on pages 417 and 420.)

[113] M. Benzi. A generalization of the Hermitian and skew-Hermitian splitting iteration. *SIAM J. Matrix Anal. Appl.*, 31(2):360–374, 2009. (Cited on page 244.)

[114] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22(4):1318–1332, 2000. (Cited on pages 380, 412, and 420.)

[115] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22(4):1333–1353, 2000. (Cited on page 417.)

[116] M. Benzi and C. D. Meyer. A direct projection method for sparse linear systems. *SIAM J. Sci. Comput.*, 16(5):1159–1176, 1995. (Cited on pages 158 and 160.)

[117] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17(5):1135–1149, 1996. (Cited on pages 160, 380, 412, and 420.)

[118] M. Benzi, D. B. Szyld, and A. Van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM J. Sci. Comput.*, 20(5):1652–1670, 1999. (Cited on page 377.)

[119] M. Benzi and M. Tůma. Numerical experiments with two approximate inverse preconditioners. *BIT Numerical Mathematics*, 38(2):234–241, 1998. (Cited on pages 382 and 420.)

[120] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998. (Cited on pages 160, 381, and 420.)

[121] M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30(2-3):305–340, 1999. (Cited on pages 382 and 420.)

[122] M. Benzi and M. Tůma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21(5):1851–1868, 2000. (Cited on pages 382 and 420.)

[123] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, 1979. Reprinted by SIAM. (Cited on pages 3, 58, 59, 78, 108, and 109.)

[124] P. Berman and G. Schnitger. On the performance of the minimum degree ordering for Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(1):83–89, 1990. (Cited on page 204.)

[125] R. D. Berry. An optimal ordering of electronic circuit equations for a sparse matrix solution. *IEEE Trans. Circuit Theory*, 19(1):40–50, 1971. (Cited on page 205.)

[126] D. Bertaccini and F. Durastante. *Iterative Methods and Preconditioning for Large and Sparse Linear Systems with Applications*. CRC Press, 2018. (Cited on page 417.)

[127] R. H. Bisseling and J. G. G. van de Vorst. Parallel triangular system solving on a mesh network of transputers. *SIAM J. Sci. Statist. Comput.*, 12(4):787–799, 1991. (Cited on page 140.)

[128] Å. Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT Numerical Mathematics*, 7(1):1–21, 1967. (Cited on pages 47 and 48.)

[129] Å. Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra Appl.*, 197:297–316, 1994. (Cited on pages 46 and 47.)

[130] Å. Björck. *Numerical Methods for Least squares Problems*. SIAM, 1996. (Cited on page 47.)

[131] P. Bjørstad and O. B. Widlund. Iterative methods for the solution of elliptic problems on regions partitioned into substructures. *SIAM J. Numer. Anal.*, 23(6):1097–1120, 1986. (Cited on page 398.)

[132] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. W. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. SIAM, 1997. (Cited on page 140.)

[133] J.R. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In A. George, J. R. Gilbert, and J. W.-H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 1–29. Springer Verlag, 1993. (Cited on page 66.)

[134] P. Blanchard, N. J. Higham, and T. Mary. A class of fast and accurate summation algorithms. *SIAM J. Sci. Comput.*, 42(3):A1541–A1557, 2020. (Cited on page 32.)

[135] J. A. M. Bollen. Numerical stability of descent methods for solving linear equations. *Numer. Math.*, 43:361–377, 1984. (Cited on page 252.)

[136] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra Appl.*, 338:201–218, 2001. (Cited on pages 370, 416, and 420.)

[137] M. Bollhöfer. A robust and efficient ILU that incorporates the growth of the inverse triangular factors. *SIAM J. Sci. Comput.*, 25(1):86–103, 2003. (Cited on pages 370, 416, and 420.)

[138] M. Bollhöfer and V. Mehrmann. Algebraic multilevel methods and sparse approximate inverses. *SIAM J. Matrix Anal. Appl.*, 24(1):191–218, 2002. (Cited on page 415.)

[139] M. Bollhöfer and Y. Saad. ILUs and factorized approximate inverses are strongly related. Part I: Overview of results. Technical Report UMSI-2000-39, University of Minnesota, 2000. (Cited on page 380.)

[140] M. Bollhöfer and Y. Saad. A factored approximate inverse preconditioner with pivoting. *SIAM J. Matrix Anal. Appl.*, 23(3):692–705, 2002. (Cited on pages 160 and 380.)

[141] M. Bollhöfer and Y. Saad. Multilevel preconditioners constructed from inverse-based ILUs. *SIAM J. Sci. Comput.*, 27(5):1627–1650, 2006. (Cited on pages 415 and 421.)

[142] E. G. Boman and B. Hendrickson. A multilevel algorithm for reducing the envelope of sparse matrices. Technical Report SCCM-96-14, Computer Science Dept., Stanford University, 1996. (Cited on page 203.)

[143] J. H. Bramble. *Multigrid Methods*. Longman Scientific & Technical, Essex, UK, 1993. Pitman Research Notes in Mathematical Sciences Vol. 294. (Cited on pages 411 and 421.)

[144] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I. *Math. Comput.*, 47(175):103–134, 1986. (Cited on pages 404, 405, and 406.)

[145] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring II. *Math. Comput.*, 49(179):1–16, 1987. (Cited on page 404.)

[146] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring III. *Math. Comput.*, 51(184):415–430, 1988. (Cited on page 404.)

[147] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring IV. *Math. Comput.*, 53(187):1–24, 1989. (Cited on page 404.)

[148] J. H. Bramble, J. E. Pasciak, J. P. Wang, and J. Xu. Convergence estimates for multigrid algorithms without regularity assumptions. *Math. Comput.*, 57(195):23–45, 1991. (Cited on page 421.)

[149] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comput.*, 55(191):1–22, 1990. (Cited on page 416.)

[150] C. W. Brand. An incomplete factorization preconditioning using repeated Red/Black ordering. *Numer. Math.*, 61:433–454, 1992. (Cited on page 417.)

[151] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In H. Cabannes and R. Temam, editors, *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, pages 82–89. Springer, 1973. Lecture Notes in Physics 18. (Cited on page 421.)

[152] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comput.*, 31:333–390, 1977. (Cited on page 421.)

[153] A. Brandt. Guide to multigrid development. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods, Proceedings of the Koln-Porz Conference*, pages 220–312. Springer, 1982. Lecture Notes in Mathematics 960. (Cited on page 421.)

[154] A. Brandt. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.*, 19:23–56, 1986. (Cited on page 421.)

[155] A. Brandt. Rigorous quantitative analysis of multigrid, I: Constant coefficients two-level cycle with L_2-norm. *SIAM J. Numer. Anal.*, 31:1695–1730, 1994. (Cited on page 421.)

[156] S. C. Brenner. Preconditioning complicated finite elements by simple finite elements. *SIAM J. Sci. Comput.*, 17:1269–1274, 1996. (Cited on page 417.)

[157] C. Brezinski. *Projection Methods for Systems of Equations*. North-Holland, Amsterdam, 1997. (Cited on page 156.)

[158] C. Brezinski. Hybrid methods for solving systems of equations. In *NATO ASI Series C Mathematical and Physical Sciences - Advanced Study Institute*, volume 508, pages 271–290. NATO, 1998. (Cited on page 359.)

[159] C. Brezinski. Error estimates for the solution of linear systems. *SIAM J. Sci. Comput.*, 21(2):764–781, 1999. (Cited on pages 284 and 304.)

[160] C. Brezinski, G. Meurant, and M. Redivo-Zaglia. *A Journey Through the History of Numerical Linear Algebra*. SIAM, 2022. (Cited on pages 75, 83, 100, 129, 146, 183, 224, 260, and 304.)

[161] C. Brezinski and M. Redivo-Zaglia. Treatment of near-breakdown in the CGS algorithm. *Numer. Algorithms*, 7:33–73, 1994. (Cited on page 360.)

[162] C. Brezinski and M. Redivo-Zaglia. Look-ahead in BiCGstab and other methods for linear systems. *BIT*, 35:169–201, 1995. (Cited on pages 343 and 360.)

[163] C. Brezinski and M. Redivo-Zaglia. Transpose-free Lanczos-type algorithms for nonsymmetric linear systems. *Numer. Algorithms*, 17:67–103, 1998. (Cited on page 360.)

[164] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numer. Algorithms*, 1:261–284, 1991. (Cited on page 336.)

[165] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. Addendum to "Avoiding breakdown and near-breakdown in Lanczos type algorithms". *Numer. Algorithms*, 2:133–136, 1992. (Cited on page 336.)

[166] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. A breakdown-free Lanczos type algorithm for solving linear systems. *Numer. Math.*, 63:29–38, 1992. (Cited on page 336.)

[167] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. Breakdowns in the implementation of the Lanczos method for solving linear systems. *Computers Math. Applic.*, 33(1):31–44, 1997. (Cited on page 336.)

[168] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. New look-ahead Lanczos-type algorithms for linear systems. *Numer. Math.*, 83(1):53–85, 1999. (Cited on page 336.)

[169] C. Brezinski, M. Redivo-Zaglia, and H. Sadok. A review of formal orthogonality in Lanczos-based methods. *J. Comput. Appl. Math.*, 140(1):81–98, 2002. (Cited on page 336.)

[170] C. Brezinski and H. Sadok. Avoiding breakdown in the CGS algorithm. *Numer. Algorithms*, 1:199–206, 1991. (Cited on page 360.)

[171] C. Brezinski and D. Tournès. *André-Louis Cholesky, mathematician, topographer and army officer*. Birkhäuser, 2014. (Cited on page 183.)

[172] W. L. Briggs. *A Multigrid Tutorial*. SIAM, 1987. (Cited on page 421.)

[173] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000. (Cited on pages 411 and 421.)

[174] P. N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM J. Sci. Statist. Comput.*, 12(1):58–78, 1991. (Cited on pages 308 and 359.)

[175] P. N. Brown and A. C. Hindmarsh. Reduced storage matrix methods in stiff ODE systems. *Appl. Math. Comput.*, 31:40–91, 1989. (Cited on page 325.)

[176] R. Bru, J. Cerdán, J. Marín, and J. Mas. Preconditioning sparse nonsymmetric linear systems with the Sherman-Morrison formula. *SIAM J. Matrix Anal. Appl.*, 25(2):701–715, 2003. (Cited on page 417.)

[177] R. Bru, J. Marín, J. Mas, and M. Tůma. Balanced incomplete factorization. *SIAM J. Sci. Comput.*, 30(5):2302–2318, 2008. (Cited on pages 417 and 420.)

[178] H. M. Bücker and M. Sauren. A parallel version of the quasi-minimal residual method based on coupled two-term recurrences. In *International Workshop on Applied Parallel Computing*, Berlin, Heidelberg, 1996. Springer. (Cited on page 347.)

[179] H. M. Bücker and M. Sauren. A variant of the biconjugate gradient method suitable for massively parallel computing. In *International Symposium on Solving Irregularly Structured Problems in Parallel*, Berlin, Heidelberg, 1997. Springer. (Cited on page 338.)

[180] H. M. Bücker and M. Sauren. Reducing global synchronization in the biconjugate gradient method. In L. T. Yang, editor, *Parallel Numerical Computation with Applications*, pages 63–76, Boston, MA, 1999. Springer. (Cited on page 338.)

[181] N. I. Buleev. A numerical method for the solution of two-dimensional and three-dimensional equations of diffusion. *Mat. Sb.*, 51:227–238, 1960. (in Russian). (Cited on pages 371 and 418.)

[182] J. R. Bunch. Equilibration of symmetric matrices in the max-norm. *J. ACM*, 18:566–572, 1971. (Cited on page 128.)

[183] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comput.*, 31(137):63–179, 1977. (Cited on page 107.)

[184] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 8(4):639–655, 1971. (Cited on page 107.)

[185] J. V. Burke and A. Greenbaum. Some equivalent characterizations of the polynomial numerical hull of degree k. Technical Report NA-04-29, The Mathematical Institute, University of Oxford, 2004. (Cited on page 317.)

[186] J. V. Burke and A. Greenbaum. Characterizations of the polynomial numerical hull of degree k. *Linear Algebra Appl.*, 419:37–47, 2006. (Cited on page 317.)

[187] A. Buttari, J. J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Trans. Math. Soft. (TOMS)*, 34(4):1–22, 2008. (Cited on page 223.)

[188] A. Buttari, J. Langou, J. Langou, P. Luszczek, and J. Kurzak. Mixed precision iterative refinement techniques for the solution of dense linear systems. *Int. J. High Perform. Comput. Appl.*, 21(4):457–466, 2007. (Cited on page 129.)

[189] B. Bylina and J. Bylina. Solving linear systems using the method of WZ factorization with BLAS1 and BLAS2 libraries. *Studia Informatica*, 25(1):57–67, 2004. (Cited on page 168.)

[190] B. Bylina and J. Bylina. Markowitz scheme for the sparse WZ factorization. *Annales UMCS Informatica AI*, 6:85–94, 2007. (Cited on page 168.)

[191] B. Bylina and J. Bylina. Mixed precision iterative refinement techniques for the WZ factorization. In *Federated Conference on Computer Science and Information Systems*, pages 425–431. IEEE, 2013. (Cited on page 168.)

[192] B. Bylina and J. Bylina. The WZ factorization in MATLAB. In *Federated Conference on Computer Science and Information Systems*, pages 561–568. IEEE, 2014. (Cited on page 168.)

[193] B. Bylina and J. Bylina. Strategies of parallelizing nested loops on the multicore architectures on the example of the WZ factorization for the dense matrices. In *Federated Conference on Computer Science and Information Systems*, pages 629–639. IEEE, 2015. (Cited on page 168.)

[194] X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.*, 21(2):792–797, 1999. (Cited on page 394.)

[195] X.-C. Cai and O. B. Widlund. Multiplicative Schwarz algorithms for some nonsymmetric and indefinite problems. *SIAM J. Numer. Anal.*, 30(4):936–952, 1993. (Cited on page 416.)

[196] D. Calvetti, G. H. Golub, and L. Reichel. A computable error bound for matrix functionals. *J. Comput. Appl. Math.*, 103:301–306, 1999. (Cited on page 304.)

[197] D. Calvetti, S. Morigi, L. Reichel, and F. Sgallari. Computable error bounds and estimates for the conjugate gradient method. *Numer. Algorithms*, 25:79–88, 2000. (Cited on pages 284 and 304.)

[198] D. Calvetti, S. Morigi, L. Reichel, and F. Sgallari. An iterative method with error estimators. *J. Comput. Appl. Math.*, 127:93–119, 2001. (Cited on page 304.)

[199] D. Calvetti, L. Reichel, and F. Sgallari. Application of anti-Gauss quadrature rules in linear algebra. In W. Gautschi, G. H. Golub, and G. Opfer, editors, *Applications and Computation of Orthogonal Polynomials*, pages 41–56, Basel, 1999. Birkhäuser. (Cited on pages 284 and 304.)

[200] B. Carpentieri, I. S. Duff, and L. Giraud. A class of spectral two-level preconditioners. *SIAM J. Sci. Comput.*, 25(2):749–765, 2003. (Cited on page 417.)

[201] E. C. Carson. *Communication-avoiding Krylov subspace methods in theory and practice*. PhD thesis, University of California at Berkeley, 2015. Technical Report UCB/EECS-2015-179. (Cited on pages 338, 344, and 360.)

[202] E. C. Carson. The adaptive s-step conjugate gradient method. *SIAM J. Matrix Anal. Appl.*, 39(3):1318–1338, 2018. (Cited on page 286.)

[203] E. C. Carson and J. W. Demmel. Error analysis of the s-step biconjugate gradient method in finite precision. Technical Report UCB/EECS-2014-18, EECS Department, University of California, Berkeley, CA, 2014. (Cited on page 338.)

[204] E. C. Carson and N. J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Sci. Comput.*, 39(6):A2834–A2856, 2017. (Cited on page 223.)

[205] E. C. Carson and N. J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Sci. Comput.*, 40(2):A817–A847, 2018. (Cited on pages 129, 146, and 223.)

[206] E. C. Carson, N. Knight, and J. W. Demmel. Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods. *SIAM J. Sci. Comput.*, 35(5):S42–S61, 2013. (Cited on page 338.)

[207] E. C. Carson, M. Rozložník, Z. Strakoš, P. Tichý, and M. Tůma. The numerical stability analysis of pipelined conjugate gradient methods: Historical context and methodology. *SIAM J. Sci. Comput.*, 40(5):A3549–A3580, 2018. (Cited on pages 286 and 304.)

[208] L. M. Carvalho. *Preconditioned Schur complement methods in distributed memory environments*. PhD thesis, INPT Toulouse, 1997. CERFACS report TH/PA/97/41. (Cited on pages 407 and 408.)

[209] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numer. Linear Algebra Appl.*, 8(4):207–227, 2001. (Cited on page 408.)

[210] S. Catalán, P. Ezzatti, E. S. Quintana-Ortí, and A. Remón. The impact of panel factorization on the Gauss-Huard algorithm for the solution of linear systems on modern architectures. In *Algorithms and Architectures for Parallel Processing: 16th International Conference*, pages 405–416, Cham, 2016. Springer. (Cited on page 153.)

[211] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. *La Ricerca Scientifica, Progr. Tecn. Econom. Naz., serie II, anno VIII*, 1(11-12):512–522, 1937. (Cited on pages 382 and 420.)

[212] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. Technical Report 12, Istituto per le applicazioni del calcolo, Consiglio Nazionale delle Richerche, Roma, 1937. (Cited on pages 382 and 420.)

[213] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. *Rass. Poste, dei Teleg. e dei Telef., anno IX*, 4:285–296, 1937. (Cited on pages 382 and 420.)

[214] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. *Rendic. Reale Accademia Nazionale dei Lincei, Classe Scienze Fis., Mat., Natur., ser. 6a*, 25:422–428, 1937. (Cited on pages 382 and 420.)

[215] F. Chaitin-Chatelin and V. Frayssé. *Lectures on Finite Precision Computations*. SIAM, 1996. (Cited on page 3.)

[216] R. M. Chamberlain. An algorithm for LU factorization with partial pivoting on the hypercube. Technical Report CCS 86/11, Chr. Michelsen Institute, 1986. (Cited on pages 139 and 141.)

[217] R. M. Chamberlain. An aalternative view of LU factorization with partial pivoting on a hypercube multiprocessor. In M. T. Heath, editor, *Hypercube Multiprocessors 1987*, pages 569–575. SIAM, 1987. (Cited on pages 139 and 141.)

[218] R. H. Chan, C. Greif, and D. P. O' Leary. *Milestones in Matrix Computation: The Selected Works of Gene H. Golub with Commentaries*. Oxford University Press, Oxford, 2007. (Cited on page 451.)

[219] T. F. Chan and K. Chen. Two-stage preconditioners using wavelet band splitting and sparse approximation. Dpt. of Mathematics, University of California, Los Angeles, 2000. (Cited on page 416.)

[220] T. F. Chan and K. Chen. On two variants of an algebraic wavelet preconditioner. *SIAM J. Sci. Comput.*, 24(1):260–283, 2002. (Cited on page 416.)

[221] T. F. Chan and D. E. Foulser. Effectively well conditioned linear systems. *SIAM J. Sci. Statist. Comput.*, 9:963–969, 1988. (Cited on page 126.)

[222] T. F. Chan, C. C. J. Kuo, and C. Tong. Multilevel filtering preconditioners. *SIAM J. Matrix Anal. Appl.*, 11(3):403–429, 1990. (Cited on pages 415 and 417.)

[223] T. F. Chan and T. Mathew. An application of the probing technique to the vertex space method in domain decomposition. In R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and O. B. Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 101–111. SIAM, 1991. (Cited on pages 400 and 406.)

[224] T. F. Chan and T. Mathew. The interface probing in domain decomposition. *SIAM J. Matrix Anal. Appl.*, 13:212–238, 1992. (Cited on page 400.)

[225] T. F. Chan, W. P. Tang, and W. L. Wan. Wavelet sparse approximate inverse preconditioners. *BIT Numerical Mathematics*, 37(3):644–660, 1997. (Cited on page 416.)

[226] T. F. Chan and J. Zou. A convergence theory of multilevel additive Schwarz methods on unstructured meshes. *Numer. Algorithms*, 13(2):365–398, 1996. (Cited on page 416.)

[227] R. Chandra. *Conjugate gradient methods for partial differential equations*. PhD thesis, Yale University, 1978. (Cited on pages 286, 287, 304, and 372.)

[228] S. Chandrasekaran and I. C. F. Ipsen. On rank-revealing factorisations. *SIAM J. Matrix Anal. Appl.*, 15(2):592–622, 1994. (Cited on page 127.)

[229] S. Chandrasekaran and I. C. F. Ipsen. On the sensitivity of solution components in linear systems of equations. *SIAM J. Matrix Anal. Appl.*, 16(1):93–112, 1995. (Cited on page 127.)

[230] X.-W. Chang. Some features of Gaussian elimination with rook pivoting. *BIT Numerical Mathematics*, 42(1):66–83, 2002. (Cited on page 96.)

[231] X.-W. Chang and C. C. Paige. On the sensitivity of the LU factorization. *BIT Numerical Mathematics*, 38(3):486–501, 1998. (Cited on page 125.)

[232] X.-W. Chang, C. C. Paige, and G. W. Stewart. New perturbation analyses for the Cholesky factorization. *IMA J. Numer. Anal.*, 16(4):457–484, 1996. (Cited on page 125.)

[233] X.-W. Chang and D. Stehlé. Rigorous perturbation bounds of some matrix factorizations. *SIAM J. Matrix Anal. Appl.*, 31(5):2841–2859, 2010. (Cited on page 125.)

[234] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. *Numer. Linear Algebra Appl.*, 4:43–66, 1997. (Cited on page 359.)

[235] A. Charara, D. E. Keyes, and H. Ltaief. A framework for dense triangular matrix kernels on various manycore architectures. *Concurr. Comp.-Pract. E.*, 29(15):e4187, 2017. (Cited on page 141.)

[236] A. Charara, H. Ltaief, and D. E. Keyes. Redesigning triangular dense matrix computations on GPUs. In *22nd International Conference on Parallel and Distributed Computing*, pages 477–489. Springer, 2016. (Cited on page 141.)

[237] P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissection emboîtée. *Numer. Math.*, 55:463–476, 1989. (Cited on page 208.)

[238] P. Charrier and J. Roman. Analysis of refined partitions for a parallel implementation of nested dissection. Technical Report LABRI 91-38, Université de Bordeaux, Laboratoire Bordelais de Recherche en Informatique, 1991. (Cited on page 208.)

[239] Y. Chauvet and G. Meurant. Multitasking on the CRAY X-MP. *The Journal of Systems and Software*, 2:17–20, 1986. Proceedings of International Workshop on Modeling and Performance Evaluation of Parallel Systems, IMAG Grenoble (1984). (Cited on page 304.)

[240] C. Chen and P. G. Martinsson. Solving linear systems on a GPU with hierarchically off-diagonal low-rank approximations. ArXiv preprint, arXiv:2208.06290, 2022. (Cited on page 223.)

[241] D. Chen and S. Toledo. Vaidya's preconditioners: Implementation and experimental study. *Electron. Trans. Numer. Anal.*, 16:30–49, 2003. (Cited on page 417.)

[242] K. Chen. *Matrix Preconditioning Techniques and Applications*. Cambridge University Press, UK, 2005. (Cited on pages 146, 147, 153, 417, and 420.)

[243] K. Chen and D. Evans. An efficient variant of Gauss-Jordan type algorithms for direct and parallel solution of dense linear systems. *Int. J. Comput. Math.*, 76(3):387–410, 2001. (Cited on page 157.)

[244] K. Chen and C.-H. Lai. Parallel algorithms of the Purcell method for direct solution of linear systems. *Parallel Comput.*, 28(9):1275–1291, 2002. (Cited on page 157.)

[245] S. C. Chen and D. J. Kuck. Time and parallel processor bounds for linear recurrence systems. *IEEE Trans. Comput.*, 100(7):701–717, 1975. (Cited on page 138.)

[246] S. C. Chen, D. J. Kuck, and A. H. Sameh. Practical parallel band triangular system solvers. *ACM Trans. Math. Soft. (TOMS)*, 1(3):270–277, 1978. (Cited on page 138.)

[247] T. Chen and E. C. Carson. Predict-and-recompute conjugate gradient variants. *SIAM J. Sci. Comput.*, 42(5):A3084–A3108, 2020. (Cited on pages 285, 298, and 304.)

[248] X. Chen, Y. Wang, and H. Yang. NICSLU: An adaptive sparse matrix solver for parallel circuit simulation. *IEEE Trans. Computer-Aided Design Integ. Circ. Sys.*, 32(2):261–274, 2013. (Cited on page 218.)

[249] Y. T. Chen and R. P. Tewarson. On the optimal choice of pivots for the Gaussian elimination. *Computing*, 9(3):245–250, 1972. (Cited on page 217.)

[250] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Comput.*, 34(6-8):318–331, 2008. (Cited on page 220.)

[251] F. Chiò. *Mémoire sur les Fonctions Connues sous les Noms de Résultantes ou de Déterminans*. A. Pons et C., Torino, 1853. (Cited on page 84.)

[252] J. Choi, J. J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. Design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. *Sci. Programming*, 5(3):173–184, 1996. (Cited on page 142.)

[253] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *The Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE, 1992. (Cited on page 140.)

[254] E. Chow. Parallel implementation and performance characteristics of least squares sparse approximate inverse preconditioners. Technical Report UCRL-JC-138883, Lawrence Livermore National Laboratory, 2000. (Cited on page 379.)

[255] E. Chow and A. Patel. Fine-grained parallel incomplete LU factorization. *SIAM J. Sci. Comput.*, 37(2):C169–C193, 2015. (Cited on pages 377 and 420.)

[256] E. Chow and Y. Saad. Approximate inverse preconditioners for general sparse matrices. Technical Report UMSI 94-101, Supercomputer Institute, University of Minnesota, 1994. (Cited on page 380.)

[257] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, 1998. (Cited on page 417.)

[258] A. T. Chronopoulos. *A class of parallel iterative methods implemented on multiprocessors*. PhD thesis, University of Illinois at Urbana-Champaign, 1987. (Cited on page 304.)

[259] A. T. Chronopoulos and C. W. Gear. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Comput.*, 11(1):37–53, 1989. (Cited on pages 285, 298, and 304.)

[260] A. T. Chronopoulos and C. W. Gear. S-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math.*, 25(2):153–168, 1989. (Cited on pages 285, 298, and 304.)

[261] E. Chu and A. George. Gaussian elimination with partial pivoting and load balancing on a multiprocessor. *Parallel Comput.*, 5:65–74, 1987. (Cited on page 141.)

[262] M. T. Chu, R. E. Funderlic, and G. H. Golub. A rank one reduction formula and its applications to matrix factorizations. *SIAM Rev.*, 37:512–530, 1995. (Cited on page 162.)

[263] G. Ciaramella and M. J. Gander. *Iterative Methods and Preconditioners for Systems of Linear Equations*. SIAM, 2021. (Cited on pages 417 and 421.)

[264] P. G. Ciarlet. *The finite element method for elliptic problems*. SIAM, 2002. (Cited on pages 7 and 78.)

[265] P. Ciarlet Jr. and F. Lamour. An efficient low cost greedy graph partitioning heuristic. Technical Report CAM 94-1, Dept. of Mathematics, UCLA, 1994. (Cited on page 209.)

[266] P. Ciarlet Jr. and F. Lamour. Spectral partitioning methods and greedy partitioning methods: a comparison on finite element graphs. Technical Report CAM 94-9, Dept. of Mathematics, UCLA, 1994. (Cited on page 209.)

[267] P. Ciarlet Jr. and F. Lamour. On the validity of a front oriented approach to partitioning large sparse graphs with a connectivity constraint. *Numer. Algorithms*, 12(1):193–214, 1996. (Cited on page 209.)

[268] B. I. Clasen. Sur une nouvelle méthode de résolution des équations linéaires et sur l'application de cette méthode au calcul des déterminants. *Ann. Soc. Sci. Bruxelles*, 12:251–281, 1888. also in *Mathesis*, t. IX, 1889, pp. 320-350. (Cited on pages 146 and 183.)

[269] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones. Coarse-grid selection for parallel algebraic multigrid. In A. Ferreira, J. Rolim, H. Simon, and S.-H. Teng, editors, *Solving Irregularly Structured Problems in Parallel: 5th International Symposium*, pages 104–115. Springer, 1998. (Cited on page 413.)

[270] T. P. Collignon. *Efficient iterative solution of large linear systems on heterogeneous computing systems*. PhD thesis, Delft University of Technology, The Netherlands, 2013. (Cited on page 351.)

[271] T. P. Collignon and M. B. van Gijzen. Minimizing synchronization in IDR(s). *Numer. Linear Algebra Appl.*, 18(5):805–825, 2011. (Cited on pages 351 and 361.)

[272] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations*, pages 309–332. Academic Press, 1976. (Cited on page 303.)

[273] P. Concus, G.H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:220–252, 1985. (Cited on pages 59, 112, 378, and 420.)

[274] P. Concus and G. Meurant. On computing INV block preconditionings for the conjugate gradient method. *BIT*, 26:493–504, 1986. (Cited on page 420.)

[275] S. Cools. Analyzing and improving maximal attainable accuracy in the communication hiding pipelined BiCGStab method. *Parallel Comput.*, 86:16–35, 2019. PMAA'18 Special Issue. (Cited on pages 344 and 360.)

[276] S. Cools, J. Cornelis, P. Ghysels, and W. Vanroose. Improving strong scaling of the conjugate gradient method for solving large linear systems using global reduction pipelining. ArXiv preprint arXiv:1905.06850, 2019. EuroMPI'19: Proceedings of the 26th European MPI Users' Group Meeting. (Cited on page 304.)

[277] S. Cools, J. Cornelis, and W. Vanroose. Numerically stable recurrence relations for the communication hiding pipelined conjugate gradient method. *IEEE Trans. Parallel Distrib. Syst.*, 30(11):2507–2522, 2019. (Cited on page 286.)

[278] S. Cools, J. Cornelis, and W. Vanroose. Numerically stable variants of the communication-hiding pipelined conjugate gradient method. *IEEE Trans. Parallel Distrib. Syst.*, 30(11):2507–2522, 2019. (Cited on pages 286 and 360.)

[279] S. Cools and W. Vanroose. The communication-hiding pipelined BiCGStab method for the efficient parallel solution of large unsymmetric linear systems. *Parallel Comput.*, 65:1–20, 2017. (Cited on pages 344 and 360.)

[280] S. Cools and W. Vanroose. Numerically stable variants of the communication-hiding pipelined conjugate gradients algorithm for the parallel solution of large scale symmetric linear systems. ArXiv preprint, arXiv:1706.05988, 2017. (Cited on page 298.)

[281] S. Cools, W. Vanroose, E. F. Yetkin, E. Agullo, and L. Giraud. On rounding error resilience, maximal attainable accuracy and parallel performance of the pipelined conjugate gradients method for large-scale linear systems in PETSc. In *Proceedings of the EASC'16 Exascale Applications and Software Conference 2016*, pages 1–10. ACM, 2016. (Cited on page 286.)

[282] S. Cools, W. Vanroose, E. F. Yetkin, E. Agullo, and L. Giraud. Analyzing the effect of local rounding error propagation on the maximal attainable accuracy of the pipelined conjugate gradient method. *SIAM J. Matrix Anal. Appl.*, 39(1):426–450, 2018. (Cited on pages 286 and 304.)

[283] J. Cornelis, S. Cools, and W. Vanroose. The communication-hiding conjugate gradient method with deep pipelines. ArXiv preprint arXiv:1801.04728v2, 2018. (Cited on page 304.)

[284] V. Cortés and J. M. Peña. Growth factor and expected growth factor of some pivoting strategies. *J. Comput. Appl. Math.*, 202(2):292–303, 2007. (Cited on page 96.)

[285] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *Int. J. Comput. Math.*, 44(1-4):91–110, 1992. (Cited on page 417.)

[286] M. Cosnard, Y. Robert, and D. Trystram. Résolution parallèle de systèmes linéaires denses par diagonalisation. *EDF, Bulletin DER, serie C*, 2:67–88, 1986. (Cited on page 153.)

[287] M. Cosnard, Y. Robert, and D. Trystram. Parallel solution of dense linear systems using diagonalization methods. *Int. J. Comput. Math.*, 22(3-4):249–270, 1987. (Cited on page 153.)

[288] E. J. Craig. *Iteration procedures for simultaneous equations*. PhD thesis, Massachusetts Institute of Technology, 1954. (Cited on page 331.)

[289] E. J. Craig. The N-step iteration procedures. *J. Math. Phys.*, 34(1-4):64–73, 1955. (Cited on page 331.)

[290] G. Cramer. *Introduction à l'Analyse des Lignes Courbes Algébriques*. Frères Cramer et CL. Philibert, Genéve, 1750. (Cited on pages 84 and 182.)

[291] P. D. Crout. A short method for evaluating determinants and solving systems of linear equations with real or complex coefficients. *Trans. Amer. Inst. Elect. Engrs*, 60(12):1235–1241, 1941. (Cited on page 183.)

[292] C. W. Cryer. Pivot growth in Gaussian elimination. *Numer. Math.*, 12:335–345, 1968. (Cited on page 123.)

[293] L. Csanky. Fast parallel matrix inversion algorithms. *SIAM J. Comput.*, 5(4):618–623, 1976. (Cited on page 138.)

[294] R. Cuevas, C. E. Schaerer, and A. Bhaya. A proportional-derivative control strategy for varying the restart parameter in GMRES(m). In *Anais do XXXIII CNMAC*, volume 3, pages 1000–1001, 2010. (Cited on page 324.)

[295] J. Cullum. Peaks, plateaus, numerical instabilities in a Galerkin minimal residual pair of methods for solving $Ax = b$. *Appl. Numer. Math.*, 19(3):255–278, 1995. (Cited on page 359.)

[296] J. Cullum. Iterative methods for solving $Ax = b$, GMRES/FOM versus QMR/BiCG. *Adv. Comput. Math.*, 6:1–24, 1996. (Cited on page 359.)

[297] J. Cullum and A. Greenbaum. Relations between Galerkin and norm-minimizing iterative methods for solving linear systems. *SIAM J. Matrix Anal. Appl.*, 17(2):223–247, 1996. (Cited on pages 308 and 359.)

[298] A. R. Curtis and J. K. Reid. The solution of large sparse unsymmetric systems of linear equations. *IMA J. Appl. Math.*, 8(3):344–353, 1971. (Cited on pages 217 and 224.)

[299] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *IMA J. Appl. Math.*, 10(l):118–124, 1972. (Cited on page 128.)

[300] E. H. Cuthill. Several strategies for reducing the bandwidth of matrices. In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and their Applications*, pages 157–166, New York, 1972. Plenum Press. (Cited on page 199.)

[301] E. H. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *24th ACM National Conference, ACM Publ P-69*, pages 157–172. ACM, 1969. (Cited on pages 194 and 224.)

[302] G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice-Hall, 1974. (Cited on page 74.)

[303] G. Dahlquist, S. C. Eisenstat, and G. H. Golub. Bounds for the error of linear systems of equations using the theory of moments. *J. Math. Anal. Appl.*, 37:151–166, 1972. (Cited on page 304.)

[304] G. Dahlquist, G. H. Golub, and S. G. Nash. Bounds for the error in linear systems. In R. Hettich, editor, *Proceedings of the Workshop on Semi-infinite Programming*, pages 154–172, Berlin, 1978. Springer. (Cited on pages 279 and 304.)

[305] W. Dahmen and A. Kunoth. Multilevel preconditioning. *Numer. Math.*, 63:315–344, 1992. (Cited on page 415.)

[306] M. Dailey, F. M. Dopico, and Q. Ye. Relative perturbation theory for diagonally dominant matrices. *SIAM J. Matrix Anal. Appl.*, 35(4):1303–1328, 2014. (Cited on page 126.)

[307] J.-P. David. Low latency and division free Gauss-Jordan solver in floating point arithmetic. *J. Parallel Dist. Comput.*, 106:185–193, 2017. (Cited on page 150.)

[308] T. A. Davis. Performance of an unsymmetric pattern multifrontal method for sparse LU factorization. Technical Report TR-92-014, University of Florida, 1992. (Cited on page 219.)

[309] T. A. Davis. Users' guide for the unsymmetric pattern multifrontal package. Technical Report TR-93-020, University of Florida, 1993. (Cited on page 219.)

[310] T. A. Davis. Algorithm 832: UMFPACK v4.3 - an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Soft. (TOMS)*, 30(2):196–199, 2004. (Cited on pages 217, 219, and 221.)

[311] T. A. Davis. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Trans. Math. Soft. (TOMS)*, 31(4):587–591, 2005. (Cited on page 193.)

[312] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006. (Cited on pages 185, 193, 218, and 219.)

[313] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Appl.*, 18(1):140–158, 1997. (Cited on pages 217, 219, 221, and 224.)

[314] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Soft. (TOMS)*, 30(3):353–376, 2004. (Cited on page 206.)

[315] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016. (Cited on pages 185 and 224.)

[316] M. J. Daydé and I. S. Duff. Use of Level 3 BLAS in LU factorization on the Cray 2, the ETA-10P and the IBM 3090/VF. *Intl. J. Supercomp. Appl.*, 3(2):40–70, 1989. (Cited on page 142.)

[317] E. F. D'Azevedo and J. J. Dongarra. Packed storage extension for ScaLAPACK. Technical Report ORNL/TM-13545, Oak Ridge National Lab., 1997. (Cited on page 140.)

[318] E. F. D'Azevedo, V. Eijkhout, and C. H. Romine. Conjugate gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Technical Report Lapack Working Notes 56, University of Tennessee, 1999. (Cited on page 285.)

[319] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM J. Matrix Anal. Appl.*, 13(3):944–961, 1992. (Cited on page 377.)

[320] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Towards a cost-effective ILU preconditioner with high level fill. *BIT Numerical Mathematics*, 32(3):442–463, 1992. (Cited on page 417.)

[321] E. F. D'Azevedo and J. C. Hill. Parallel LU factorization on GPU cluster. *Procedia Comput. Sci.*, 9:67–75, 2012. (Cited on page 144.)

[322] C. de Boor and J. R. Rice. Extremal polynomials with application to Richardson iteration for indefinite linear systems. *SIAM J. Sci. Statist. Comput.*, 3(1):47–57, 1982. (Cited on pages 248 and 388.)

[323] F. de Dinechin, L. Forget, J. M. Muller, and Y. Uguen. Posits: The good, the bad and the ugly. In *Proceedings of the Conference for Next Generation Arithmetic 2019*, pages 1–10. ACM, 2019. (Cited on page 18.)

[324] E. de Sturler. Nested Krylov methods based on GCR. *J. Comput. Appl. Math.*, 67:15–41, 1996. (Cited on page 329.)

[325] E. de Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM J. Numer. Anal.*, 36(3):864–889, 1999. (Cited on page 329.)

[326] E. de Sturler and D. R. Fokkema. Nested Krylov methods and preserving the orthogonality. In D. Melson, T. Manteuffel, and S. F. Mc Cormick, editors, *Sixth Copper Mountain Conference on Multigrid Methods*, pages 111–126. NASA conference publication 3324, 1993. (Cited on page 329.)

[327] E. de Sturler and J. Liesen. Block-diagonal and constraint preconditioners for nonsymmetric indefinite linear systems. Part I: Theory. *SIAM J. Sci. Comput.*, 26(5):1598–1619, 2005. (Cited on page 417.)

[328] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971. (Cited on pages 29 and 35.)

[329] T. J. Dekker and W. Hoffmann. Rehabilitation of the Gauss-Jordan algorithm. *Numer. Math.*, 54(5):591–599, 1989. (Cited on page 149.)

[330] T. J. Dekker, W. Hoffmann, and K. Potma. Parallel algorithms for solving large linear systems. *J. Comput. Appl. Math.*, 50:221–232, 1994. (Cited on pages 150 and 153.)

[331] T. J. Dekker, W. Hoffmann, and K. Potma. Stability of the Gauss-Huard algorithm with partial pivoting. *Computing*, 58(3):225–244, 1997. (Cited on page 153.)

[332] J. W. Demmel. On floating point errors in Cholesky. Technical Report Lapack Working Note # 14, University of Tennessee, 1989. Also CS-89-87, Department of Computer Science, University of Tennessee. (Cited on page 124.)

[333] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999. (Cited on pages 211, 219, and 224.)

[334] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comput.*, 34(1):A206–A239, 2012. (Cited on page 143.)

[335] J. W. Demmel, L. Grigori, and H. Xiang. CALU: A communication optimal LU factorization algorithm. Technical Report EECS-2010-29, University of California, Berkeley, 2010. (Cited on page 143.)

[336] J. W. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy. Error bounds from extra-precise iterative refinement. *ACM Trans. Math. Soft. (TOMS)*, 32(2):325–351, 2006. (Cited on page 129.)

[337] J. W. Demmel, N. J. Higham, and R. S. Schreiber. Stability of block LU factorization. *Numer. Linear Algebra Appl.*, 2(2):173–190, 1995. (Cited on pages 110 and 167.)

[338] V. R. Deshpande, M. J. Grote, P. Messmer, and W. B. Sawyer. Parallel sparse approximate inverse preconditioner. Technical Report TR-96-14, Swiss Center for Scientific Computing, 1996. (Cited on page 379.)

[339] F. Desprez, B. Tourancheau, and J. J. Dongarra. Performance complexity of LU factorization with efficient pipelining and overlap on a multiprocessor. Technical Report Lapack Working Note 67, University of Tennessee, 1994. (Cited on page 141.)

[340] C. L. Dodgson. Condensation of determinants, being a new and brief method for computing their arithmetical values. *Proc. R. Soc. Lond.*, 15:150–155, 1867. (Cited on page 84.)

[341] C. R. Dohrmann. A preconditioner for substructuring based on constrained energy minimization. *SIAM J. Sci. Comput.*, 25(1):246–258, 2003. (Cited on page 407.)

[342] C. R. Dohrmann, K. H. Pierson, and O. B. Widlund. On inexact solvers for the coarse problem of BDDC. In R. Haynes, S. MacLachlan, X.-C. Cai, L. Halpern, H. H. Kim, A. Klawonn, and O. B. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXV*, pages 117–124. Springer, 2020. (Cited on page 407.)

[343] S. Doi. On parallelism and convergence of incomplete LU factorizations. *Appl. Numer. Math.*, 7(5):417–436, 1991. (Cited on page 420.)

[344] S. Doi and A. Hoshi. Large-numbered multicolor MILU preconditioning on SX-3/14. *Intl. J. Comput. Math.*, 44(1-4):143–152, 1992. (Cited on pages 377 and 417.)

[345] S. Doi and A. Lichnewsky. Some parallel and vector implementations of preconditioned iterative methods on Cray-2. *Intl. J. High Speed Computing*, 2(02):143–179, 1990. (Cited on pages 376 and 420.)

[346] S. Doi and A. Lichnewsky. A graph-theory approach for analyzing the effects of ordering on ILU preconditionning. Technical Report RR-1452, INRIA, Rocquencourt, France, 1991. (Cited on pages 376 and 420.)

[347] V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*. SIAM, 2015. (Cited on pages 394, 408, and 421.)

[348] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *Linpack User's Guide*. SIAM, Philadelphia, 1979. (Cited on pages 104 and 130.)

[349] J. J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek. Exploiting fine-grain parallelism in recursive LU factorization. In *Applications, Tools and Techniques on the Road to Exascale Computing*, pages 429–436. IOS Press, 2012. (Cited on page 144.)

[350] J. J. Dongarra, M. Faverge, H. Ltaief, and P. Luszczek. Achieving numerical accuracy and high performance using recursive tile LU factorization with partial pivoting. *Concurr. Comp.-Pract. E.*, 16(7):1408–1431, 2014. (Cited on page 145.)

[351] J. J. Dongarra and A. Geist. Report on the Oak Ridge National Laboratory's Frontier system. Technical Report ICL-UT-22-05, University of Tennessee, 2022. (Cited on page 146.)

[352] J. J. Dongarra, F. G. Gustavson, and A. Karp. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Rev.*, 26(1):91–112, 1984. (Cited on pages 101 and 105.)

[353] J. J. Dongarra and D. W. Walker. The design of linear algebra libraries for high performance computers. Technical Report ORNL/TM-12404, Oak Ridge National Lab., 1993. (Cited on page 142.)

[354] M. H. Doolittle. Explanations and illustrations of method employed in the office in solution of normal equations and the adjustment of a triangulation. In *Report of the Superintendant of the U.S. Coast and Geodetic Survey showing the progress of the work during fiscal year ending June 1878*, pages 115–120. Government Publishing Office, Washington, 1881. (Cited on pages 129 and 183.)

[355] F. M. Dopico and J. M. Molera. Perturbation theory for factorizations of LU type through series expansions. *SIAM J. Matrix Anal. Appl.*, 27(2):561–581, 2005. (Cited on page 125.)

[356] Z. Dostál. Projector preconditioning and domain decomposition methods. *Appl. Math. Comput.*, 37(2):75–81, 1990. (Cited on page 417.)

[357] C. C. Douglas. Multi-grid algorithms with applications to elliptic boundary-value problems. *SIAM J. Numer. Anal.*, 21(2):236–254, 1984. (Cited on page 421.)

[358] C. C. Douglas. A review of numerous parallel multigrid methods. In G. Astfalk, editor, *Applications on Advanced Architecture Computers*, pages 187–202. SIAM, 1996. (Cited on page 421.)

[359] C. C. Douglas and J. Douglas, Jr. A unified convergence theory for abstract multigrid or multilevel algorithms, serial and parallel. *SIAM J. Numer. Anal.*, 30:136–158, 1993. (Cited on page 421.)

[360] Z. Drmač, M. Omladič, and K. Veselič. On the perturbation of the Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15(4):1319–1332, 1994. (Cited on page 125.)

[361] M. Dryja. A capacitance matrix method for Dirichlet problem on polygon region. *Numer. Math.*, 39(1):51–64, 1982. (Cited on page 397.)

[362] M. Dryja and W. Proskurowski. Capacitance matrix method using strips with alternating Neumann and Dirichlet boundary conditions. *Appl. Numer. Math.*, 1:285–298, 1985. (Cited on page 402.)

[363] M. Dryja, M. V. Sarkis, and O. B. Widlund. Multilevel Schwarz methods for elliptic problems with discontinuous coefficients in three dimensions. *Numer. Math.*, 72(3):313–348, 1996. (Cited on page 416.)

[364] M. Dryja and O. Widlund. Some domain decomposition algorithms for elliptic problems. In L. Hayes and D. Kincaid, editors, *Iterative Methods for Large Linear Systems*, pages 273–291, San Diego, 1989. Academic Press. (Cited on page 393.)

[365] M. Dryja and O. B. Widlund. Additive Schwarz methods for elliptic finite element problems in three dimensions. In D. E. Keyes, T. F. Chan, G. Meurant, J. S. Scroggs, and R. G. Voigt, editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 3–18. SIAM, 1992. (Cited on page 393.)

[366] K. Du, J. Duintjer Tebbens, and G. Meurant. Any admissible harmonic Ritz value set is possible for GMRES. *Electron. Trans. Numer. Anal.*, 47:36–56, 2017. (Cited on page 359.)

[367] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. Approximating the inverse of a matrix for use in iterative algorithms on vector processors. *Computing*, 22(3):257–268, 1979. (Cited on pages 382 and 420.)

[368] I. S. Duff. Data structures, algorithms and software for sparse matrices. In D J. Evans, editor, *Sparsity and its Applications*, pages 1–29. Cambridge University Press, Cambridge, U.K., 1985. (Cited on page 217.)

[369] I. S. Duff. Parallel implementation of multifrontal schemes. *Parallel Comput.*, 3:193–204, 1986. (Cited on pages 221 and 224.)

[370] I. S. Duff. The parallel solution of sparse linear equations. In W. Handler, D. Haupt, R. Jeltsch, W. Juling, and O. Lange, editors, *International Conference on Parallel Processing*, pages 18–24, Berlin, Heidelberg, 1986. Springer. (Cited on page 221.)

[371] I. S. Duff. A review of frontal methods for solving linear systems. *Comput. Phys. Comm.*, 97:45–52, 1996. (Cited on page 221.)

[372] I. S. Duff, A. M. Erisman, and J. K. Reid. On George's nested dissection method. *SIAM J. Numer. Anal.*, 13(5):686–695, 1976. (Cited on page 207.)

[373] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2nd edition, 2017. First edition 1987. (Cited on pages 185 and 217.)

[374] I. S. Duff, N. I. Gould, M. Lescrenier, and J. K. Reid. The multifrontal method in a parallel environment. In M. G. Cox and S. Hammarling, editors, *Reliable Numerical Computation*, pages 93–111, London, 1990. Oxford University Press. (Cited on page 220.)

[375] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(4):889–901, 1999. (Cited on page 128.)

[376] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996, 2001. (Cited on page 128.)

[377] I. S. Duff and G. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989. (Cited on pages 375, 377, and 418.)

[378] I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Anal. Appl.*, 27(2):313–340, 2005. (Cited on page 128.)

[379] I. S. Duff and J. K. Reid. A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination. *J. Inst. Maths Applics*, 14:281–291, 1974. (Cited on page 217.)

[380] I. S. Duff and J. K. Reid. Some design features of a sparse matrix code. *ACM Trans. Math. Soft. (TOMS)*, 5(1):18–35, 1979. (Cited on page 217.)

[381] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Soft. (TOMS)*, 9(3):302–325, 1983. (Cited on pages 211, 213, and 224.)

[382] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Statist. Comput.*, 5(3):633–641, 1984. (Cited on pages 211, 216, 219, and 224.)

[383] I. S. Duff and J. K. Reid. The design of MA48: a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Soft. (TOMS)*, 22(2):187–226, 1996. (Cited on page 217.)

[384] I. S. Duff, J. K. Reid, and J. A. Scott. The use of profile reduction algorithms with a frontal code. *Internat. J. Numer. Methods Engrg.*, 28:2555–2568, 1989. (Cited on page 200.)

[385] J. Duintjer Tebbens and G. Meurant. Any Ritz value behavior is possible for Arnoldi and for GMRES. *SIAM J. Matrix Anal. Appl.*, 33(3):958–978, 2012. (Cited on page 359.)

[386] J. Duintjer Tebbens and G. Meurant. Prescribing the behavior of early terminating GMRES and Arnoldi iterations. *Numer. Algorithms*, 65(1):69–90, 2014. (Cited on page 359.)

[387] J. Duintjer Tebbens and G. Meurant. On the convergence of Q-OR and Q-MR Krylov methods for solving linear systems. *BIT Numerical Mathematics*, 56(1):77–97, 2016. (Cited on page 357.)

[388] J. Duintjer Tebbens, G. Meurant, H. Sadok, and Z. Strakoš. On investigating GMRES convergence using unitary matrices. *Linear Algebra Appl.*, 450:83–107, 2014. (Cited on page 359.)

[389] J. Duintjer Tebbens and M. Tůma. Efficient preconditioning of sequences of nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 29(5):1918–1941, 2007. (Cited on page 417.)

[390] J. Duintjer Tebbens and M. Tůma. Preconditioner updates for solving sequences of linear systems in matrix-free environment. *Numer. Linear Algebra Appl.*, 17(6):997–1019, 2010. (Cited on page 417.)

[391] T. F. Dupont, R. P. Kendall, and H. H. Rachford. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5(3):559–573, 1968. (Cited on pages 371 and 418.)

[392] A. Edelman and M. Ohlroch. Editor's note. SIAM J. Matrix Anal. Appl., vol 12, 1991. (Cited on page 123.)

[393] J. Egerváry. On rank-diminishing operations and their applications to the solution of linear equations. *Z. Angew. Math. Phys.*, 11:376–386, 1960. (Cited on pages 156 and 162.)

[394] M. Eiermann. On semiiterative methods generated by Faber polynomials. *Numer. Math.*, 56:139–156, 1989. (Cited on page 251.)

[395] M. Eiermann. Fields of values and iterative methods. *Linear Algebra Appl.*, 180:167–197, 1993. (Cited on page 316.)

[396] M. Eiermann and O. G. Ernst. Geometric aspects of the theory of Krylov subspace methods. *Acta Numerica*, 10:251–312, 2001. (Cited on pages 305, 316, and 356.)

[397] M. Eiermann, O. G. Ernst, and O. Schneider. Analysis of acceleration strategies for restarted minimal residual methods. *J. Comput. Appl. Math.*, 123(1):261–292, 2000. (Cited on pages 322, 323, and 359.)

[398] M. Eiermann, X. Li, and R. S. Varga. On hybrid semi-iterative methods. *SIAM J. Numer. Anal.*, 26(1):152–168, 1989. (Cited on page 251.)

[399] M. Eiermann and W. Niethammer. On the construction of semiiterative methods. *SIAM J. Numer. Anal.*, 20(6):1153–1160, 1983. (Cited on page 251.)

[400] M. Eiermann, W. Niethammer, and R. S. Varga. A study of semiiterative methods for nonsymmetric systems of linear equations. *Numer. Math.*, 47:505–533, 1985. (Cited on page 251.)

[401] V. Eijkhout. Analysis of parallel incomplete point factorizations. *Linear Algebra Appl.*, 154:723–740, 1991. (Cited on pages 376 and 420.)

[402] S. C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput.*, 2(1):1–4, 1981. (Cited on page 418.)

[403] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 20:345–367, 1983. (Cited on pages 316, 327, and 329.)

[404] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package I: The symmetric codes. *Internat. J. Numer. Methods Engrg.*, 18(8):1145–1151, 1982. (Cited on page 193.)

[405] S. C. Eisenstat, M. T. Heath, C. S. Henkel, and C. H. Romine. Modified cyclic algorithms for solving triangular systems on distributed memory multiprocessors. *SIAM J. Sci. Statist. Comput.*, 9(3):589–600, 1988. (Cited on page 139.)

[406] S. C. Eisenstat and J. W. H. Liu. Exploiting structural symmetry in unsymmetric sparse symbolic factorization. *SIAM J. Matrix Anal. Appl.*, 13(1):202–211, 1992. (Cited on page 216.)

[407] S. C. Eisenstat and J. W. H. Liu. Exploiting structural symmetry in a sparse partial pivoting code. *SIAM J. Sci. Comput.*, 14(1):253–257, 1993. (Cited on page 218.)

[408] S. C. Eisenstat and J. W. H. Liu. Structural representations of Schur complements in sparse matrices. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA Volumes in Applied Mathematics*, pages 85–100, New York, 1993. Springer Verlag. (Cited on page 216.)

[409] S. C. Eisenstat and J. W. H. Liu. The theory of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 26(3):686–705, 2005. (Cited on page 218.)

[410] S. C. Eisenstat and J. W. H. Liu. Algorithmic aspects of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 29(4):1363–1381, 2008. (Cited on page 218.)

[411] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Efficient implementation of sparse symmetric Gaussian elimination. In R. Vichnevetsky, editor, *Advances in Computer Methods for Partial Differential Equations*, pages 33–39. IMACS, 1975. (Cited on page 193.)

[412] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Applications of an element model for Gaussian elimination. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations*, pages 85–96. Academic Press, 1976. (Cited on page 193.)

[413] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Software for sparse Gaussian elimination with limited core storage. In I. S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 135–153. SIAM, 1979. (Cited on page 218.)

[414] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM J. Sci. Statist. Comput.*, 2:225–237, 1981. (Cited on page 193.)

[415] H. C. Elman. *Iterative methods for large sparse nonsymmetric systems of linear equations*. PhD thesis, Computer Science Dept., Yale University, USA, 1982. Research Report 229. (Cited on pages 316, 327, and 329.)

[416] H. C. Elman. Approximate Schur complement preconditioners on serial and parallel computers. *SIAM J. Sci. Statist. Comput.*, 10(3):581–605, 1989. (Cited on page 417.)

[417] H. C. Elman. Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems. *BIT Numerical Mathematics*, 29(4):890–915, 1989. (Cited on page 417.)

[418] H. C. Elman and X.-Z. Guo. Performance enhancements and parallel algorithms for two multilevel preconditioners. *SIAM J. Sci. Comput.*, 14(4):890–913, 1993. (Cited on page 415.)

[419] H. C. Elman, Y. Saad, and P. E. Saylor. A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations. *SIAM J. Sci. Statist. Comput.*, 7(3):840–855, 1986. (Cited on pages 388 and 420.)

[420] H. C. Elman and R. L. Streit. Polynomial iteration for nonsymmetric indefinite linear systems. In *Numerical Analysis*, pages 103–117, Berlin, Heidelberg, 1986. Springer. (Cited on pages 388 and 420.)

[421] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Rev.*, 46(1):3–45, 2004. (Cited on page 140.)

[422] M. Embree. How descriptive are GMRES convergence bounds? Technical Report 99/08, Mathematical Institute, University of Oxford, UK, 1999. (Cited on page 317.)

[423] M. Embree. The tortoise and the hare restart GMRES. *SIAM Rev.*, 45(2):259–266, 2003. (Cited on page 322.)

[424] M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel. *Refined Iterative Methods for Computations of the Solution and the Eigenvalues of Self-adjoint Boundary Value Problems*. Birkhäuser Verlag, Basel/Stuttgart, 1959. (Cited on page 303.)

[425] T. Ericsson. Computing functions of matrices using Krylov subspace methods. Technical Report TR ?, Chalmers University, Sweden, 1990. (Cited on page 67.)

[426] Y. A. Erlangga and R. Nabben. Deflation and balancing preconditioners for Krylov subspace methods applied to nonsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 30(2):684–699, 2008. (Cited on page 417.)

[427] Y. A. Erlangga and R. Nabben. Algebraic multilevel Krylov methods. *SIAM J. Sci. Comput.*, 31(5):3417–3437, 2009. (Cited on pages 415 and 417.)

[428] R. Estrin, D. Orban, and M. A. Saunders. Euclidean-norm error bounds for SYMMLQ and CG. *SIAM J. Matrix Anal. Appl.*, 40:235–253, 2019. (Cited on page 304.)

[429] R. Estrin, D. Orban, and M. A. Saunders. LNLQ: An iterative method for least-norm problems with an error minimization property. *SIAM J. Matrix Anal. Appl.*, 40(3):1102–1124, 2019. (Cited on page 304.)

[430] D. J. Evans. The use of preconditioning in iterative method for solving linear equations with symmetric positive definite matrices. *J. Inst. Maths Applics*, 4(3):295–314, 1967. (Cited on pages 365 and 420.)

[431] D. J. Evans. *Preconditioning Methods: Theory and Applications*. Gordon and Breach Science Publishers, Inc., 1983. (Cited on page 420.)

[432] D. J. Evans and M. Hatzopoulos. A parallel linear system solver. *Int. J. Comput. Math.*, 7(3):227–238, 1979. (Cited on page 163.)

[433] G. C. Everstine. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. *Internat. J. Numer. Methods Engrg.*, 14(6):837–853, 1979. (Cited on page 199.)

[434] V. Faber, A. Greenbaum, and D. E. Marshall. The polynomial numerical hulls of Jordan blocks and related matrices. *Linear Algebra Appl.*, 374:231–246, 2003. (Cited on page 317.)

[435] V. Faber, J. Liesen, and P. Tichý. The Faber-Manteuffel theorem for linear operators. *SIAM J. Numer. Anal.*, 46:1323–1337, 2008. (Cited on page 334.)

[436] V. Faber, J. Liesen, and P. Tichý. On orthogonal reduction to Hessenberg form with small bandwidth. *Numer. Algorithms*, 51(2):133–142, 2009. (Cited on page 334.)

[437] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21(2):352–362, 1984. (Cited on page 334.)

[438] D. K. Faddeev. Properties of the inverse of a Hessenberg matrix. *J. Sov. Math.*, 24(1):118–120, 1984. in Numerical Methods and Computational Issues, volume 5, V. P. Il'in and V. N. Kublanoskaya Eds., 1981 (in Russian). (Cited on pages 68 and 79.)

[439] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computer and Structures*, 28(5):579–602, 1988. (Cited on page 209.)

[440] C. Farhat, S. Lanteri, and H. D. Simon. TOP/DOMDEC a software tool for mesh partitioning and parallel processing. *Comput. Syst. Eng.*, 6:13–26, 1995. (Cited on page 209.)

[441] C. Farhat, M. Lesoinne, P. Le Tallec, K. Pierson, and D. Rixen. FETI-DP: A dual-primal unified FETI method -part I: A faster alternative to the two-level FETI method. *Internat. J. Numer. Methods Engrg.*, 50(7):1523–1544, 2001. (Cited on page 406.)

[442] C. Farhat, M. Lesoinne, and K. Pierson. A scalable dual-primal domain decomposition method. *Numer. Linear Algebra Appl.*, 7:687–714, 2000. (Cited on page 406.)

[443] C. Farhat and F. X. Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg.*, 32:1205–1227, 1991. (Cited on page 406.)

[444] M. Faverge, J. Herrmann, J. Langou, B. Lowery, Y. Robert, and J. J. Dongarra. Mixing LU and QR factorization algorithms to design high-performance dense linear algebra solvers. *J. Parallel Dist. Comput.*, 85:32–46, 2015. (Cited on page 145.)

[445] R. P. Fedorenko. A relaxation method for solving elliptic difference equations. *Zh. Vychisl. Mat. Mat. Fiz.*, 1:922–927, 1961. Also in *U.S.S.R. Comput. Math. and Math. Phys.*, 1 (1962), pp. 1092-1096. (Cited on page 421.)

[446] R. P. Fedorenko. The speed of convergence of one iterative process. *Zh. Vychisl. Mat. Mat. Fiz.*, 4:559–563, 1964. Also in *U.S.S.R. Comput. Math. and Math. Phys.*, 4 (1964), pp. 227-235. (Cited on page 421.)

[447] R. P. Fedorenko. Iterative methods for elliptic difference equations. *Russian Math. Surveys*, 28(2):129–195, 1973. (Cited on page 421.)

[448] S. Feuerriegel and H. M. Bücker. The non-symmetric s-step Lanczos algorithm: Derivation of efficient recurrences and synchronization-reducing variants of BiCG and QMR. *Int. J. Appl. Math. Comput. Sci.*, 25(4):769–785, 2015. (Cited on page 338.)

[449] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. 19th Design Automation Conference, IEEE, 1982. (Cited on page 209.)

[450] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.*, 25:619–633, 1975. (Cited on page 201.)

[451] M. Fiedler. *Special Matrices and their Applications in Numerical Mathematics*. Martinus Nijhoff Publishers, Dordrecht, The Netherlands, 1st edition, 1986. reprinted by Dover (2008). (Cited on page 108.)

[452] M. Fiedler and V. Pták. On matrices with non-positive off-diagonal elements and positive principal minors. *Czechoslovak Math. J.*, 12(3):382–400, 1962. (Cited on page 108.)

[453] P. Fika, M. Mitrouli, and P. Roupa. Estimates for the bilinear form $x^T A^{-1} y$ with applications to linear algebra problems. *Electron. Trans. Numer. Anal.*, 43:70–89, 2014. (Cited on page 284.)

[454] P. Fika, M. Mitrouli, and O. Turek. On the estimation of $x^T A^{-1} x$ for symmetric matrices. *The Electron. J. Linear Algebra*, 37:549–561, 2021. (Cited on page 284.)

[455] B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*. Wiley Teubner, 1996. (Cited on pages 286, 290, 304, and 388.)

[456] B. Fischer and R. W. Freund. On adaptive weighted polynomial preconditioning for Hermitian positive definite matrices. *SIAM J. Sci. Comput.*, 15(2):408–426, 1994. (Cited on page 417.)

[457] B. Fischer and G. H. Golub. On the error computation for polynomial based iteration methods. In A. Greenbaum and M. Luskin, editors, *Recent Advances in Iterative Methods*, pages 59–67, New York, 1994. Springer. (Cited on page 304.)

[458] C. F. Fischer and R. A. Usmani. Properties of some tridiagonal matrices and their application to boundary value problems. *SIAM J. Numer. Anal.*, 6(1):127–142, 1969. (Cited on page 79.)

[459] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Numerical Analysis. Lecture Notes in Mathematics, vol 506*, pages 73–89, Berlin, Heidelberg, 1976. Springer. (Cited on pages 336 and 360.)

[460] R. Fletcher. Dense factors of sparse matrices. In M. D. Buchmann and A. Iserles, editors, *Approximation Theory and Optimization, Tributes to M.J.D. Powell*, pages 145–166. ambridge University Press, 1997. (Cited on page 157.)

[461] G. E. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1967. (Cited on pages 3 and 128.)

[462] G. E. Forsythe and E. G. Straus. On best conditioned matrices. *Proc. Am. Math. Soc.*, 6:340–345, 1955. (Cited on pages 128 and 364.)

[463] L. V. Foster. Gaussian elimination with partial pivoting can fail in practice. *SIAM J. Matrix Anal. Appl.*, 15(4):1354–1362, 1994. (Cited on page 123.)

[464] L. Fox. *An Introduction to Numerical Linear Algebra*. Oxford University Press, 1964. (Cited on pages 159 and 160.)

[465] L. Fox, H. D. Huskey, and J. H. Wilkinson. Notes on the solution of algebraic linear simultaneous equations. *Quart. J. Mech. Appl. Math.*, 1(1):149–173, 1948. (Cited on page 158.)

[466] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput.*, 23(2):442–462, 2001. (Cited on page 417.)

[467] R. W. Freund. On conjugate gradient type methods and polynomial preconditioners for a class of complex non Hermitian matrices. *Numer. Math.*, 57:285–312, 1990. (Cited on page 417.)

[468] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993. (Cited on pages 347 and 360.)

[469] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part I. Technical Report RIACS 90.45, NASA Ames Research Center, USA, 1990. (Cited on pages 345 and 360.)

[470] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993. (Cited on pages 345 and 360.)

[471] R. W. Freund and M. Hochbruck. A biconjugate gradient type algorithm on massively parallel architectures. In R. Vichnevetsky and J. J. H. Miller, editors, *IMACS'91*, pages 720–721, Dublin, 1991. Criterion Press. (Cited on page 347.)

[472] R. W. Freund and M. Hochbruck. A biconjugate gradient-type algorithm for the iterative solution of non-Hermitian linear systems on massively parallel architectures. In C. Brezinski and U. Kulisch, editors, *Computational and Applied Mathematics, I: Algorithms and Theory*, pages 169–178, Amsterdam, 1992. North Holland. (Cited on page 347.)

[473] R. W. Freund and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part II. Technical Report RIACS 90.46, NASA Ames Reasearch Center, USA, 1990. (Cited on pages 345 and 360.)

[474] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60(3):315–339, 1991. (Cited on pages 336, 345, 347, and 360.)

[475] R. W. Freund and N. M. Nachtigal. Implementation details of the coupled QMR algorithm. Technical Report RIACS 92.19, NASA Ames Research Center, USA, 1992. (Cited on page 360.)

[476] R. W. Freund and N. M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. Technical Report RIACS 92.15, NASA Ames Research Center, USA, 1992. (Cited on pages 345 and 360.)

[477] R. W. Freund and N. M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. *SIAM J. Sci. Comput.*, 15(2):313–337, 1994. (Cited on pages 336, 345, 347, and 360.)

[478] R. W. Freund and N. M. Nachtigal. Software for simplified Lanczos and QMR algorithms. *Appl. Numer. Math.*, 19(3):319–341, 1995. (Cited on page 345.)

[479] A. Frommer, K. Kahl, T. Lippert, and H. Rittich. 2-norm error bounds and estimates for Lanczos approximations to linear systems and rational matrix functions. *SIAM J. Matrix Anal. Appl.*, 34(3):1046–1065, 2013. (Cited on page 304.)

[480] A. Frommer and D. B. Szyld. An algebraic convergence theory for restricted additive Schwarz methods using weighted max norms. *SIAM J. Matrix Anal. Appl.*, 39(2):463–479, 2001. (Cited on page 394.)

[481] S. Fujino and S. Doi. Optimizing multicolor ICCG methods on some vector computers. In R. Beauwens, editor, *Proceedings IMACS Int. Symp. Iterative Methods in Linear Algebra*. North Holland, 1991. (Cited on page 377.)

[482] R. E. Funderlic, M. Neumann, and R. J. Plemmons. LU decompositions of generalized diagonally dominant matrices. *Numer. Math.*, 40:57–69, 1982. (Cited on page 109.)

[483] R. E. Funderlic and R. J. Plemmons. LU decomposition of M-matrices by elimination without pivoting. *Linear Algebra Appl.*, 41:41–99, 1981. (Cited on page 109.)

[484] A. Galántai. *Projectors and Projection Methods*. Kluwer Academic Publishers, Boston, 2004. (Cited on page 156.)

[485] A. Galántai. The rank reduction procedure of Egerváry. *Cent. Eur. J. Oper. Res.*, 18(1):5–24, 2009. (Cited on page 156.)

[486] E. Gallopoulos, B. Philippe, and A. Sameh. *Parallelism in Matrix Computations*. Springer, Dordrecht, 2016. (Cited on page 139.)

[487] M. J. Gander. Schwarz methods over the course of time. *Electron. Trans. Numer. Anal.*, 31(5):228–255, 2008. (Cited on pages 389, 392, 393, and 421.)

[488] M. J. Gander. On the origins of linear and non-linear preconditioning. In C.-O. Lee, X.-C. Cai, D. E. Keyes, H. H. Kim, A. Klawonn, E.-J. Park, and O. B. Widlund, editors, *Domain decomposition methods in science and engineering XXIII*, pages 153–161. Springer, 2017. (Cited on page 417.)

[489] M. J. Gander, S. Loisel, and D. B. Szyld. An optimal block iterative method and preconditioner for banded matrices with applications to PDEs on irregular domains. *SIAM J. Matrix Anal. Appl.*, 33(2):653–680, 2012. (Cited on page 417.)

[490] M. J. Gander and X. Tu. On the origins of iterative substructuring methods. In J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXI*, pages 597–605. Springer, 2014. (Cited on page 389.)

[491] M. J. Gander and G. Wanner. The origins of the alternating Schwarz method. In J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXI*, pages 487–495. Springer, 2014. (Cited on page 389.)

[492] M. J. Gander and H. Zhang. Optimized interface preconditioners for the FETI method. In J. Erhel, M. J. Gander, L. Halpern, G. Pichot, T. Sassi, and O. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXI*, pages 657–665. Springer, 2014. (Cited on page 406.)

[493] M. J. Gander and H. Zhang. Schwarz methods by domain truncation. *Acta Numerica*, 31:1–134, 2022. (Cited on page 394.)

[494] F. R. Gantmacher. *The Theory of Matrices, Vol. 1*. AMS Chelsea Publishing, Providence, RI, 1959. (Cited on pages 73 and 84.)

[495] I. García, J. J. Merelo, J. D. Bruguera, and E. L. Zapata. Parallel quadrant interlocking factorization on hypercube computers. *Parallel Comput.*, 15(1-3):87–100, 1990. (Cited on page 167.)

[496] M. Gates, J. Kurzak, A. Charara, A. YarKhan, and J. J. Dongarra. SLATE: Design of a modern distributed and accelerated linear algebra library. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page Article No.: 26. ACM, 2019. (Cited on page 141.)

[497] A. Gaul, M. H. Gutknecht, J. Liesen, and R. Nabben. A framework for deflated and augmented Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 34(2):495–518, 2013. (Cited on page 323.)

[498] C. F. Gauss. Disquisitio de elementis ellipticis palladis. In *Carl Friedrich Gauss Werke, Vol VI*, pages 1–24. Königlichen Gesellschaft der Wissenschaften zu Göttingen, 1810. (Cited on page 182.)

[499] G. A. Geist. Efficient parallel LU factorization with pivoting on a hypercube multiprocessor. Technical Report ORNL-6211, Oak Ridge Nat. Laboratory, 1985. (Cited on page 141.)

[500] G. A. Geist and M. T. Heath. Matrix factorization on a hypercube multiprocessor. In M. T. Heath, editor, *Hypercube Multiprocessors 1986*, pages 161–180. SIAM, Philadelphia, 1986. (Cited on page 139.)

[501] G. A. Geist and C. H. Romine. LU factorization algorithms on distributed memory multiprocessor architectures. *SIAM J. Sci. Statist. Comput.*, 9(4):639–649, 1988. (Cited on page 141.)

[502] J. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10(2):345–363, 1973. (Cited on pages 206, 207, and 224.)

[503] J. A. George. Numerical experiments using dissection methods to solve n by n grid problems. *SIAM J. Numer. Anal.*, 14(2):161–179, 1977. (Cited on page 224.)

[504] J. A. George, M. T. Heath, J. W. H. Liu, and E. G. Ng. Solution of sparse positive definite systems on a shared-memory multiprocessor. *Int. J. Parallel Program.*, 15(4):309–325, 1986. (Cited on pages 186 and 193.)

[505] J. A. George, M. T. Heath, J. W. H. Liu, and E. G. Ng. Sparse Cholesky factorization on a local memory multiprocessor. *SIAM J. Sci. Statist. Comput.*, 9(2):327–340, 1988. (Cited on pages 186 and 193.)

[506] J. A. George, M. T. Heath, J. W. H. Liu, and E. G. Ng. Solution of sparse positive definite systems on a hypercube. *J. Comput. Appl. Math.*, 27:129–156, 1989. (Cited on pages 186 and 221.)

[507] J. A. George and J. W. H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15(5):1053–1069, 1978. (Cited on page 206.)

[508] J. A. George and J. W. H. Liu. An implementation of a pseudo-peripheral node finder. *ACM Trans. Math. Soft. (TOMS)*, 5(3):284–295, 1979. (Cited on page 196.)

[509] J. A. George and J. W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Soft. (TOMS)*, 6(3):337–358, 1980. (Cited on page 204.)

[510] J. A. George and J. W. H. Liu. A minimal storage implementation of the minimum degree algorithm. *SIAM J. Numer. Anal.*, 17(2):282–299, 1980. (Cited on page 204.)

[511] J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981. (Cited on pages 185, 192, 193, 196, and 224.)

[512] J. A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989. (Cited on pages 204, 205, and 224.)

[513] J. A. George and D. R. McIntyre. On the application of the minimum degree algorithm to finite element systems. *SIAM J. Numer. Anal.*, 15(1):90–112, 1978. (Cited on page 204.)

[514] J. A. George and E. G. Ng. An implementation of Gaussian elimination with partial pivoting for sparse systems. *SIAM J. Sci. Statist. Comput.*, 6(2):390–409, 1985. (Cited on page 218.)

[515] J. A. George and E. G. Ng. Symbolic factorization for sparse Gaussian elimination with partial pivoting. *SIAM J. Sci. Statist. Comput.*, 8(6):877–898, 1987. (Cited on page 218.)

[516] J. A. George and E. G. Ng. Parallel sparse Gaussian elimination with partial pivoting. *Annals of Oper. Res.*, 22(1):219–240, 1990. (Cited on page 218.)

[517] J. A. George and A. Pothen. An analysis of spectral envelope reduction via quadratic assignment problems. *SIAM J. Matrix Anal. Appl.*, 18(3):706–732, 1997. (Cited on page 201.)

[518] T. George, V. Saxena, A. Gupta, A. Singh, and A. R. Choudhury. Multifrontal factorization of sparse SPD matrices on GPUs. In *Parallel Distributed Processing Symposium (IPDPS)*, pages 372–383. IEEE, 2011. (Cited on page 222.)

[519] T. Gergelits, K. A. Mardal, B. F. Nielsen, and Z. Strakoš. Laplacian preconditioning of elliptic PDEs: Localization of the eigenvalues of the discretized operator. *SIAM J. Numer. Anal.*, 57(3):1369–1394, 2019. (Cited on page 417.)

[520] T. Gergelits and Z. Strakoš. Composite convergence bounds based on Chebyshev polynomials and finite precision conjugate gradient computations. *Numer. Algorithms*, 65(4):759–782, 2014. (Cited on page 276.)

[521] P. Ghysels and R. Synk. High performance sparse multifrontal solvers on modern GPUs. *Parallel Comput.*, 110:102897, 2022. (Cited on page 222.)

[522] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Comput.*, 40(7):224–238, 2014. (Cited on pages 286, 298, and 304.)

[523] N. E. Gibbs. Algorithm 509: A hybrid profile reduction algorithm. *ACM Trans. Math. Soft. (TOMS)*, 2(4):378–387, 1976. (Cited on page 199.)

[524] N. E. Gibbs, W. G. Poole Jr., and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.*, 13(2):236–250, 1976. (Cited on page 195.)

[525] J. R. Gilbert and L. Grigori. A note on the column elimination tree. *SIAM J. Matrix Anal. Appl.*, 25(1):143–151, 2003. (Cited on page 218.)

[526] J. R. Gilbert, X. S. Li, E. G. Ng, and B. W. Peyton. Computing row and column counts for sparse QR and LU factorization. *BIT*, 41(4):693–710, 2001. (Cited on page 193.)

[527] J. R. Gilbert and J. W. H. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM J. Matrix Anal. Appl.*, 14(2):334–352, 1993. (Cited on pages 217 and 218.)

[528] J. R. Gilbert, C. B. Moler, and R. S. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, 1992. (Cited on page 206.)

[529] J. R. Gilbert and E. G. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA Volumes in Applied Mathematics*, pages 107–139, New York, 1993. Springer Verlag. (Cited on page 218.)

[530] J. R. Gilbert, E. G. Ng, and B. W. Peyton. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15(4):1075–1091, 1994. (Cited on page 193.)

[531] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.*, 9(5):862–874, 1988. (Cited on page 218.)

[532] L. Giraud and S. Gratton. On the sensitivity of some spectral preconditioners. *SIAM J. Matrix Anal. Appl.*, 27(4):1089–1105, 2006. (Cited on page 417.)

[533] L. Giraud and J. Langou. Another proof for modified Gram-Schmidt with reorthogonalization. Technical Report WN/PA/02/53, CERFACS, 2002. (Cited on page 48.)

[534] L. Giraud, J. Langou, and M. Rozložník. On the loss of orthogonality in the Gram-Schmidt orthogonalization process. *Comp. Math. Appl.*, 50:1069–1075, 2005. (Cited on page 48.)

[535] L. Giraud, J. Langou, M. Rozložník, and J. van den Eshof. Rounding error analysis of the classical Gram-Schmidt orthogonalization process. *Numer. Math.*, 101:87–100, 2005. (Cited on page 48.)

[536] R. Glowinski, G. H. Golub, G. Meurant, and J. Périaux Eds. *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*. SIAM, 1988. (Cited on page 420.)

[537] D. Golberg. What every computer scientist should know about floating point arithmetic. *ACM Comp. Surveys*, 23(1):5–48, 1991. (Cited on page 78.)

[538] H. H. Goldstine and J. von Neumann. Numerical inverting of matrices of high order. II. *Proc. Am. Math. Soc.*, 2(2):188–202, 1951. (Cited on page 123.)

[539] G. H. Golub. Bounds for the round-off errors in the Richardson second order method. *BIT Numerical Mathematics*, 2(4):212–223, 1962. (Cited on pages 250 and 252.)

[540] G. H. Golub and D. Mayers. The use of preconditioning over irregular regions. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Science and Engineering VI*, pages 3–14. North-Holland, 1984. (Cited on page 398.)

[541] G. H. Golub and G. Meurant. Matrices, moments and quadrature. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1993*, volume 303 of *Pitman Research Notes in Mathematics*, pages 105–156. Longman Sci. Tech., 1994. Reprinted in [218]. (Cited on page 304.)

[542] G. H. Golub and G. Meurant. Matrices, moments and quadrature II or how to compute the norm of the error in iterative methods. *BIT*, 37(3):687–705, 1997. (Cited on pages 283 and 304.)

[543] G. H. Golub and G. Meurant. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2010. (Cited on pages 282, 283, and 304.)

[544] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948-1976. *SIAM Rev.*, 31(1):50–102, 1989. (Cited on page 304.)

[545] G. H. Golub and M. L. Overton. The convergence of inexact Chebyshev and Richardson iterative methods for solving linear systems. *Numer. Math.*, 53(5):571–593, 1988. (Cited on page 252.)

[546] G. H. Golub and Z. Strakoš. Estimates in quadratic formulas. *Numer. Algorithms*, 8(2):241–268, 1994. (Cited on pages 279 and 304.)

[547] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 2nd edition, 1989. (Cited on pages 125, 286, and 379.)

[548] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 3rd edition, 1996. (Cited on pages 3, 38, 46, 71, 107, 313, and 379.)

[549] G. H. Golub and J. M. Varah. On a characterization of the best ℓ_2-scaling of a matrix. *SIAM J. Numer. Anal.*, 11(3):472–479, 1974. (Cited on page 128.)

[550] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods. I. *Numer. Math.*, 3(1):147–156, 1961. (Cited on page 250.)

[551] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods. II. *Numer. Math.*, 3(1):157–168, 1961. (Cited on page 250.)

[552] N. Gould. On growth in Gaussian elimination with complete pivoting. *SIAM J. Matrix Anal. Appl.*, 12(2):354–361, 1991. (Cited on page 123.)

[553] N. I. Gould and J. A. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM J. Sci. Comput.*, 19(2):605–625, 1998. (Cited on pages 379, 380, 417, and 420.)

[554] J. P. Gram. Ueber die entwickelung reeller functionen in reihen mittelst der methode der kleinsten quadrate. *Journal für die reine und angewandte Mathematik, (Crelle Journal)*, 94:41–73, 1883. (Cited on page 78.)

[555] T. Grauschopf, M. Griebel, and H. Regler. Additive multilevel preconditioners based on bilinear interpolation, matrix-dependent geometric coarsening and algebraic multigrid coarsening for second-order elliptic PDEs. *Appl. Numer. Math.*, 23(1):63–95, 1997. (Cited on page 415.)

[556] J. F. Grcar. How ordinary elimination became Gaussian elimination. *Historia Math.*, 38(2):163–218, 2011. (Cited on page 183.)

[557] J. F. Grcar. John von Neumann's analysis of Gaussian elimination and the origins of modern Numerical Analysis. *SIAM Rev.*, 53(4):607–682, 2011. (Cited on page 183.)

[558] J. F. Grcar. Mathematicians of Gaussian elimination. *Notices Amer. Math. Soc.*, 58(6):782–792, 2011. (Cited on pages 182 and 183.)

[559] J. F. Grcar. Review of The Chinese Roots of Linear Algebra by Roger Hart. ArXiv preprint arXiv:1104.2282, 2011. (Cited on page 182.)

[560] A. Greenbaum. *Convergence properties of the conjugate gradient algorithm in exact and finite precision arithmetic*. PhD thesis, University of California, Berkeley, USA, 1981. (Cited on pages 278 and 303.)

[561] A. Greenbaum. Behavior of slightly perturbed Lanczos and conjugate gradient recurrences. *Linear Algebra Appl.*, 113:7–63, 1989. (Cited on pages 278 and 303.)

[562] A. Greenbaum. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. In J. D. Brown, M. T. Chu, D. C. Ellison, and R.J. Plemmons, editors, *Proceedings of the Cornelius Lanczos International Centenary Conference (1993)*, pages 49–60. SIAM, 1994. (Cited on pages 278 and 303.)

[563] A. Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Appl.*, 1(3):535–551, 1997. (Cited on pages 265, 296, 304, and 337.)

[564] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, 1997. (Cited on pages 278 and 303.)

[565] A. Greenbaum. Generalizations of the field of values useful in the study of polynomial functions of a matrix. *Linear Algebra Appl.*, 347:233–249, 2002. (Cited on page 317.)

[566] A. Greenbaum. Some theoretical results derived from polynomial numerical hulls of Jordan blocks. *Electron. Trans. Numer. Anal.*, 18:81–90, 2004. (Cited on page 317.)

[567] A. Greenbaum. Upper and lower bounds on norms of functions of matrices. *Linear Algebra Appl.*, 430:52–65, 2009. (Cited on page 317.)

[568] A. Greenbaum, H. Liu, and T. Chen. On the convergence rate of variants of the conjugate gradient algorithm in finite precision arithmetic. *SIAM J. Sci. Comput.*, 43(5):S496–S515, 2019. (Cited on page 286.)

[569] A. Greenbaum, V. Pták, and Z. Strakoš. Any convergence curve is possible for GMRES. *SIAM J. Matrix Anal. Appl.*, 17(3):465–470, 1996. (Cited on page 359.)

[570] A. Greenbaum and G. H. Rodrigue. Optimal preconditioners of a given sparsity pattern. *BIT Numerical Mathematics*, 29(4):610–634, 1989. (Cited on page 365.)

[571] A. Greenbaum and Z. Strakoš. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM J. Matrix Anal. Appl.*, 13(1):121–137, 1992. (Cited on pages 278 and 303.)

[572] A. Greenbaum and Z. Strakoš. Matrices that generate the same Krylov residual spaces. In G. H. Golub, A. Greenbaum, and M. Luskin, editors, *Recent Advances in Iterative Methods*, pages 95–118. Springer, 1994. (Cited on page 359.)

[573] L. Grigori, M. Cosnard, and E. G. Ng. On the row merge tree for sparse LU factorization with partial pivoting. *BIT Numerical Mathematics*, 47(1):45–76, 2007. (Cited on page 218.)

[574] L. Grigori, J. W. Demmel, and H. Xiang. Communication avoiding Gaussian elimination. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12. IEEE, 2008. (Cited on page 143.)

[575] L. Grigori, J. W. Demmel, and H. Xiang. CALU: A communication optimal LU factorization algorithm. *SIAM J. Matrix Anal. Appl.*, 32(4):1317–1350, 2011. (Cited on pages 96 and 143.)

[576] L. Grigori, J. R. Gilbert, and M. Cosnard. Symbolic and exact structure prediction for sparse Gaussian elimination with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 30(4):1520–1545, 2009. (Cited on page 218.)

[577] L. Grigori and S. Moufawad. Communication avoiding ILU0 preconditioner. *SIAM J. Sci. Comput.*, 37(2):C217–C246, 2015. (Cited on page 417.)

[578] M. J. Grote and H. D. Simon. Parallel preconditioning and approximate inverses on the Connection Machine. In R. Sincovec, editor, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 519–523. SIAM, 1993. (Cited on page 420.)

[579] P. Guillaume, Y. Saad, and M. Sosonkina. Rational approximation preconditioners for sparse linear systems. *J. Comput. Appl. Math.*, 158(2):419–442, 2003. (Cited on page 417.)

[580] R. Guo and R. D. Skeel. An algebraic hierarchical basis preconditioner. *Appl. Numer. Math.*, 9:21–32, 1992. (Cited on page 417.)

[581] A. Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM J. Matrix Anal. Appl.*, 24:529–552, 2002. (Cited on page 219.)

[582] A. Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Trans. Math. Soft. (TOMS)*, 28(3):301–324, 2002. (Cited on page 219.)

[583] J. Gustafson. *The End of Error, Unum Computing*. CRC Press, 2015. (Cited on page 18.)

[584] J. Gustafson and I. Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomput. Front. Innov.*, 4(2):71–86, 2017. (Cited on page 18.)

[585] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978. (Cited on page 371.)

[586] I. Gustafsson and G. Lindskog. A preconditioning technique based on element matrix factorization. *Comput. Methods Appl. Mech. Engrg.*, 55:201–220, 1986. (Cited on page 417.)

[587] F. G. Gustavson. Some basic techniques for solving sparse systems of linear equations. In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and their Applications*, pages 41–52. Plenum Press, 1972. (Cited on page 224.)

[588] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear algebra algorithms. *IBM J. Res. Dev.*, 41(6):731–755, 1997. (Cited on page 144.)

[589] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms. Part I. *SIAM J. Matrix Anal. Appl.*, 13(2):594–639, 1992. (Cited on page 336.)

[590] M. H. Gutknecht. Changing the norm in conjugate gradient type algorithms. *SIAM J. Numer. Anal.*, 30:40–56, 1993. (Cited on pages 343 and 360.)

[591] M. H. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14(5):1020–1033, 1993. (Cited on page 343.)

[592] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part II. *SIAM J. Matrix Anal. Appl.*, 15:15–58, 1994. (Cited on page 336.)

[593] M. H. Gutknecht. Lanczos-type solvers for nonsymmetric linear systems of equations. *Acta Numerica*, 6:271–397, 1997. (Cited on pages 335, 336, and 360.)

[594] M. H. Gutknecht. IDR explained. *Electron. Trans. Numer. Anal.*, 36(3):126–148, 2010. (Cited on pages 351 and 361.)

[595] M. H. Gutknecht and Z. Strakoš. Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM J. Matrix Anal. Appl.*, 22(1):213–229, 2000. (Cited on page 304.)

[596] H.-C. Lee and A. J. Wathen. On element-by-element preconditioning for general elliptic problems. *Comput. Methods Appl. Mech. Engrg.*, 92(2):215–229, 1991. (Cited on page 417.)

[597] M. Habu and T. Nodera. GMRES(m) algorithm with changing the restart cycle adaptively. In *Proceedings of Algorithmy 2000 Conference on Scientific Computing*, pages 354–263, 2000. (Cited on page 324.)

[598] W. Hackbusch. Convergence of multigrid iterations applied to difference equations. *Math. Comput.*, 34(150):425–440, 1980. (Cited on page 421.)

[599] W. Hackbusch. *Multigrid Methods and Applications*. Springer, 1985. (Cited on pages 411 and 421.)

[600] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer, 1994. (Cited on pages 410 and 411.)

[601] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Springer, 2015. (Cited on page 223.)

[602] W. Hackbusch and U. Trottenberg. *Multigrid Methods*. Springer, 1982. Lecture Notes in Mathematics 960. (Cited on page 421.)

[603] S. M. Hadfield and T. A. Davis. Potential and achievable parallelism in the unsymmetric-pattern multifrontal LU factorization method for sparse matrices. In *Fifth SIAM Conference on Applied Linear Algebra*. SIAM, 1994. (Cited on page 221.)

[604] A. Hadjidimos. Accelerated overrelaxation method. *Math. Comput.*, 32(141):149–157, 1978. (Cited on page 238.)

[605] A. Hadjidimos. Successive overrelaxation (SOR) and related methods. *J. Comput. Appl. Math.*, 123(1-2):177–199, 2000. (Cited on page 238.)

[606] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, 1981. Reprinted by Dover (2003). (Cited on page 235.)

[607] W. W. Hager. Condition estimates. *SIAM J. Sci. Statist. Comput.*, 5(2):311–316, 1984. (Cited on pages 128 and 220.)

[608] W. W. Hager. Minimizing the profile of a symmetric matrix. *SIAM J. Sci. Comput.*, 23(5):1799–1816, 2002. (Cited on page 199.)

[609] A. Haidar, H. Bayraktar, S. Tomov, J. J. Dongarra, and N. J. Higham. Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proc. R. Soc. A*, 476(2243):20200110, 2020. (Cited on pages 129 and 146.)

[610] A. Haidar, S. Tomov, J. J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 603–613. IEEE, 2018. (Cited on page 146.)

[611] E. Hallman. Sharp 2-norm error bounds for LSQR and the conjugate gradient method. *SIAM J. Matrix Anal. Appl.*, 41(3):1183–1207, 2020. (Cited on page 304.)

[612] E. Hallman. A refined probabilistic error bound for sums. ArXiv preprint arXiv:2104.06531, 2021. (Cited on page 30.)

[613] E. Hallman and I. C. F. Ipsen. Deterministic and probabilistic error bounds for floating point summation algorithms. ArXiv preprint arXiv:2107.01604, 2021. (Cited on page 30.)

[614] E. Hallman and I. C. F. Ipsen. Precision-aware deterministic and probabilistic error bounds for floating point summation. ArXiv preprint arXiv:2203.15928, 2022. (Cited on page 30.)

[615] R. J. Hanson, F. T. Krogh, and C. L. Lawson. A proposal for standard linear algebra subprograms. Technical Report Memorandum 33-660, Jet Propulsion Laboratory, California, USA, 1973. (Cited on page 130.)

[616] F. Harary. A graph theoretic method for the complete reduction of a matrix with a view toward finding its eigenvalues. *J. Math. Phys.*, 38(1-4):104–111, 1959. (Cited on pages 78 and 223.)

[617] F. Harary. Sparse matrices and graph theory. In J. K. Reid, editor, *Large Sparse Sets of Linear Equations*, pages 139–150. Academic Press, 1971. (Cited on page 78.)

[618] M. A. Hassanein. Block Gauss-Huard algorithm with column pivoting on a hybrid CPU-GPU architecture. *Concurr. Comp.-Pract. E.*, 29(4):e3884, 2017. (Cited on page 153.)

[619] M. T. Heath and C. H. Romine. Parallel solution of triangular systems on distributed memory multiprocessors. *SIAM J. Sci. Statist. Comput.*, 9(3):558–588, 1988. (Cited on pages 139 and 140.)

[620] C. J. Hegedüs. Purcell's method, Egerváry and related results. *Cent. Eur. J. Oper. Res.*, 18(1):25–35, 2010. (Cited on pages 156 and 157.)

[621] D. Heller. On the efficient computation of recurrence relations. Technical Report NASA-CR-136766, NASA, 1974. (Cited on page 138.)

[622] D. Heller. A survey of parallel algorithms in numerical linear algebra. *SIAM Rev.*, 20(4):740–777, 1978. (Cited on page 138.)

[623] P. Henrici. *Elements of Numerical Analysis*. John Wiley, 1964. (Cited on page 123.)

[624] K. Hessenberg. Behandlung linearer eigenwertaufgaben mit hilfe der Hamilton-Cayleyschen gleichung. Technical Report 1) Bericht der Reihe Numerische Verfahren, Institut für Praktische Mathematik, Technische Hochschule Darmstadt, 1940. (Cited on pages 79 and 360.)

[625] K. Hessenberg. *Die berechnung der eigenwerte und eigenlösungen linearer gleichungesysteme*. PhD thesis, Technische Hochschule Darmstadt, Germany, 1940. (Cited on page 79.)

[626] M. R. Hestenes. The conjugate gradient method for solving linear systems. Technical Report INA 54-11, National Bureau of Standards, 1954. (Cited on page 302.)

[627] M. R. Hestenes. Inversion of matrices by biorthogonalization and related results. *J. Soc. Ind. Appl. Math.*, 6(1):51–90, 1958. (Cited on page 159.)

[628] M. R. Hestenes. *Conjugate Direction Methods in Optimization*. Springer, 1980. (Cited on page 302.)

[629] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49(6):409–436, 1952. (Cited on pages 159, 261, 280, 302, and 418.)

[630] J. E. Hicken and D. W. Zingg. A simplified and flexible variant of GCROT for solving nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 32(3):1672–1694, 2010. (Cited on page 329.)

[631] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996. (Cited on pages 3, 46, 47, 78, 125, 183, 251, and 252.)

[632] N. J. Higham. Iterative refinement for linear systems and LAPACK. *IMA J. Numer. Anal.*, 17(4):495–509, 1997. (Cited on page 129.)

[633] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 2nd edition, 2002. (Cited on pages 3, 15, 21, 24, 27, 29, 42, 44, 47, 78, 119, 120, 124, 125, 148, 167, 183, 251, and 252.)

[634] N. J. Higham. Error analysis for standard and GMRES-based iterative refinement in two and three-precisions. Technical Report MIMS EPrint 2019.19, The University of Manchester, 2019. (Cited on page 129.)

[635] N. J. Higham and D. J. Higham. Large growth factors in Gaussian elimination with pivoting. *SIAM J. Matrix Anal. Appl.*, 10(2):155–164, 1989. (Cited on page 123.)

[636] N. J. Higham and T. Mary. A new approach to probabilistic rounding error analysis. *SIAM J. Sci. Comput.*, 41(5):A2815–A2835, 2019. (Cited on page 124.)

[637] N. J. Higham and T. Mary. A new preconditioner that exploits low-rank approximations to factorization error. *SIAM J. Sci. Comput.*, 41(1):A59–A82, 2019. (Cited on page 416.)

[638] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58:13–30, 1963. (Cited on page 124.)

[639] M. F. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, University of California at Berkeley, USA, 2010. (Cited on page 338.)

[640] W. Hoffmann. Solving linear systems on a vector computer. *J. Comput. Appl. Math.*, 18:353–367, 1987. (Cited on page 149.)

[641] W. Hoffmann. *Basic transformations in linear algebra for vector computing*. PhD thesis, University of Amsterdam, 1989. (Cited on pages 149 and 153.)

[642] W. Hoffmann. The Gauss-Huard algorithm and LU factorization. *Linear Algebra Appl.*, 275:281–286, 1998. (Cited on page 153.)

[643] W. Hoffmann, K. Potma, and G. Pronk. Solving dense linear systems by Gauss-Huard's method on a distributed memory system. *Future Gener. Comp. Syst.*, 10(2-3):21–325, 1994. (Cited on page 153.)

[644] J. D. Hogg, E. Ovtchinnikov, and J. A. Scott. A sparse symmetric indefinite direct solver for GPU architectures. *ACM Trans. Math. Soft. (TOMS)*, 42(1):1–25, 2016. (Cited on page 222.)

[645] J. D. Hogg and J. A. Scott. A fast and robust mixed-precision solver for the solution of sparse symmetric linear systems. *ACM Trans. Math. Soft. (TOMS)*, 37(2):1–19, 2010. Art. 17. (Cited on page 223.)

[646] J. D. Hogg and J. A. Scott. An efficient analyse phase for element problems. *Numer. Linear Algebra Appl.*, 20(3), 2013. (Cited on page 193.)

[647] A. S. Householder. *Principles of Numerical Analysis*. McGraw-Hill Book Company, New York, 1953. (Cited on pages 3, 52, and 78.)

[648] A. S. Householder. On norms of vectors and matrices. Technical Report ORNL-1756, Oak Ridge National Laboratory, 1954. (Cited on page 78.)

[649] A. S. Householder. Terminating and nonterminating iterations for solving linear systems. *J. Soc. Ind. Appl. Math.*, 3(2):67–72, 1955. (Cited on page 78.)

[650] J. Høyrup. The tortuous ways toward a new understanding of algebra in the Italian abbacus school (14th-16th centuries). In O. Figueras, J. L. Cortina, S. Alatorre, T. Rojano, and A. Sepulveda, editors, *Proceedings of the Joint Meeting PME32-PME NA XXX*, pages 1–15. Cinvestav-UMSNH, 2008. (Cited on page 182.)

[651] J. Høyrup. Old babylonian "algebra", and what it teaches us about possible kinds of mathematics. *Ganita Bhāratī*, 32:87–110, 2012. (Cited on page 182.)

[652] J. Hrnčíř, I. Pultarová, and Z. Strakoš. Decomposition into subspaces preconditioning: abstract framework. *Numer. Algorithms*, 83(1):57–98, 2020. (Cited on page 417.)

[653] H.-Y. Huang. A direct method for the general solution of a system of linear equations. *J. Optim. Theory Appl.*, 16(5-6):429–445, 1975. (Cited on page 156.)

[654] J. W. Huang and O. Wing. Optimal parallel triangulation of a sparse matrix. *IEEE Trans. Circuits and Systems*, 26(9):726–732, 1979. (Cited on page 204.)

[655] P. Huard. La méthode du simplexe sans inverse explicite. *EDF Bulletin de la Direction des Etudes et Recherches*, Série C, 2(2):79–98, 1979. (Cited on pages 151 and 183.)

[656] T. Huckle. Efficient computation of sparse approximate inverses. Technical Report TUM-19608, Institut fur Informatik, Technische Universitat Munchen, 1996. (Cited on page 379.)

[657] T. Huckle and M. Grote. A new approach to parallel preconditioning with sparse approximate inverses. Technical Report SCCM-94-03, Computer Science Dept., Stanford University, 1994. (Cited on pages 379, 380, and 420.)

[658] T. Huckle and M. Grote. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, 1997. (Cited on pages 379 and 420.)

[659] T. Huckle and J. Staudacher. Matrix multilevel methods and preconditioning. *BIT Numerical Mathematics*, 42(3):541–560, 2002. (Cited on page 415.)

[660] T. J. R. Hughes, I. Levit, and J.Winget. An element-by-element solution algorithm for problems of structural and solid mechanics. *Comput. Methods Appl. Mech. Engrg.*, 36(2):241–254, 1983. (Cited on page 416.)

[661] D. Hysom and A. Pothen. Efficient parallel computation of ILU(k) preconditioners. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 29. ACM, 1999. (Cited on page 417.)

[662] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Sci. Comput.*, 22(6):2194–2215, 2001. (Cited on page 417.)

[663] Y. Ikebe. On inverses of Hessenberg matrices. *Linear Algebra Appl.*, 24:93–97, 1979. (Cited on pages 68 and 79.)

[664] Intel Corporation. BFLOAT16–hardware numerics definition. White paper, document number 338302-001US, 2018. (Cited on page 17.)

[665] I. C. F. Ipsen. Expressions and bounds for the GMRES residual. *BIT Numerical Mathematics*, 40(3):524–535, 2000. (Cited on page 359.)

[666] B. M. Irons. A frontal solution program for finite element analysis. *Internat. J. Numer. Methods Engrg.*, 2(1):5–32, 1970. (Cited on pages 211 and 224.)

[667] D. Irony and S. Toledo. Communication-efficient parallel dense lu using a 3-dimensional approach. In *Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing*, pages ?–? SIAM, 2001. (Cited on page 142.)

[668] M. Jankowski and H. Wozniakowski. Iterative refinement implies numerical stability. *BIT Numerical Mathematics*, 17(3):303–311, 1977. (Cited on page 129.)

[669] C. C. Jay Kuo and T. F. Chan. Multilevel filtering elliptic preconditioners. *SIAM J. Sci. Statist. Comput.*, 11:403–429, 1990. (Cited on page 415.)

[670] C. C. Jay Kuo and T. F. Chan. Multilevel filtering elliptic preconditioners: extensions to more general elliptic problems. *SIAM J. Sci. Statist. Comput.*, 13:227–243, 1992. (Cited on page 415.)

[671] K. Jbilou and A. Messaoudi. Matrix recursive interpolation algorithm for block linear systems, direct methods. *Linear Algebra Appl.*, 294:137–154, 1999. (Cited on page 157.)

[672] K. C. Jea. *Generalized conjugate gradient acceleration of iterative methods*. PhD thesis, Center for Numerical Analysis, University of Texas at Austin, 1982. Report CNA-176. (Cited on pages 329 and 360.)

[673] K. C. Jea and D. M. Young. On the simplification of generalized conjugate-gradient methods for nonsymmetrizable linear systems. *Linear Algebra Appl.*, 52:399–417, 1983. (Cited on pages 329 and 360.)

[674] C.-P. Jeannerod. Further remarks on Kahan summation with decreasing ordering. HAL-03475741, 2021. (Cited on page 30.)

[675] C.-P. Jeannerod and S. M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. Appl.*, 34(2):338–344, 2013. (Cited on pages 43 and 44.)

[676] C.-P. Jeannerod and S. M. Rump. On relative errors of floating-point operations: optimal bounds and applications. *Math. Comput.*, 87(310):803–819, 2018. (Cited on pages 29, 42, 43, 44, and 45.)

[677] A. Jennings. Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method. *IMA J. Appl. Math.*, 20(1):61–72, 1977. (Cited on page 303.)

[678] A. Jennings. *Matrix Computations for Engineers and Scientists*. John Wiley, 1977. (Cited on pages 194 and 224.)

[679] A. Jennings and G. M. Malik. Partial elimination. *IMA J. Appl. Math.*, 20(3):307–316, 1977. (Cited on page 370.)

[680] J. A. G. Jess and H. G. M. Kees. A data structure for parallel LU decomposition. *IEEE Trans. Comput.*, C-31(3):231–239, 1982. (Cited on page 220.)

[681] Y. Jia, P. Luszczek, and J. J. Dongarra. Multi-GPU implementation of LU factorization. *Procedia Comput. Sci.*, 9:106–115, 2012. (Cited on page 144.)

[682] P. Jiránek and M. Rozložník. Adaptive version of Simpler GMRES. *Numer. Algorithms*, 53(1):93–112, 2010. (Cited on page 327.)

[683] P. Jiránek, M. Rozložník, and M. H. Gutknecht. How to make simpler GMRES and GCR more stable. *SIAM J. Matrix Anal. Appl.*, 30(4):1483–1499, 2008. (Cited on page 327.)

[684] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditioners for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20(2):362–376, 1983. (Cited on pages 382, 383, and 420.)

[685] L. Johnsson. Highly concurrent algorithms for solving linear systems of equations. Technical Report 5079-TR-83, California Institute of Technology, 1983. (Cited on pages 285 and 304.)

[686] L. Johnsson. Highly concurrent algorithms for solving linear systems of equations. In G. Birkhoff and A. Schoenstadt, editors, *Elliptic Problem Solvers, II*, pages 105–126. Academic Press, 1984. (Monterey, Calif., 1983). (Cited on pages 285 and 304.)

[687] W. Jordan. *Handbuch der Vermessungskunde, vol. 1*. Metzler, 1888. (Cited on pages 146 and 182.)

[688] W. Joubert. On the convergence behavior of the restarted GMRES algorithm for solving nonsymmetric linear systems. *Numer. Linear Algebra Appl.*, 1(5):427–447, 1994. (Cited on pages 323 and 324.)

[689] W. Kahan. Further remarks on reducing truncation errors. *Communications of the ACM*, 8(1):40, 1965. (Cited on page 29.)

[690] W. Kahan. A survey of error analysis. In *Proceedings IFIP Congress, Ljubljana, Information Processing 71*, pages 1214–1239. North-Holland, 1971. (Cited on page 29.)

[691] I. E. Kaporin. High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition. *Numer. Linear Algebra Appl.*, 5(6):483–509, 1998. (Cited on page 417.)

[692] M. Kaps and M. Schlegl. A short proof for the existence of the WZ factorisation. *Parallel Comput.*, 4(2):229–232, 1987. (Cited on page 167.)

[693] G. Karypis and V. Kumar. A high performance sparse Cholesky factorization algorithm for scalable parallel computers. In *Proceedings Frontiers' 95. The Fifth Symposium on the Frontiers of Massively Parallel Computation*. IEEE, 1995. (Cited on page 221.)

[694] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998. (Cited on page 402.)

[695] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. Parallel Dist. Comput.*, 48(1):71–95, 1998. (Cited on page 402.)

[696] R. Kehl, R. Nabben, and D. B. Szyld. Adaptive multilevel Krylov methods. *Electron. Trans. Numer. Anal.*, 51:512–528, 2017. (Cited on page 415.)

[697] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.*, 49(2):291–307, 1970. (Cited on pages 203 and 209.)

[698] D. S. Kershaw. The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.*, 26(1):43–65, 1978. (Cited on page 418.)

[699] D. S. Kershaw. On the problem of unstable pivots in the incomplete LU conjugate gradient method. *J. Comp. Phys.*, 38:114–123, 1980. (Cited on page 418.)

[700] A. Khabou, J. W. Demmel, L. Grigori, and M. Gu. LU factorization with panel rank revealing pivoting and its communication avoiding version. *SIAM J. Matrix Anal. Appl.*, 34(3):1401–1429, 2013. (Cited on page 145.)

[701] A. Kiełbasiński. Iterative refinement for linear systems in variable-precision arithmetic. *BIT Numerical Mathematics*, 21(1):97–103, 1981. (Cited on page 129.)

[702] S. K. Kim and A. T. Chronopoulos. An efficient nonsymmetric Lanczos method on parallel vector computers. *J. Comput. Appl. Math.*, 42(3):357–374, 1992. (Cited on page 338.)

[703] T. D. Kimura. Gauss-Jordan elimination by VLSI mesh-connected processors. Technical Report WUCS-79-3, Washington University St. Louis, 1979. (Cited on page 150.)

[704] I. P. King. An automatic reordering scheme for simultaneous equations derived from network analysis. *Internat. J. Numer. Methods Engrg.*, 2:523–533, 1970. (Cited on page 199.)

[705] N. G. Kingsbury and P. J. W. Rayner. Digital filtering using logarithmic arithmetic. *Electronics Letters*, 7(2):56–58, 1971. (Cited on page 17.)

[706] A. Klein. A generalized Kahan-Babuška-Summation-Algorithm. *Computing*, 76(3-4):279–293, 2006. (Cited on page 29.)

[707] P. A. Knight and D. Ruiz. A fast algorithm for matrix balancing. *IMA J. Numer. Anal.*, 33(3):1029–1047, 2013. (Cited on page 128.)

[708] P. A. Knight, D. Ruiz, and B. Uçar. A symmetry preserving algorithm for matrix scaling. *SIAM J. Matrix Anal. Appl.*, 35(3):931–955, 2014. (Cited on page 128.)

[709] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms, volume 2*. Addison-Wesley, 1969. (Cited on page 30.)

[710] L. Yu. Kolotilina and A. Yu Yeremin. Factorized sparse approximate inverse preconditionings I. Theory. *SIAM J. Matrix Anal. Appl.*, 14(1):45–58, 1993. (Cited on pages 380 and 420.)

[711] J. Kopal, M. Rozložník, and M. Tůma. Factorized approximate inverses with adaptive dropping. *SIAM J. Sci. Comput.*, 38(3):A1807–A1820, 2016. (Cited on page 382.)

[712] J. Kopal, M. Rozložník, and M. Tůma. A note on adaptivity in factorized approximate inverse preconditioning. *An. St. Univ. Ovidius Constanta - Seria Matematica*, 28(2):149–159, 2020. (Cited on pages 382 and 417.)

[713] B. Krasnopolsky. The reordered BiCGStab method for distributed memory computer systems. *Procedia Comput. Sci.*, 1:213–218, 2012. (Cited on page 343.)

[714] G. Krawezik and G. Poole. Accelerating the ANSYS direct sparse solver with GPUs. Proc. Symposium on Application Accelerators in High Performance Computing (SAAHPC), NCSA, Urbana-Champaign, IL., 2009. (Cited on page 222.)

[715] G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT Numerical Mathematics*, 37(3):559–590, 1997. (Cited on pages 200 and 202.)

[716] J. Kurzak, M. Gates, A. Charara, A. YarKhan, I. Yamazaki, and J. J. Dongarra. Linear systems solvers for distributed-memory machines with GPU accelerators. In *Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing*, pages 495–506. Springer, 2019. (Cited on page 145.)

[717] Y. Kuznetsov and A. Prokopenko. A new multilevel algebraic preconditioner for the diffusion equation in heterogeneous media. *Numer. Linear Algebra Appl.*, 17:759–769, 2010. (Cited on page 415.)

[718] X. Lacoste. *Scheduling and memory optimizations for sparse direct solver on multi-core/multi-GPU duster systems*. PhD thesis, Université de Bordeaux, France, 2015. (Cited on page 222.)

[719] C.-H. Lai. On Purcell's method and the Gauss-Jordan elimination method. *Int. J. Math. Educ. Sci. Technol.*, 25:759–778, 1994. (Cited on page 157.)

[720] C.-H. Lai. An extension of Purcell's vector method with applications to panel element equations. *Computers Math. Applic.*, 33(7):101–114, 1997. (Cited on pages 155 and 157.)

[721] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand.*, 45:255–282, 1950. (Cited on pages 302, 334, and 360.)

[722] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand.*, 49:33–53, 1952. (Cited on pages 302, 334, and 360.)

[723] P. Laurent-Gengoux and D. Trystram. A new presentation of the conjugate direction algorithm. *J. Comput. Appl. Math.*, 32:417–422, 1990. (Cited on pages 153 and 160.)

[724] D. P. Laurie. Anti-Gaussian quadrature formulas. *Math. Comput.*, 65(214):739–747, 1996. (Cited on page 304.)

[725] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, Philadelphia, 1995. First edition, Prentice-Hall (1974). (Cited on page 47.)

[726] C. L. Lawson, R. J. Hanson, F. T. Krogh, and D. R. Kincaid. Algorithm 539: Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft. (TOMS)*, 5(3):308–325, 1979. (Cited on page 130.)

[727] P. Le Tallec. Neumann-Neumann domain decomposition algorithms for solving 2D elliptic problems with nonmatching grids. *East-West J. Numer. Math.*, 1(2):129–146, 1993. (Cited on page 399.)

[728] V. I. Lebedev and S. A. Finogenov. The order of choice of the iteration parameters in the cyclic Chebyshev iteration method. *Zh. Vychisl. Mat. Mat. Fiz.*, 11(2):425–438, 1971. English translation: USSR Comput. Math. Math. Phys. 11 (2), (1971), pp. 155-170. (Cited on page 248.)

[729] R. B. Lehoucq. *Analysis and implementation of an implicitly restarted Arnoldi iteration*. PhD thesis, Rice University, Houston, Tx, USA, 1995. (Cited on page 324.)

[730] R. B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl.*, 17(4):789–821, 1996. (Cited on page 324.)

[731] R. J. LeVeque. *Finit Difference Methods for Ordinary and Partial Differential Equations*. SIAM, 2007. (Cited on page 78.)

[732] R. Levy. Restructuring the structural stiffness matrix to improve computational efficiency. *Jet Propulsion Laboratory Quarterly Technical Review*, 1:91–70, 1971. (Cited on page 199.)

[733] J. G. Lewis. Algorithm 582: The Gibbs-Poole-Stockmeyer and Gibbs-King algorithms for reordering sparse matrices. *ACM Trans. Math. Soft. (TOMS)*, 8(2):190–194, 1982. (Cited on page 199.)

[734] J. G. Lewis. Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms. *ACM Trans. Math. Soft. (TOMS)*, 8(2):180–189, 1982. (Cited on page 199.)

[735] J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Sci. Statist. Comput.*, 10(6):1146–1173, 1989. (Cited on page 220.)

[736] G. Li and T. F. Coleman. A parallel triangular solver for a distributed-memory multiprocessor. *SIAM J. Sci. Statist. Comput.*, 9(3):485–502, 1988. (Cited on pages 139 and 140.)

[737] G. Li and T. F. Coleman. A new method for solving triangular systems on distributed memory message passing multiprocessors. *SIAM J. Sci. Statist. Comput.*, 10(2):382–396, 1989. (Cited on pages 139 and 140.)

[738] N. Li, Y. Saad, and E. Chow. Crout versions of ILU for general sparse matrices. *SIAM J. Sci. Comput.*, 25(2):716–728, 2003. (Cited on page 368.)

[739] R. Li and Y. Saad. Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM J. Sci. Comput.*, 35(4):A2069–A2095, 2013. (Cited on page 416.)

[740] X. S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Soft. (TOMS)*, 31(3):302–325, 2005. (Cited on page 219.)

[741] X. S. Li. Evaluation of SuperLU on multicore architectures. *J. Phys.*, 125(1):012079, 2008. (Cited on page 219.)

[742] X. S. Li, P. Lin, Y. Liu, and P. Sao. Newly released capabilities in the distributed-memory SuperLU sparse direct solver. *ACM Trans. Math. Soft. (TOMS)*, 49(1):1–20, 2023. (Cited on pages 219 and 222.)

[743] Y. Li. An explicit construction of Gauss-Jordan elimination matrix. ArXiv preprint, arXiv:0907.5038, 2009. (Cited on page 150.)

[744] Z. Li, Y. Saad, and M. Sosonkina. pARMS: A parallel version of the algebraic recursive multilevel solver. *Numer. Linear Algebra Appl.*, 10(5-6):485–509, 2003. (Cited on pages 415 and 421.)

[745] J. Liesen, M. Rozložník, and Z. Strakoš. Least squares residuals and minimal residual methods. *SIAM J. Sci. Comput.*, 23(5):1503–1525, 2002. (Cited on page 326.)

[746] J. Liesen and Z. Strakoš. GMRES convergence analysis for a convection-diffusion model problem. *SIAM J. Sci. Comput.*, 26(6):1989–2009, 2005. (Cited on page 320.)

[747] J. Liesen and Z. Strakoš. *Krylov Subspace Methods, Principles and Analysis*. Oxford University Press, 2013. (Cited on page 334.)

[748] J. Liesen and P. Tichý. The worst-case GMRES for normal matrices. *BIT Numerical Mathematics*, 44:79–98, 2004. (Cited on page 321.)

[749] N. Lindquist, M. Gates, P. Luszczek, and J. J. Dongarra. Threshold pivoting for dense LU factorization. In *ScalAH22: 13th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Heterogeneous Systems*, pages ?–? IEEE, 2022. (Cited on pages 96 and 145.)

[750] N. Lindquist, P. Luszczek, and J. J. Dongarra. Replacing pivoting in distributed Gaussian elimination with randomized techniques. In *IEEE/ACM 11th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 35–43. IEEE, 2020. (Cited on page 96.)

[751] P.-L. Lions. On the Schwarz alternating method. I. In R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 1–42. SIAM, 1988. (Cited on pages 390 and 391.)

[752] P.-L. Lions. On the Schwarz alternating method. II: Stochastic interpretation and orders properties. In T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, editors, *Proceedings of the Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 47–70. SIAM, 1989. (Cited on pages 390, 391, and 421.)

[753] P.-L. Lions. On the Schwarz alternating method. III: A variant for nonoverlapping subdomains. In T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 202–223. SIAM, 1989. (Cited on pages 390, 394, and 421.)

[754] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16(2):346–358, 1979. (Cited on page 208.)

[755] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. (Cited on page 208.)

[756] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Soft. (TOMS)*, 11:141–153, 1985. (Cited on pages 204 and 220.)

[757] J. W. H. Liu. A compact row storage scheme for Cholesky factors using elimination trees. *ACM Trans. Math. Soft. (TOMS)*, 12(2):127–148, 1986. (Cited on pages 191, 193, and 224.)

[758] J. W. H. Liu. Equivalent sparse matrix reordering by elimination tree rotations. *SIAM J. Sci. Statist. Comput.*, 9(3):424–444, 1988. (Cited on pages 220 and 221.)

[759] J. W. H. Liu. The minimum degree ordering with constraints. *SIAM J. Sci. Statist. Comput.*, 10(6):1136–1145, 1989. (Cited on pages 205 and 220.)

[760] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990. (Cited on pages 191, 193, and 224.)

[761] J. W. H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Rev.*, 34(1):82–109, 1992. (Cited on pages 212, 215, and 224.)

[762] J. W. H. Liu and A. Mirzaian. A linear reordering algorithm for parallel pivoting of chordal graphs. *SIAM J. Alg. Disc. Meth.*, 2(1):100–107, 1989. (Cited on page 220.)

[763] J. W. H. Liu, E. G. Ng, and B. W. Peyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(1):242–252, 1993. (Cited on pages 211 and 224.)

[764] O. E. Livne and G. H. Golub. Scaling by binormalization. *Numer. Algorithms*, 35(1):97–120, 2004. (Cited on page 128.)

[765] F. Lopez and T. Mary. Mixed precision LU factorization on GPU tensor cores: Reducing data movement and memory footprint. Technical Report MIMS EPrint 2020.20, University of Manchester, 2020. To be published in The International Journal of High Performance Computing Applications. (Cited on page 146.)

[766] L. D. J. C. Loyens and R. B. Bisseling. The formal construction of a parallel triangular system solver. In J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction*, pages 325–334, Berlin, Heidelberg, 1989. Springer. (Cited on page 140.)

[767] R. F. Lucas, G. Wagenbreth, D. M. Davis, and R. Grimes. Multifrontal computations on GPUs and their multi-core hosts. In *High Performance Computing for Computational Science-VECPAR 2010: 9th International Conference*, pages 71–82. Springer, 2011. (Cited on page 222.)

[768] R. Luce and E. G. Ng. On the minimum FLOPs problem in the sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 35(1):1–21, 2014. (Cited on page 191.)

[769] P. Luszczek, J. Kurzak, and J. J. Dongarra. Looking back at dense linear algebra software. *J. Parallel Dist. Comput.*, 74(7):2548–2560, 2014. (Cited on pages 133 and 145.)

[770] S. MacLachlan, D. Osei-Kuffuor, and Y. Saad. Modification and compensation strategies for threshold-based incomplete factorizations. *SIAM J. Sci. Comput.*, 14(1):A48–A75, 2012. (Cited on page 373.)

[771] N. Mahdavi-Amiri and E. Golpar-Raboky. Extended rank reduction formulas containing Wedderburn and Abaffy-Broyden-Spedicato rank reducing processes. *Linear Algebra Appl.*, 439(11):3318–3331, 2013. (Cited on page 163.)

[772] J. Màlek and Z. Strakoš. *Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs*. SIAM, Philadelphia, 2015. (Cited on page 417.)

[773] J. Mandel. On block diagonal and Schur complement preconditioning. *Numer. Math.*, 58:79–93, 1990. (Cited on page 417.)

[774] J. Mandel. Balancing domain decomposition. *Comm. Numer. Meth. Eng.*, 9(3):233–241, 1993. (Cited on page 407.)

[775] J. Mandel and M. Brezina. Balancing domain decomposition for problems with large jumps in coefficients. *Math. Comput.*, 65(216):1387–1401, 1996. (Cited on page 407.)

[776] J. Mandel and C. R. Dohrmann. Convergence of a balancing domain decomposition by constraints and energy minimization. *Numer. Linear Algebra Appl.*, 10:639–659, 2003. (Cited on page 407.)

[777] J. Mandel and B. Sousedík. BDDC and FETI-DP under minimalist assumptions. *Computing*, 81(4):269–280, 2007. (Cited on page 407.)

[778] J. Mandel and R. Tezaur. Convergence of a substructuring method with lagrange multipliers. *Numer. Math.*, 73:473–487, 1996. (Cited on page 406.)

[779] T. A. Manteuffel. The Tchebychev iteration for nonsymmetric linear systems. *Numer. Math.*, 28(3):307–327, 1977. (Cited on pages 251, 388, and 420.)

[780] T. A. Manteuffel. Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration. *Numer. Math.*, 31(2):183–208, 1978. (Cited on pages 251, 388, and 420.)

[781] T. A. Manteuffel. The shifted incomplete Cholesky factorization. In I. S. Duff and G. W. Stewart, editors, *Sparse Matrix Proceedings 1978*, pages 41–61. SIAM, 1979. (Cited on page 418.)

[782] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comput.*, 34(150):473–497, 1980. (Cited on pages 370 and 418.)

[783] T. A. Manteuffel and J. S. Otto. Optimal equivalent preconditioners. *SIAM J. Numer. Anal.*, 30(3):790–812, 1993. (Cited on page 417.)

[784] G. I. Marchuk and Y. A. Kuznetsov. On optimal iteration processes. *Soviet Math. Dokl.*, 9(4):1041–1045, 1968. (Cited on page 358.)

[785] G. I. Marchuk and Y. A. Kuznetsov. Méthodes itératives et fonctionnelles quadratiques. In J. L. Lions and G. I. Marchuk, editors, *Sur les Méthodes Numériques en Sciences Physiques et Économiques*, pages 1–117. Dunod, 1974. (Cited on page 358.)

[786] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957. (Cited on page 217.)

[787] R. S. Martin, G. Peters, and J. H. Wilkinson. Iterative refinement of the solution of a positive definite system of equations. *Numer. Math.*, 8:203–216, 1966. (Cited on page 129.)

[788] T. Mary. *Block low-rank multifrontal solvers: Complexity, performance, and scalability*. PhD thesis, Université Toulouse 3 Paul Sabatier, France, 2017. (Cited on page 223.)

[789] S. Massei, L. Robol, and D. Kressner. hm-toolbox: Matlab software for HODLR and HSS matrices. *SIAM J. Sci. Comput.*, 42(2):C43–C68, 2020. (Cited on page 223.)

[790] T. P. Mathew, P. L. Polyakov, G. Russo, and J. Wang. Domain decomposition operator splittings for the solution of parabolic equations. *SIAM J. Sci. Comput.*, 19(3):912–932, 1998. (Cited on page 244.)

[791] A. M. Matsokin and S. V. Nepomnyaschikh. A Schwarz alternating method in a subspace. *Soviet Math. (Iz. VUZ)*, 29:78–84, 1985. (Cited on page 393.)

[792] C. McCarthy and G. Strang. Optimal conditioning of matrices. *SIAM J. Numer. Anal.*, 10(2):370–388, 1973. (Cited on page 128.)

[793] S. F. McCormick Ed. *Multigrid Methods: Theory, Applications, and Supercomputing*. Marcel Dekker, 1988. Lecture Notes in Pure and Applied Mathematics 110. (Cited on page 421.)

[794] U. Meier Yang. On long-range interpolation operators for aggressive coarsening. *Numer. Linear Algebra Appl.*, 17(2-3):453–472, 2010. (Cited on page 414.)

[795] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, 31(137):148–162, 1977. (Cited on pages 368 and 418.)

[796] N. Melab, E.-G. Talbi, and S. Petiton. A parallel adaptive Gauss-Jordan algorithm. *J. Supercomput.*, 17:167–185, 2000. (Cited on page 150.)

[797] R. G. Melhem. Parallel Gauss-Jordan elimination for the solution of dense linear systems. *Parallel Comput.*, 4(3):339–343, 1987. (Cited on page 150.)

[798] A. Messaoudi. Recursive interpolation algorithm: A formalism for solving systems of linear equations-I. Direct methods. *J. Comput. Appl. Math.*, 76(1-2):13–30, 1996. (Cited on page 157.)

[799] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984. (Cited on page 417.)

[800] G. Meurant. Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2. Technical Report LBL-18023, Lawrence Berkeley Laboratory, 1984. (Cited on page 304.)

[801] G. Meurant. Multitasking the conjugate gradient on the CRAY X-MP/48. Technical Report NA-85-33, Computer Science Department, Stanford University, 1985. (Cited on page 304.)

[802] G. Meurant. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Comput.*, 5:267–280, 1987. (Cited on pages 285, 298, and 304.)

[803] G. Meurant. Domain decomposition preconditioners for the conjugate gradient method. *Calcolo*, 25(1):103–119, 1988. (Cited on page 417.)

[804] G. Meurant. Domain decomposition vs block preconditioning. In R. Glowinski, G. H. Golub, G. Meurant, and J. Périaux, editors, *Domain Decomposition Methods for Partial Differential Equations*, pages 231–249. SIAM, 1988. (Cited on page 417.)

[805] G. Meurant. Incomplete domain decomposition preconditioners for the conjugate gradient method. In G. H. Rodrigue, editor, *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 100–106. SIAM, 1988. (Cited on page 417.)

[806] G. Meurant. Incomplete domain decomposition preconditioners for nonsymmetric problems. In T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, editors, *Second International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 219–225. SIAM, 1989. (Cited on page 417.)

[807] G. Meurant. The conjugate gradient method on vector and parallel supercomputers. In W. L. Hogarth and B. J. Noye, editors, *Computational Techniques and Applications*. Hemisphere Publishing Corp., 1990. (Cited on page 304.)

[808] G. Meurant. A domain decomposition method for parabolic problems. *Appl. Numer. Math.*, 8:427–441, 1991. (Cited on page 417.)

[809] G. Meurant. A review on the inverse of symmetric tridiagonal and block tridiagonal matrices. *SIAM J. Matrix Anal. Appl.*, 13(3):707–728, 1992. (Cited on pages 70, 79, 111, 116, and 367.)

[810] G. Meurant. The computation of bounds for the norm of the error in the conjugate gradient algorithm. *Numer. Algorithms*, 16:77–87, 1997. (Cited on pages 283 and 304.)

[811] G. Meurant. *Computer Solution of Large Linear Systems*. North-Holland, Amsterdam, 1999. (Cited on pages 3, 304, 366, 398, and 402.)

[812] G. Meurant. Numerical experiments in computing bounds for the norm of the error in the preconditioned conjugate gradient algorithm. *Numer. Algorithms*, 22:353–365, 1999. (Cited on pages 283 and 304.)

[813] G. Meurant. Numerical experiments with algebraic multilevel preconditioners. *Electron. Trans. Numer. Anal.*, 12:1–65, 2001. (Cited on pages 382 and 414.)

[814] G. Meurant. A multilevel AINV preconditioner. *Numer. Algorithms*, 29:107–129, 2002. (Cited on pages 382, 413, and 414.)

[815] G. Meurant. Estimates of the ℓ_2 norm of the error in the conjugate gradient algorithm. *Numer. Algorithms*, 40(2):157–169, 2005. (Cited on pages 284 and 304.)

[816] G. Meurant. *The Lanczos and Conjugate Gradient Algorithms, from Theory to Finite Precision Computations*. SIAM, Philadelphia, 2006. (Cited on pages 265, 267, 277, 304, and 337.)

[817] G. Meurant. Estimates of the norm of the error in solving linear systems with FOM and GMRES. *SIAM J. Sci. Comput.*, 33(5):2686–2705, 2011. (Cited on page 322.)

[818] G. Meurant. The complete stagnation of GMRES for $n < 4$. *Electron. Trans. Numer. Anal.*, 39:75–101, 2012. (Cited on page 321.)

[819] G. Meurant. Necessary and sufficient conditions for GMRES complete and partial stagnation. *Appl. Numer. Math.*, 75:100–107, 2014. (Cited on pages 321 and 359.)

[820] G. Meurant. The coefficients of the FOM and GMRES residual polynomials. *SIAM J. Matrix Anal. Appl.*, 38(1):96–117, 2017. (Cited on page 320.)

[821] G. Meurant. The floatp toolbox. https://gerard-meurant.fr/, 2020. (Cited on page 32.)

[822] G. Meurant and J. Duintjer Tebbens. The role eigenvalues play in forming GMRES residual norms with non-normal matrices. *Numer. Algorithms*, 68(1):143–165, 2015. (Cited on page 359.)

[823] G. Meurant and J. Duintjer Tebbens. *Krylov Methods for Nonsymmetric Linear Systems*. Springer International Publishing, 2020. Springer Series in Computational Mathematics, Vol. 57. (Cited on pages 292, 307, 308, 309, 310, 318, 319, 320, 322, 323, 326, 327, 336, 345, 349, and 351.)

[824] G. Meurant, J. Papež, and P. Tichý. Accurate error estimation in CG. *Numer. Algorithms*, 88(3):1337–1359, 2021. (Cited on pages 284 and 304.)

[825] G. Meurant and Z. Strakoš. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica*, 15:471–542, 2006. (Cited on pages 277, 304, and 337.)

[826] G. Meurant and P. Tichý. On computing quadrature-based bounds for the A-norm of the error in conjugate gradients. *Numer. Algorithms*, 62(2):163–191, 2013. (Cited on pages 283 and 304.)

[827] G. Meurant and P. Tichý. Erratum to: On computing quadrature-based bounds for the A-norm of the error in conjugate gradients. *Numer. Algorithms*, 66(3):679–680, 2014. (Cited on pages 283 and 304.)

[828] G. Meurant and P. Tichý. Approximating the extreme Ritz values and upper bounds for the A-norm of the error in CG. *Numer. Algorithms*, 82(3):937–968, 2019. (Cited on pages 283 and 304.)

[829] G. Meurant and P. Tichý. The behavior of the Gauss-Radau upper bound of the error norm in CG. *Numer. Algorithms*, 94(2):847–876, 2023. (Cited on pages 284, 298, and 304.)

[830] G. Meurant and P. Tichý. *Error Norm Estimation in the Conjugate Gradient Algorithm.* to be published by SIAM, 2024. (Cited on pages 283 and 284.)

[831] P. D. Michailidis and K. G. Margaritis. Open multi processing (OpenMP) of Gauss-Jordan method for solving system of linear equations. In *11th International Conference on Computer and Information Technology*, pages 314–319. IEEE, 2011. (Cited on page 150.)

[832] G. Migliorati and A. Quarteroni. Multilevel Schwarz methods for elliptic partial differential equations. *Comput. Methods Appl. Mech. Engrg.*, 200(25-28):2282–2296, 2011. (Cited on page 416.)

[833] C. B. Moler. Iterative refinement in floating point. *J. ACM*, 14(2):316–321, 1967. (Cited on page 129.)

[834] C. B. Moler. Matrix computation on distributed memory multiprocessors. In M. T. Heath, editor, *Hypercube Multiprocessors 1986*, pages 181–195. SIAM, 1986. (Cited on page 139.)

[835] M. M. Monga Made and H. A. van der Vorst. ParIC: A family of parallel incomplete Cholesky preconditioners. In M. Bubak, J. Afsarmanesh, R. Williams, and B. Hertzberger, editors, *Proceedings of the 8th International Conference on High-Performance Computing and Networking*, pages 89–98, Berlin, Heidelberg, 2000. Springer. (Cited on page 417.)

[836] M. M. Monga Made and H. A. van der Vorst. Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap. *Numer. Linear Algebra Appl.*, 9(1):45–64, 2002. (Cited on page 417.)

[837] J. Moré. Global convergence of Newton Gauss Seidel methods. *SIAM J. Numer. Anal.*, 8:325–336, 1971. (Cited on page 61.)

[838] R. B. Morgan. A restarted GMRES method augmented with eigenvectors. *SIAM J. Matrix Anal. Appl.*, 16(4):1154–1171, 1995. (Cited on pages 324 and 360.)

[839] R. B. Morgan. Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Appl.*, 21(4):1112–1135, 2000. (Cited on pages 324 and 360.)

[840] R. B. Morgan. GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 24(1):20–37, 2002. (Cited on pages 324 and 360.)

[841] R. B. Morgan, Z. Yang, and B. Zhong. Pseudoeigenvector bases and deflated GMRES for highly nonnormal matrices. *Numer. Linear Algebra Appl.*, 23(6):1032–1045, 2016. (Cited on page 325.)

[842] K. Moriya and T. Nodera. Usage of the convergence test of the residual norm in the Tsuno-Nodera version of the GMRES algorithm. *ANZIAM J.*, 49:293–308, 2007. (Cited on page 324.)

[843] J. Morris. A successive approximation process for solving simultaneous linear equations. Technical Report A.R.C.R. & M., 1711, Aeronautical Research Council, 1936. (Cited on page 158.)

[844] R. Morris. Tapered floating point: A new floating-point representation. *IEEE Trans. Comput.*, 100(12):1578–1579, 1971. (Cited on page 18.)

[845] D. Moskovitz. The numerical solution of Laplace's and Poisson's equations. *Quart. Appl. Math.*, 2(2):148–163, 1944. (Cited on page 79.)

[846] J.-M. Muller, N. Brunie, F. de Dinechin, C. P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres. *Handbook of Floating-point Arithmetic*. Birkhäuser, 2018. (Cited on pages 3 and 78.)

[847] R. Nabben. Decay rates of the inverse of nonsymmetric tridiagonal and band matrices. *SIAM J. Matrix Anal. Appl.*, 20(3):820–837, 1999. (Cited on page 112.)

[848] R. Nabben and C. Vuik. A comparison of deflation and the balancing preconditioner. *SIAM J. Sci. Comput.*, 27(5):1742–1759, 2006. (Cited on page 417.)

[849] M. Nakhla, K. Singhal, and J. Vlach. An optimal pivoting order for the solution of sparse systems of equations. *IEEE Trans. Circuits and Systems*, 22(2):222–225, 1974. (Cited on page 205.)

[850] A. Napov and Y. Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM J. Sci. Comput.*, 34(2):A1079–1109, 2012. (Cited on page 413.)

[851] A. Neumaier and R. S. Varga. Exact convergence and divergence domains for the symmetric successive overrelaxation iterative (SSOR) method applied to H-matrices. *Linear Algebra Appl.*, 58:261–272, 1980. (Cited on page 238.)

[852] M. Neumann and R. S. Varga. On the sharpness of some upper bounds for the spectral radii of SOR iteration matrices. *Numer. Math.*, 35:69–79, 1980. (Cited on page 235.)

[853] O. Nevanlinna. *Convergence of Iterations for Linear Equations*. Birkhäuser Verlag, Basel, 1993. Lectures in Mathematics ETH Zürich. (Cited on page 316.)

[854] E. G. Ng and B. W. Peyton. A supernodal Cholesky factorization algorithm for shared memory multiprocessors. *SIAM J. Sci. Comput.*, 14(4):761–769, 1993. (Cited on page 224.)

[855] E. G. Ng and P. Raghavan. Performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(4):902–914, 1999. (Cited on page 205.)

[856] R. A. Nicolaides. On multiple grid and related techniques for solving discrete elliptic systems. *J. Comp. Phys.*, 19:418–431, 1975. (Cited on page 421.)

[857] R. A. Nicolaides. On multigrid convergence in the indefinite case. *Math. Comput.*, 32(144):1082–1086, 1978. (Cited on page 421.)

[858] R. A. Nicolaides. On some theoretical and practical aspects of multigrid methods. *Math. Comput.*, 33(147):933–952, 1979. (Cited on page 421.)

[859] Y. Notay. On the convergence rate of the conjugate gradients in presence of rounding errors. *Numer. Math.*, 65(1):301–317, 1993. (Cited on pages 278 and 303.)

[860] Y. Notay. Optimal V-cycle algebraic multilevel preconditioning. *Numer. Linear Algebra Appl.*, 5(5):441–459, 1998. (Cited on page 415.)

[861] Y. Notay. A multilevel block incomplete factorization preconditioning. *Appl. Numer. Math.*, 31(2):209–225, 1999. (Cited on pages 415 and 421.)

[862] Y. Notay. Optimal order preconditioning of finite difference matrices. *SIAM J. Sci. Comput.*, 21(6):1991–2007, 2000. (Cited on page 415.)

[863] Y. Notay. Robust parameter-free algebraic multilevel preconditioning. *Numer. Linear Algebra Appl.*, 9(6-7):409–428, 2002. (Cited on pages 415 and 421.)

[864] Y. Notay. Aggregation-based algebraic multilevel preconditioning. *SIAM J. Matrix Anal. Appl.*, 27(4):998–1018, 2006. (Cited on pages 413, 415, and 421.)

[865] Y. Notay. An aggregation-based algebraic multigrid method. *Electron. Trans. Numer. Anal.*, 37:123–147, 2010. (Cited on page 413.)

[866] B. Nour-Omid and B. N. Parlett. Element preconditioning using splitting techniques. *SIAM J. Sci. Statist. Comput.*, 6:761–771, 1985. (Cited on page 417.)

[867] P. Novati, M. Redivo-Zaglia, and M. R. Russo. Preconditioning linear systems via matrix function evaluation. *Appl. Numer. Math.*, 62(12):1804–1818, 2012. (Cited on page 417.)

[868] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numer. Math.*, 6(1):405–409, 1964. (Cited on page 127.)

[869] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005. (Cited on pages 30 and 36.)

[870] S. Oishi, T. Ogita, and S. M. Rump. Iterative refinement for ill-conditioned linear systems. *Japan J. Indust. Appl. Math.*, 26(2-3):465–476, 2009. (Cited on page 129.)

[871] E. Oktay and E. C. Carson. Mixed precision GMRES-based iterative refinement with recycling. ArXiv preprint arXiv:2201.09827, 2022. (Cited on page 129.)

[872] E. Oktay and E. C. Carson. Multistage mixed precision iterative refinement. *Numer. Linear Algebra Appl.*, 29(4):e2434, 2022. (Cited on page 129.)

[873] R. Oldenburger. Infinite powers of matrices and characteristic roots. *Duke Math. J.*, 6(2):357–361, 1940. (Cited on page 52.)

[874] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.*, 29:293–322, 1980. (Cited on page 303.)

[875] D. P. O'Leary. Yet another polynomial precondtioner for the conjugate gradient method. *Linear Algebra Appl.*, 154-156:377–388, 1991. (Cited on pages 417 and 420.)

[876] D. P. O'Leary. Conjugate gradients and related KMP algorithms: The beginnings. In L. Adams and J. L. Nazareth, editors, *Linear and Nonlinear Conjugate Gradient-related Methods*, pages 1–8, Philadelphia, 1996. SIAM. (Cited on page 302.)

[877] L. Oliker, X. Li, P. Husbands, and R. Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *SIAM Rev.*, 44(3):373–393, 2002. (Cited on page 377.)

[878] T. A. Oliphant. An extrapolation procedure for solving linear systems. *Quart. Appl. Math.*, 20(3):257–265, 1962. (Cited on page 418.)

[879] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996. (Cited on page 96.)

[880] M. E. Ong. Hierarchical basis preconditioners in three dimensions. *SIAM J. Sci. Comput.*, 18(2):479–498, 1997. (Cited on page 417.)

[881] G. Opfer and G. Schober. Richardson's iteration for nonsymmetric matrices. *Linear Algebra Appl.*, 58:343–361, 1984. (Cited on page 248.)

[882] T. C. Oppe and D. R. Kincaid. Parallel LU-factorization algorithms for dense matrices. In E. N. Houstis, T. S. Papatheodorou, and C. D. Polychronopoulos, editors, *Supercomputing: 1st International Conference*, pages 576–594, Berlin, Heidelberg, 1988. Springer. (Cited on pages 132 and 142.)

[883] W. Orchard-Hays. An efficient form of inverse for sparse matrices. In *Proceedings of the 1956 11th ACM National Meeting*, pages 154–157, New York, 1956. Association for Computing Machinery. (Cited on page 223.)

[884] J. M. Ortega and R. J. Plemmons. Extensions of the Ostrowski-Reich theorem for SOR iterations. *Linear Algebra Appl.*, 28, 1979. (Cited on page 63.)

[885] D. Osei-Kuffuor, R. Li, and Y. Saad. Matrix reordering using multilevel graph coarsening for ILU preconditioning. *SIAM J. Sci. Comput.*, 37(1):A391–A419, 2015. (Cited on page 377.)

[886] O. Østerby and Z. Zlatev. *Direct Methods for Sparse Matrices*. Springer, 1983. Lecture notes in computer science 157. (Cited on page 217.)

[887] A. M. Ostrowski. On the linear iteration procedures for symmetric matrices. *Rend. Mat. Appl.*, 14:140–163, 1954. (Cited on page 235.)

[888] A. M. Ostrowski. Uber normen von matrizen. *Math. Zeitschrift*, 63(1):2–18, 1955. (Cited on page 78.)

[889] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, University of London, UK, 1971. (Cited on pages 277, 303, and 337.)

[890] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Maths Applics*, 10:373–381, 1972. (Cited on pages 277 and 303.)

[891] C. C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *J. Inst. Maths Applics*, 18:341–349, 1976. (Cited on pages 277 and 303.)

[892] C. C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra Appl.*, 34:235–258, 1980. (Cited on pages 277 and 303.)

[893] C. C. Paige. A useful form of unitary matrix obtained from any sequence of unit 2-norm n-vectors. *SIAM J. Matrix Anal. Appl.*, 31(2):565–583, 2009. (Cited on page 278.)

[894] C. C. Paige. An augmented stability result for the Lanczos Hermitian matrix tridiagonalization process. *SIAM J. Matrix Anal. Appl.*, 31(5):2347–2359, 2010. (Cited on pages 277, 279, and 337.)

[895] C. C. Paige. Accuracy of the Lanczos process for the eigenproblem and solution of equations. *SIAM J. Matrix Anal. Appl.*, 40(4):1371–1398, 2019. (Cited on pages 277 and 279.)

[896] C. C. Paige, I. Panayotov, and J. P. M. Zemke. An augmented analysis of the perturbed two-sided Lanczos tridiagonalization process. *Linear Algebra Appl.*, 447:119–132, 2014. (Cited on pages 337 and 338.)

[897] C. C. Paige, M. Rozložník, and Z. Strakoš. Modified Gram-Schmidt (MGS), least squares and backward stability of MGS-GMRES. *SIAM J. Matrix Anal. Appl.*, 28(1):264–284, 2006. (Cited on page 359.)

[898] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations and least squares problems. Technical Report STAN-CS-73-399, Computer Science Dept., Stanford University, 1973. (Cited on pages 287, 288, 289, and 304.)

[899] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975. (Cited on pages 287, 291, and 304.)

[900] C. C. Paige and W. Wülling. Properties of a unitary matrix obtained from a sequence of normalized vectors. *SIAM J. Matrix Anal. Appl.*, 35(2):526–545, 2014. (Cited on page 278.)

[901] V. Y. Pan. Improved parallel solution of a triangular linear system. *Computers Math. Applic.*, 27(11):41–43, 1994. (Cited on page 139.)

[902] V. Y. Pan, G. Qian, and A.-L. Zheng. Randomized preprocessing versus pivoting. *Linear Algebra Appl.*, 438(4):1883–1899, 2013. (Cited on page 96.)

[903] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976. (Cited on page 194.)

[904] D. S. Parker. Random butterfly transformations with applications in computational linear algebra. Technical Report CSD-950023, UCLA Computer Science Department, 1995. (Cited on page 96.)

[905] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, 1980. Reprinted by SIAM (1998). (Cited on page 202.)

[906] S. V. Parter. The use of linear graphs in Gauss elimination. *SIAM Rev.*, 3(2):119–130, 1961. (Cited on pages 78, 188, 192, and 223.)

[907] G. H. Paulino, I. F. Menezes, M. Gattass, and S. Mukherjee. Node and element resequencing using the Laplacian of a finite element graph: part I- general concepts and algorithm. *Internat. J. Numer. Methods Engrg.*, 37:1511–1530, 1994. (Cited on page 202.)

[908] D. W. Peaceman and H. H. Rachford Jr. The numerical solution of parabolic and elliptic differential equations. *J. Soc. Ind. Appl. Math.*, 3(1):28–41, 1955. (Cited on page 240.)

[909] C. Pechstein. *Finite and Boundary Element Tearing and Interconnecting Solvers for Multiscale Problems*. Springer, 2013. (Cited on page 408.)

[910] F. Pellegrini. Scotch and libScotch 5.1 User's guide. Technical Report hal-00410327, INRIA, 2008. (Cited on page 220.)

[911] F. Pellegrini, J. Roman, and P. R. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency-Pract. Exp.*, 12(2-3):68–84, 2000. (Cited on page 205.)

[912] J. M. Peña. Pivoting strategies leading to diagonal dominance by rows. *Numer. Math.*, 81(2):293–304, 1998. (Cited on page 96.)

[913] D. V. Perevozkin and G. A. Omarova. Preconditioning methods based on spanning tree algorithms. *Journal of Physics*, 1715(1):1–5, 2021. (Cited on page 417.)

[914] O. Perlot. *Préconditionnements de systèmes linéaires sur machines massivement parallèles CM-2 et CM-5*. PhD thesis, Université Paris VI, 1995. (Cited on page 384.)

[915] G. Peters and J. H. Wilkinson. On the stability of Gauss-Jordan elimination with pivoting. *Communications of the ACM*, 18(1):20–24, 1975. (Cited on page 148.)

[916] B. W. Peyton, A. Pothen, and X. Yuan. Partitioning a chordal graph into transitive subgraphs for parallel sparse triangular solution. *Linear Algebra Appl.*, 192:329–354, 1993. (Cited on page 186.)

[917] G. Pichon, E. Darve, M. Faverge, P. Ramet, and J. Roman. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. *J. Comput. Sci.*, 27:255–270, 2018. (Cited on page 223.)

[918] D. Pierce, Y. Hung, C. C. Liu, Y. H. Tsai, W. Wang, and D. Yu. Sparse multifrontal performance gains via NVIDIA GPU. Workshop on GPU Supercomputing, National Taiwan University, 2009. (Cited on page 222.)

[919] T. Pietrzykowski. Projection method. Zaktadu Aparatow Matematycznych Polskiej Akad. Nauk, Report A 8, 1960. (Cited on page 156.)

[920] S. Pissanetsky. *Sparse Matrix Technology*. Academic Press, 1984. (Cited on page 185.)

[921] E. L. Poole and J. M. Ortega. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24(6):1394–1418, 1987. (Cited on page 377.)

[922] G. Poole and L. Neal. A geometric analysis of Gaussian elimination, I. *Linear Algebra Appl.*, 149:249–272, 1991. (Cited on page 95.)

[923] G. Poole and L. Neal. Gaussian elimination: when is scaling beneficial? *Linear Algebra Appl.*, 162:309–324, 1992. (Cited on pages 95, 96, and 128.)

[924] C. Popa, T. Preclik, H. Köstler, and U. Rüde. On Kaczmarz's projection iteration as a direct solver for linear least squares problems. *Linear Algebra Appl.*, 436(2):389–404, 2012. (Cited on page 158.)

[925] A. Pothen and F. L. Alvarado. A fast reordering algorithm for parallel sparse triangular solution. *SIAM J. Sci. Statist. Comput.*, 13(2):645–653, 1992. (Cited on page 186.)

[926] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990. (Cited on page 209.)

[927] A. Pothen and S. Toledo. Elimination structures in scientific computing. In D. Mehta and S. Sahni, editors, *Handbook on Data Structures and Applications*, chapter 59. Chapman and Hall, CRC Press, 2004. (Cited on page 193.)

[928] H. Pouransari, P. Coulier, and E. Darve. Fast hierarchical solvers for sparse matrices. ArXiv preprint, arXiv:1510.07363, 2015. (Cited on page 223.)

[929] D. M. Priest. Algorithms for arbitrary precision floating point arithmetic. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 132–144. IEEE, 1991. (Cited on page 29.)

[930] D. M. Priest. *On properties of floating point arihmetics: numerical stability and the cost of accurate computations*. PhD thesis, University of California, Berkeley, 1992. (Cited on page 29.)

[931] E. E. Prudencio and X.-C. Cai. Robust multilevel restricted Schwarz preconditioners and applications. In D. E. Keyes and O. B. Widlund, editors, *Domain Decomposition Methods in Science and Engineering XVI*, pages 155–162. Springer, 2007. (Cited on page 416.)

[932] I. Pultarová and M. Ladecký. Two-sided guaranteed bounds to individual eigenvalues of preconditioned finite element and finite difference problems. *Numer. Linear Algebra Appl.*, 28:e2382, 2021. (Cited on page 417.)

[933] E. W. Purcell. The vector method of solving simultaneous linear equations. *J. Math. Phys.*, 32(1-4):180–183, 1953. (Cited on page 154.)

[934] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, 1999. (Cited on pages 408 and 421.)

[935] P. Raghavan and K. Teranishi. Parallel hybrid preconditioning: Incomplete factorization with selective sparse approximate inversion. *SIAM J. Sci. Comput.*, 32(3):1323–1345, 2010. (Cited on page 417.)

[936] S. C. S. Rao. Existence and uniqueness of WZ factorization. *Parallel Comput.*, 23(8):1129–1139, 1997. (Cited on page 167.)

[937] E. Reich. On the convergence of the classical iterative method of solving linear simultaneous equations. *Ann. Math. Stat.*, 20(3):448–451, 1949. (Cited on page 235.)

[938] J. K. Reid. On the method of conjugate gradients for the solution of large sparse systems of linear equations. In J. K. Reid, editor, *Proc. Conference on Large Sparse Sets of Linear Equations*, New York, 1971. Academic Press. (Cited on page 303.)

[939] A. Rémy. *Solving dense linear systems on accelerated multicore architectures*. PhD thesis, Université Paris-Sud, 2015. (Cited on page 96.)

[940] O. Rendel, A. Rizvanolli, and J. P. M. Zemke. IDR: A new generation of Krylov subspace methods? *Linear Algebra Appl.*, 439(4):1040–1061, 2013. (Cited on pages 351 and 361.)

[941] S. C. Rennich, D. Stosic, and T. A. Davis. Accelerating sparse Cholesky factorization on GPUs. In *Proc. IA3 Workshop on Irregular Applications: Architectures and Algorithms*, pages 9–16. IEEE, 2014. (Cited on page 222.)

[942] S. C. Rennich, D. Stosic, and T. A. Davis. Accelerating sparse Cholesky factorization on GPUs. *Parallel Comput.*, 59:140–150, 2016. (Cited on page 222.)

[943] J. R. Rice. Experiments on Gram-Schmidt orthogonalization. *Math. Comput.*, 20(94):325–328, 1966. (Cited on page 47.)

[944] L. F. Richardson. The approximate arithmetical solution by finite difference of physical problems involving differential equations, with an application to the stress in a masonry dam. *Philos. Trans. R. Soc. Lond.*, 210:307–357, 1910. ser A. (Cited on page 245.)

[945] J. L. Rigal and J. Gaches. On the compatibility of a given solution with the data of a linear system. *J. ACM*, 14(3):543–548, 1967. (Cited on page 126.)

[946] Y. Robert. Regular incomplete factorizations of real positive definite matrices. *Linear Algebra Appl.*, 48:105–117, 1982. (Cited on pages 370 and 418.)

[947] G. H. Rodrigue and D. Wolitzer. Preconditioning by incomplete block cyclic reduction. *Math. Comput.*, 42(166):549–565, 1984. (Cited on page 417.)

[948] S. Röllin and W. Fichtner. Improving the accuracy of GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 30(1):232–245, 2007. (Cited on page 325.)

[949] J. Roman. Calcul de complexité relatifs à une méthode de dissection emboîtée. *Numer. Math.*, 47:175–190, 1985. (Cited on pages 208 and 209.)

[950] C. H. Romine and J. M. Ortega. Parallel solution of triangular systems of equations. *Parallel Comput.*, 6:109–114, 1988. (Cited on page 139.)

[951] D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970. (Cited on pages 78 and 188.)

[952] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM J. Appl. Math.*, 34(1):176–197, 1978. (Cited on page 216.)

[953] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. (Cited on page 192.)

[954] D. J. Rose, G. G. Whitten, A. H. Sherman, and R. E. Tarjan. Algorithms and software for in-core factorization of sparse symmetric positive definite matrices. *Comput. Struct.*, 11(6):597–608, 1980. (Cited on page 193.)

[955] D. Rosenblatt. On linear models and the graphs of Minkowski-Leontief matrices. *Econometrica*, 25(2):25–338, 1957. (Cited on page 223.)

[956] J. P. Roth and D. S. Scott. A vector method for solving linear equations and inverting matrices. *J. Math. Phys.*, 35(1-4):312–317, 1956. (Cited on page 155.)

[957] E. Rothberg. *Exploiting the memory hierarchy in sequential and parallel sparse Cholesky factorization*. PhD thesis, Stanford University, 1993. (Cited on pages 139 and 142.)

[958] E. Rothberg. Alternatives for solving sparse triangular systems on distributed-memory computers. *Parallel Comput.*, 21:1121–1136, 1995. (Cited on page 186.)

[959] E. Rothberg and S. C. Eisenstat. Node selection strategies for bottom-up sparse matrix orderings. *SIAM J. Matrix Anal. Appl.*, 19(3):682–695, 1998. (Cited on page 205.)

[960] E. Rothberg and A. Gupta. Efficient sparse matrix factorization on high-performance workstations - Exploiting the memory hierarchy. *ACM Trans. Math. Soft. (TOMS)*, 17(3):313–334, 1991. (Cited on pages 211 and 224.)

[961] E. Rothberg and A. Gupta. An evaluation of left-looking, right-looking, and multifrontal approaches to sparse Cholesky factorization on hierarchical-memory machines. *Intl. J. High Speed Computing*, 5(4):537–593, 1993. (Cited on pages 211 and 224.)

[962] E. Rothberg and A. Gupta. An efficient block-oriented approach to parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 15(6):1413–1439, 1994. (Cited on page 221.)

[963] F. H. Rouet, X. S. Li, P. Ghysels, and A. Napov. A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Soft. (TOMS)*, 42(4):1–35, 2016. (Cited on page 223.)

[964] J. W. Ruge and K. Stüben. Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG). In D. J. Paddon and H. Holstein, editors, *Multigrid Methods for Integral and Differential Equations*, pages 169–212. Clarendon Press, 1985. (Cited on page 421.)

[965] J. W. Ruge and K. Stüben. Algebraic multigrid. In S. F. McCormick, editor, *Multigrid Methods*, pages 73–130. SIAM, 1987. (Cited on pages 411 and 421.)

[966] A. Ruhe. Numerical aspects of Gram-Schmidt orthogonalization of vectors. *Linear Algebra Appl.*, 52:591–601, 1983. (Cited on page 46.)

[967] S. M. Rump and C.-P. Jeannerod. Improved backward error bounds for LU and Cholesky factorizations. *SIAM J. Matrix Anal. Appl.*, 35(2):684–698, 2014. (Cited on pages 119, 120, and 124.)

[968] S. M. Rump, I. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189–224, 2008. (Cited on page 31.)

[969] S. M. Rump, I. Ogita, and S. Oishi. Accurate floating-point summation part II: Sign, K-fold faithful and rounding to nearest. *SIAM J. Sci. Comput.*, 31(2):1269–1302, 2008. (Cited on page 31.)

[970] Y. Saad. Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. *Linear Algebra Appl.*, 34:269–295, 1980. (Cited on page 358.)

[971] Y. Saad. Krylov subspace methods for solving large nonsymmetric linear systems. *Math. Comput.*, 37:105–126, 1981. (Cited on pages 322, 325, and 359.)

[972] Y. Saad. Practical use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 5(1):203–228, 1984. (Cited on pages 314, 325, and 359.)

[973] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6(4):865–881, 1985. (Cited on pages 304, 384, 417, and 420.)

[974] Y. Saad. Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems. *SIAM J. Numer. Anal.*, 24(1):155–169, 1987. (Cited on pages 389 and 420.)

[975] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *J. Sci. Comput.*, 14(2):461–469, 1993. (Cited on page 330.)

[976] Y. Saad. ILUT: A dual threshold incomplete ILU preconditioner. *Numer. Linear Algebra Appl.*, 1:387–402, 1994. (Cited on pages 373 and 420.)

[977] Y. Saad. Analysis of augmented Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 18(2):435–449, 1997. (Cited on page 359.)

[978] Y. Saad. Multilevel ILU with reorderings for diagonal dominance. *SIAM J. Sci. Comput.*, 27(3):1032–1057, 2005. (Cited on pages 415 and 421.)

[979] Y. Saad and M. H. Schultz. GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems. Technical Report YALEU/DCS/RR-254, Yale University, USA, 1985. (Cited on pages 322 and 359.)

[980] Y. Saad and M. H. Schultz. GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986. (Cited on pages 314, 316, and 359.)

[981] Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numer. Linear Algebra Appl.*, 9(5):359–378, 2002. (Cited on pages 415 and 421.)

[982] Y. Saad and K. Wu. DQGMRES: A direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Numer. Linear Algebra Appl.*, 3(4):329–343, 1996. (Cited on page 325.)

[983] Y. Saad and J. Zhang. BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices. *SIAM J. Matrix Anal. Appl.*, 21(1):279–299, 1999. (Cited on page 415.)

[984] H. Sadok. CMRH: a new method for solving nonsymmetric linear systems based on the Hessenberg reduction algorithm. *Numer. Algorithms*, 20(4):303–321, 1999. (Cited on pages 331, 332, and 360.)

[985] H. Sadok. Analysis of the convergence of the minimal and the orthogonal residual methods. *Numer. Algorithms*, 40:201–216, 2005. (Cited on page 359.)

[986] H. Sadok and D. B. Szyld. A new look at CMRH and its relation to GMRES. *BIT Numerical Mathematics*, 52(2):485–501, 2012. (Cited on pages 334 and 360.)

[987] A. H. Sameh and R. P. Brent. Solving triangular systems on a parallel computer. *SIAM J. Numer. Anal.*, 14(6):1101–1113, 1977. (Cited on pages 138 and 139.)

[988] E. E. Santos. On designing optimal parallel triangular solvers. *Inf. Comput.*, 161(2):172–210, 2000. (Cited on page 140.)

[989] P. Sao, X. S. Li, and R. Vuduc. A communication-avoiding 3D algorithm for sparse LU factorization on heterogeneous systems. *J. Parallel Dist. Comput.*, 131:218–234, 2019. (Cited on page 222.)

[990] P. Sao, R. Vuduc, and X. S. Li. A distributed CPU-GPU sparse direct solver. In F. Silva, I. Dutra, and V. Santos Costa, editors, *Proc. Euro-Par 2014 Parallel Processing*, pages 487–498. Springer, 2014. (Cited on page 222.)

[991] N. Sato and W. F. Tinney. Techniques for exploiting the sparsity or the network admittance matrix. *IEEE Trans. Power App. Syst.*, 82(69):944–950, 1963. (Cited on page 216.)

[992] P. E. Saylor and D. C. Smolarski. Implementation of an adaptive algorithm for Richardson's method. *Linear Algebra Appl.*, 154(1991):615–646, 1991. (Cited on pages 388 and 420.)

[993] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker. PARDISO: A high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Gener. Comp. Syst.*, 18(1):69–78, 2001. (Cited on page 219.)

[994] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen I. *Math. Ann.*, 63:433–467, 1907. (Cited on page 78.)

[995] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen II. *Math. Ann.*, 64:161–174, 1907. (Cited on page 78.)

[996] E. Schmidt. Uber die auflösung linearer gleichungen mit unendlich vielen ubbekannten. *Rend. Circ. Mat. Palermo. Ser. 1*, 25:53–77, 1908. (Cited on page 78.)

[997] E. Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen III. *Math. Ann.*, 65(3):370–399, 1908. (Cited on page 78.)

[998] R. Schreiber. A new implementation of sparse Gaussian elimination. *ACM Trans. Math. Soft. (TOMS)*, 8(3):256–276, 1982. (Cited on pages 191, 193, and 224.)

[999] I. Schur. Ueber die charakteristischen wurzeln einer linearen substitution mit einer anwendung auf die theorie der integralgleichungen. *Math. Ann.*, 66:488–510, 1909. (Cited on page 78.)

[1000] H. A. Schwarz. Über einen grenzübergang durch alternierendes verfahren. *Vierteljahrsschrift Naturf. Ges. Zürich*, 15(3):272–286, 1870. (Cited on page 389.)

[1001] M. Schweitzer. Any finite convergence curve is possible in the initial iterations of restarted FOM. *Electron. Trans. Numer. Anal.*, 45:133–145, 2016. (Cited on page 323.)

[1002] J. S. Scott and M. Tůma. *Algorithms for sparse linear systems*. Birkhäuser, 2023. (Cited on pages 185 and 382.)

[1003] A. H. Sherman. Algorithm 533: NSPIV, a Fortran subroutine for sparse Gaussian elimination with partial pivoting. *ACM Trans. Math. Soft. (TOMS)*, 4(4):391–398, 1978. (Cited on page 218.)

[1004] A. H. Sherman. Algorithms for sparse Gaussian elimination with partial pivoting. *ACM Trans. Math. Soft. (TOMS)*, 4(4):330–338, 1978. (Cited on page 218.)

[1005] J. Sherman and W. J. Morrison. Adjustment of an inverse matrix corresponding in one element of a given matrix. *Ann. Math. Stat.*, 21(1):124–127, 1950. (Cited on page 73.)

[1006] H. D. Simon. *The Lanczos algorithm for solving symmetric linear systems*. PhD thesis, University of California, Berkeley, USA, 1982. (Cited on page 277.)

[1007] H. D. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra Appl.*, 61:101–131, 1984. (Cited on page 277.)

[1008] H. D. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comput.*, 42(165):115–142, 1984. (Cited on page 277.)

[1009] V. Simoncini. On the convergence of restarted Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 22(2):430–452, 2000. (Cited on page 323.)

[1010] V. Simoncini. Block triangular preconditioners for symmetric saddle-point problems. *Appl. Numer. Math.*, 49(1):63–80, 2004. (Cited on page 417.)

[1011] V. Simoncini. On a non-stagnation condition for GMRES and application to saddle point matrices. *Electron. Trans. Numer. Anal.*, 37:202–213, 2010. (Cited on pages 321 and 359.)

[1012] V. Simoncini and M. Benzi. Spectral properties of the Hermitian and skew-Hermitian splitting preconditioner for saddle point problems. *SIAM J. Matrix Anal. Appl.*, 26(2):377–389, 2004. (Cited on page 417.)

[1013] V. Simoncini and D. B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numer. Linear Algebra Appl.*, 14(1):1–59, 2007. (Cited on page 359.)

[1014] V. Simoncini and D. B. Szyld. New conditions for non-stagnation of minimal residual methods. *Numer. Math.*, 109:477–487, 2008. (Cited on pages 321 and 359.)

[1015] V. Simoncini and D. B. Szyld. Interpreting IDR as a Petrov-Galerkin method. *SIAM J. Sci. Comput.*, 32(4):1898–1912, 2010. (Cited on pages 348, 349, and 361.)

[1016] R. D. Skeel. Scaling for numerical stability in Gaussian elimination. *J. ACM*, 26(3):494–526, 1979. (Cited on pages 127, 128, and 129.)

[1017] G. L. G. Sleijpen and D. R. Fokkema. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 11:11–32, 1993. (Cited on pages 344 and 360.)

[1018] G. L. G. Sleijpen, P. Sonneveld, and M. B. van Gijzen. Bi-CGSTAB as an induced dimension reduction method. *Appl. Numer. Math.*, 60(11):1100–1114, 2010. (Cited on pages 347, 348, and 349.)

[1019] G. L. G. Sleijpen, H. van der Vorst, and D. R. Fokkema. BiCGstab(ℓ) and other hybrid Bi-CG methods. *Numer. Algorithms*, 7(1):75–109, 1994. (Cited on page 344.)

[1020] G. L. G. Sleijpen and M. B. van Gijzen. Exploiting BiCGstab(ℓ) strategies to induce dimension reduction. *SIAM J. Sci. Comput.*, 32:2687–2709, 2010. (Cited on page 351.)

[1021] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *Internat. J. Numer. Methods Engrg.*, 23(2):239–251, 1986. (Cited on pages 199 and 200.)

[1022] F. Sloboda. A parallel projection method for linear algebraic systems. *Aplikace Matematiky*, 23(3):185–198, 1978. (Cited on page 156.)

[1023] B. F. Smith. A domain decomposition algorithm for elliptic problems in three dimensions. *Numer. Math.*, 60:219–234, 1991. (Cited on page 406.)

[1024] B. F. Smith. An iterative substructuring algorithm for problems in three dimensions. In D. E. Keyes, T. F. Chan, G. Meurant, J. S. Scroggs, and R. G. Voigt, editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 91–98. SIAM, 1992. (Cited on page 406.)

[1025] B. F. Smith. An optimal domain decomposition preconditioner for the finite element solution of linear elasticity problems. *SIAM J. Sci. Statist. Comput.*, 13:364–378, 1992. (Cited on page 406.)

[1026] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996. (Cited on pages 406, 408, and 421.)

[1027] A. Smoktunowicz, J. L. Barlow, and J. Langou. A note on the error analysis of classical Gram-Schmidt. *Numer. Math.*, 105(2):299–313, 2006. (Cited on page 48.)

[1028] R. A. Snay. Reducing the profile of sparse symmetric matrices. *Bull. Géodésique*, 50(4):341–352, 1976. (Cited on page 199.)

[1029] E. Solomonik and J. W. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *17th European Conference on Parallel Processing, Part II*, pages 90–109. Springer, 2011. (Cited on page 143.)

[1030] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10(1):36–52, 1989. (Cited on page 338.)

[1031] P. Sonneveld. AGS-IDR-CGS-BiCGSTAB-IDR(s): The circle closed. A case of serendipity. In *Proceedings of the International Kyoto Forum 2008 on Krylov Subspace Methods*, pages 1–14, 2008. (Cited on page 361.)

[1032] P. Sonneveld. On the convergence behavior of IDR(s) and related methods. *SIAM J. Sci. Comput.*, 34(5):A2576–A2598, 2012. (Cited on page 351.)

[1033] P. Sonneveld. A history of Krylov product methods - A case of serendipity -. Technical Report 17-05, Insitute of Applied Mathematics, Delft University of Technology, The Netherlands, 2017. (Cited on page 361.)

[1034] P. Sonneveld and M. B. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM J. Sci. Comput.*, 31(2):1035–1062, 2008. (Cited on pages 349, 351, and 361.)

[1035] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM J. Matrix Anal. Appl.*, 13(1):357–385, 1992. (Cited on page 324.)

[1036] M. Sosonkina, L. T. Watson, R. K. Kapania, and H. F. Walker. A new adaptive GMRES algorithm for achieving high accuracy. *Numer. Linear Algebra Appl.*, 5:275–297, 1998. (Cited on page 324.)

[1037] B. Speelpenning. The generalized element method. Technical Report UIUCDCS-R-78-946, University of Illinois at Urbana-Champaign, 1978. (Cited on page 212.)

[1038] P. H. Sterbenz. *Floating-Point Computation*. Prentice-Hall, 1974. (Cited on page 15.)

[1039] G. W. Stewart. Conjugate direction methods for solving systems of linear equations. *Numer. Math.*, 21(4):285–297, 1973. (Cited on pages 159 and 162.)

[1040] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, 1973. (Cited on page 3.)

[1041] G. W. Stewart. The convergence of the method of conjugate gradients at isolated extreme points of the spectrum. *Numer. Math.*, 24:85–93, 1975. (Cited on page 303.)

[1042] G. W. Stewart. Research, development, and LINPACK. In J. R. Rice, editor, *Mathematical Software III*, pages 1–14, New York, 1977. Academic Press. (Cited on page 128.)

[1043] G. W. Stewart. On the perturbation of LU, Cholesky and QR factorizations. *SIAM J. Matrix Anal. Appl.*, 14(4):1141–1145, 1993. (Cited on page 125.)

[1044] G. W. Stewart. On the perturbation of LU and Cholesky factors. *IMA J. Numer. Anal.*, 17(1):1–6, 1995. (Cited on page 125.)

[1045] G. W. Stewart. *Matrix Algorithms, volume I: Basic Decompositions*. SIAM, Philadelphia, 1998. (Cited on page 3.)

[1046] G. W. Stewart. *Matrix Algorithms, volume II: Eigensystems*. SIAM, Philadelphia, 2001. (Cited on page 3.)

[1047] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, New York, 1990. (Cited on page 125.)

[1048] H. L. Stone. Iteraive solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.*, 5(3):530–558, 1968. (Cited on page 418.)

[1049] P. Stpiczyński. New data distribution for solving triangular systems on distributed memory machines. In *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 589–597, Berlin, Heidelberg, 2006. Springer. (Cited on page 140.)

[1050] Z. Strakoš. On the real convergence rate of the conjugate gradient method. *Linear Algebra Appl.*, 154-156:535–549, 1991. (Cited on pages 294 and 303.)

[1051] Z. Strakoš and A. Greenbaum. Open questions in the convergence analysis of the Lanczos process for the real symmetric eigenvalue problem. IMA preprint 934, 1992. (Cited on pages 278 and 303.)

[1052] Z. Strakoš and P. Tichý. On error estimation in the conjugate gradient method and why it works in finite precision computations. *Electron. Trans. Numer. Anal.*, 13:56–80, 2002. (Cited on pages 277, 282, and 304.)

[1053] Z. Strakoš and P. Tichý. Error estimation in preconditioned conjugate gradients. *BIT Numerical Mathematics*, 45:789–817, 2005. (Cited on page 304.)

[1054] G. Strang. *Linear Algebra and its Applications*. Academic Press, 1976. (Cited on pages 3, 50, and 78.)

[1055] J.-G. Sun. Componentwise perturbation bounds for some matrix decompositions. *BIT*, 32:702–714, 1992. (Cited on page 125.)

[1056] J.-G. Sun. Rounding-error and perturbation bounds for the Cholesky and LDL^T factorizations. *Linear Algebra Appl.*, 173:77–97, 1992. (Cited on page 125.)

[1057] W.-P. Tang. Generalized Schwarz splittings. *SIAM J. Sci. Statist. Comput.*, 13(2):573–595, 1992. (Cited on page 394.)

[1058] W.-P. Tang and W. L. Wan. Sparse approximate inverse smoother for multigrid. *SIAM J. Matrix Anal. Appl.*, 21(4), 2000. (Cited on page 412.)

[1059] P. Tichý. The shadow vector in the Lanczos method. Proceedings of the XIIIth Summer School Software and Algorithms of Numerical Mathematics, Nečtiny, 1999. (Cited on page 337.)

[1060] P. Tichý and J. Zítko. Derivation of BiCG from the conditions defining Lanczos method for solving a system of linear equations. *Appl. Math.*, 43(5):381–388, 1998. (Cited on page 337.)

[1061] O. Tingleff. Systems of linear equations solved by block Gauss-Jordan method using a transputer cube. Technical Report IMM-REF-1995-08, Institute of Mathematical Modeling, Technical University of Danmark, 1995. (Cited on page 150.)

[1062] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55(11):1801–1809, 1967. (Cited on pages 203 and 224.)

[1063] M. Tismenetsky and I. Efrat. An efficient preconditioning algorithm and its analysis. *Linear Algebra Appl.*, 80:252–256, 1986. (Cited on page 417.)

[1064] M. Tismenetsky and I. Efrat. A modified generalized minimal residual algorithm for nonsymmetric linear systems. Technical Report 88.219, IBM Israel, 1988. (Cited on page 417.)

[1065] D. Titley-Peloquin, J. Pestana, and A. J. Wathen. GMRES convergence bounds that depend on the right-hand-side vector. *IMA J. Numer. Anal.*, 34(2):462–479, 2014. (Cited on pages 317 and 320.)

[1066] S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18(4):1065–1081, 1997. (Cited on pages 94, 133, and 134.)

[1067] S. Tomov, J. J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Comput.*, 36(5-6):232–240, 2010. (Cited on page 142.)

[1068] C. H. Tong and Q. Ye. Analysis of the finite precision biconjugate gradient algorithm for nonsymmetric linear systems. *Math. Comput.*, 69:1559–1575, 2000. (Cited on page 337.)

[1069] A. Toselli and O. Widlund. *Domain Decompposition Methods - Algorithms and Theory*. Springer Series in Computational Mathematics, 2004. Vol. 34. (Cited on pages 408 and 421.)

[1070] E. Totoni, M. T. Heath, and L. V. Kale. Structure-adaptive parallel solution of sparse triangular linear systems. *Parallel Comput.*, 40(9):454–470, 2014. (Cited on page 186.)

[1071] L. N. Trefethen. Approximation theory and numerical linear algebra. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation II*, London, 1990. Chapman and Hall. (Cited on page 317.)

[1072] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton University Press, 2005. (Cited on page 317.)

[1073] L. N. Trefethen and R. S. Schreiber. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(3):335–360, 1990. (Cited on page 123.)

[1074] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Elsevier, 2000. (Cited on pages 411 and 421.)

[1075] N. Tsuno and T. Nodera. The speedup of the GMRES(m) method using the early restarting procedure. *Trans. Inform. Process. Soc. Japan*, 40:1760–1773, 1999. (Cited on page 324.)

[1076] A. D. Tuff and A. Jennings. An iterative method for large systems of linear structural equations. *Internat. J. Numer. Methods Engrg.*, 7(2):175–183, 1973. (Cited on pages 372 and 418.)

[1077] M. Tůma. Solving sparse unsymmetric sets of linear equations based on implicit Gauss projection. Technical Report 555, Institute of Computer Science, Academy of Sciences of the Czech Republic, 1993. (Cited on page 158.)

[1078] A. M. Turing. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.*, 1:287–308, 1948. (Cited on pages 78, 126, 183, and 418.)

[1079] D. J. Tylavsky. Quadrant interlocking factorization: A form of block LU factorization. *Proc. IEEE*, 74(1):232–233, 1986. (Cited on page 167.)

[1080] R. Underwood. An approximate factorization procedure based on the block Cholesky decomposition and its use with the conjugate gradient method. Technical Report NEDO-11386, General Electric, San José, USA, 1976. (Cited on page 378.)

[1081] S. Ursic. Inverse matrix representation with one triangular array. *Linear Algebra Appl.*, 47:151–157, 1982. (Cited on page 158.)

[1082] A. van der Ploeg, E. F. F. Botta, and F. W. Wubs. Nested grids ILU decomposition (NGILU). *J. Comput. Appl. Math.*, 66:515–526, 1996. (Cited on page 415.)

[1083] A. van der Sluis. Condition numbers and equilibration of matrices. *Numer. Math.*, 14(1):14–23, 1969. (Cited on pages 128 and 365.)

[1084] A. van der Sluis. Condition, equilibration and pivoting in linear algebraic systems. *Numer. Math.*, 15(1):74–86, 1970. (Cited on page 128.)

[1085] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48:543–560, 1986. (Cited on pages 274 and 303.)

[1086] H. van der Vorst. BICGSTAB: A fast and smoothly converging variant of BI-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992. (Cited on pages 341, 342, and 360.)

[1087] H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003. (Cited on page 341.)

[1088] H. van der Vorst and P. Sonneveld. CGSTAB, a more smoothly converging variant of CG-S. Technical report, Department of Mathematics and Informatics, Delft University of Technology, 1990. (Cited on page 360.)

[1089] M. B. van Gijzen, G. L. G. Sleijpen, and J. P. M. Zemke. Flexible and multi-shift induced dimension reduction algorithms for solving large sparse linear systems. *Numer. Linear Algebra Appl.*, 22(1):1–25, 2015. (Cited on page 351.)

[1090] M. B. van Gijzen and P. Sonneveld. Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties. *ACM Trans. Math. Soft. (TOMS)*, 38(1):5.1–5.19, 2011. (Cited on pages 349 and 361.)

[1091] J. Van Rosendale. Minimizing inner product data dependencies in conjugate gradient iteration. Technical Report TR 172178, ICASE, NASA Langley Research Center, 1983. (Cited on pages 285 and 304.)

[1092] N. Van Tran, J. Justino, and I. van den Berg. On the explicit formula for Gauss-Jordan elimination. ArXiv preprint arXiv:2010.01085, 2020. (Cited on page 150.)

[1093] R. Vandebril, M. Van Barel, and N. Mastronardi. *Matrix Computations and Semiseparable Matrices, Volume I: Linear Systems*. Johns Hopkins University Press, Baltimore, MD, 2008. (Cited on pages 111 and 223.)

[1094] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996. (Cited on page 413.)

[1095] P. Vaněk, J. Mandel, and M. Brezina. Convergence of algebraic multigrid based on smoothed aggregation. *Numer. Math.*, 88:559–579, 2001. (Cited on page 413.)

[1096] R. S. Varga. A comparison of the successive overrelaxation method and semi-iterative methods using Chebyshev polynomials. *J. Soc. Ind. Appl. Math.*, 5(2):39–46, 1957. (Cited on page 251.)

[1097] R. S. Varga. Factorization and normalized iterative methods. In R. E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142. University of Wisconsin Press, Madison, 1960. Proceedings of a Symposium conducted by the Mathematics Research Center at the University of Wisconsin, Madison, April 20-22, 1959. (Cited on pages 371 and 418.)

[1098] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1962. A revised and expanded edition was published by Springer in 2000. (Cited on pages 3, 60, 61, 78, 259, and 303.)

[1099] R. S. Varga and D.-Y. Cai. On the LU factorization of M-matrices. *Numer. Math.*, 38:179–192, 1981. (Cited on page 109.)

[1100] R. S. Varga, E. B. Saff, and V. Mehrmann. Incomplete factorizations of matrices and connections with H-matrices. *SIAM J. Numer. Anal.*, 17(6):787–793, 1980. (Cited on pages 369 and 418.)

[1101] P. S. Vassilevski. Hybrid V-cycle algebraic multilevel preconditioners. *Math. Comput.*, 58:489–512, 1992. (Cited on page 415.)

[1102] P. S. Vassilevski and J. Wang. Wavelet-like methods in the design of efficient multilevel preconditioners for elliptic PDEs. In *Wavelet Analysis and Its Applications. Vol. 6*, pages 59–105. Academic Pres, 1997. (Cited on page 416.)

[1103] E. Vecharynski and J. Langou. Any admissible cycle-convergence behavior is possible for restarted GMRES at its initial cycles. *Numer. Linear Algebra Appl.*, 18(3):499–511, 2011. (Cited on page 323.)

[1104] P. K. W. Vinsome. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. In *Proceedings of the Fourth SPE Symposium on Numerical Simulation of Reservoir Performance*. Society of Petroleum Engineers of AIME, 1976. (Cited on page 329.)

[1105] J. von Neumann and H. H. Goldstine. Numerical inverting of matrices of high order. *Bull. Amer. Math. Soc.*, 53:1021–1099, 1947. (Cited on pages 78, 123, and 183.)

[1106] H. F. Walker. Implementation of the GMRES and Arnoldi methods using Householder transformations. Technical Report UCRL-93589, Lawrence Livermore Laboratory, 1985. (Cited on page 313.)

[1107] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9(1):152–163, 1988. (Cited on page 313.)

[1108] H. F. Walker and L. Zhou. A simpler GMRES. *Numer. Linear Algebra Appl.*, 1(6):571–581, 1994. (Cited on page 326.)

[1109] W. L. Wan, T. F. Chan, and B. Smith. An energy-minimizing interpolation for robust multigrid methods. *SIAM J. Sci. Comput.*, 21(4):1632–1649, 1999. (Cited on page 413.)

[1110] F. Wang, C. Q. Yang, Y. F. Du, J. Chen, H. Z. Yi, and W. X. Xu. Optimizing LINPACK benchmark on GPU-accelerated petascale supercomputer. *J. Comput. Sci. Technol.*, 26(5):854–865, 2011. (Cited on page 144.)

[1111] K. Wang and J. Zhang. MSP: A class of parallel multistep successive sparse approximate inverse preconditioning strategies. *SIAM J. Sci. Comput.*, 24(4):1141–1156, 2003. (Cited on page 417.)

[1112] S. Wang, X. S. Li, F. H. Rouet, J. Xia, and M. V. De Hoop. A parallel geometric multifrontal solver using hierarchically semiseparable structure. *ACM Trans. Math. Soft. (TOMS)*, 42(3):1–21, 2016. (Cited on page 223.)

[1113] J. H. M. Wedderburn. *Lectures on Matrices*. American Mathematical Society, 1934. (Cited on page 162.)

[1114] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley, 1992. (Cited on pages 411 and 421.)

[1115] P. Wesseling and P. Sonneveld. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In R. Rautmann, editor, *Approximation Methods for Navier-Stokes Problems, Paderborn, Germany 1979, Lecture Notes in Mathematics*. Springer, 1980. (Cited on pages 347, 348, and 361.)

[1116] T. Wicky, E. Solomonik, and T. Hoefler. Communication-avoiding parallel algorithms for solving triangular systems of linear equations. In *International Parallel and Distributed Processing Symposium*, pages 678–687. IEEE, 2017. (Cited on page 141.)

[1117] O. B. Widlund and C. R. Dohrmann. BDDC deluxe domain decomposition. In T. Dickhopf, M. J. Gander, L. Halpern, R. Krause, and L. F. Pavarino, editors, *Domain decomposition methods in science and engineering XXII*, pages 93–103. Springer, 2016. (Cited on page 407.)

[1118] O. B. Widlund and M. Dryja. An additive variant of the Schwarz alternating method for the case of many subregions. Technical Report Technical Report 339, Ultracomputer Note 131, Department of Computer Science, Courant Institute, 1987. (Cited on page 393.)

[1119] J. H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. ACM*, 8:281–330, 1961. (Cited on page 41.)

[1120] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, Oxford, UK, 1965. (Cited on pages 3, 41, 78, 92, 120, 122, 123, 183, 358, and 360.)

[1121] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation, Volume II, Linear Algebra*. Springer-Verlag, 1971. (Cited on page 185.)

[1122] M. A. Woodbury. Inverting modified matrices. Technical Report Memorandum Rept. 42, Statistical Research Group, Princeton University, Princeton, USA, 1950. (Cited on page 73.)

[1123] S. J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Comput.*, 14(4):231–238, 1993. (Cited on page 123.)

[1124] K. Wu. Restarted variants of DQGMRES. In *Proceedings of 1997 International Workshop on Computational Science and Engineering*, 1997. (Cited on page 325.)

[1125] K. Wu and H. Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22(2):602–616, 2000. (Cited on pages 324 and 360.)

[1126] Y. Xi, R. Li, and Y. Saad. An algebraic multilevel preconditioner with low-rank corrections for sparse symmetric matrices. *SIAM J. Matrix Anal. Appl.*, 37(1):235–259, 2016. (Cited on pages 415 and 416.)

[1127] J. Xia. Randomized sparse direct solvers. *SIAM J. Matrix Anal. Appl.*, 34(1):197–227, 2013. (Cited on page 223.)

[1128] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Superfast multifrontal method for structured linear systems of equations. *SIAM J. Matrix Anal. Appl.*, 31(3):1382–1411, 2009. (Cited on page 223.)

[1129] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebra Appl.*, 17(6):953–976, 2010. (Cited on page 223.)

[1130] Y. Xia, P. Jiang, G. Agrawal, and R. Ramnath. End-to-end LU factorization of large matrices on GPUs. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pages 288–300. ACM, 2023. (Cited on page 222.)

[1131] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017. (Cited on pages 411 and 414.)

[1132] T. Xu, V. Kalantzis, R. Li, Y. Xi, G. Dillon, and Y. Saad. parGeMSLR: A parallel multilevel Schur complement low-rank preconditioning and solution package for general sparse matrices. *Parallel Comput.*, 113:102956, 2022. (Cited on pages 415 and 416.)

[1133] P. Yalamov and D. J. Evans. Round-off analysis of the WZ matrix factorisation method. *Int. J. Comput. Math.*, 53(1-2):61–81, 1994. (Cited on page 167.)

[1134] P. Yalamov and D. J. Evans. The WZ matrix factorisation method. *Parallel Comput.*, 21(7):1111–1120, 1995. (Cited on page 167.)

[1135] N. Yamanaka, T. Ogita, S. M. Rump, and S. Oishi. A parallel algorithm for accurate dot product. *Parallel Comput.*, 34(6-8):392–410, 2008. (Cited on page 36.)

[1136] L. T. Yang and R. P. Brent. Quantitative performance analysis of the improved quasi-minimal residual method on massively distributed memory computers. *Adv. Eng. Softw.*, 33(3):169–177, 2002. (Cited on page 347.)

[1137] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2(1):77–79, 1981. (Cited on page 187.)

[1138] A. YarKhan, J. Kurzak, P. Luszczek, and J. J. Dongarra. Porting the PLASMA numerical library to the OpenMP standard. *Int. J. Parallel Program.*, 45:612–633, 2017. (Cited on page 145.)

[1139] M. C. Yeung. Probabilistic analysis of complex Gaussian elimination without pivoting. *Linear Algebra Appl.*, 384:109–134, 2004. (Cited on page 123.)

[1140] M. C. Yeung. ML(n) BiCGSTAB: reformulation, analysis and implementation. *Numer. Math. Theor. Meth. Appl.*, 5(3):447–492, 2012. (Cited on pages 351 and 361.)

[1141] M. C. Yeung and T. F. Chan. Probabilistic analysis of Gaussian elimination without pivoting. *SIAM J. Matrix Anal. Appl.*, 18(2):499–517, 1997. (Cited on page 123.)

[1142] M. C. Yeung and T. F. Chan. ML(K)BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors. *SIAM J. Sci. Comput.*, 21(4):1263–1290, 1999. (Cited on pages 351 and 361.)

[1143] D. M. Young. *Iterative methods for solving partial difference equations of elliptic type*. PhD thesis, Harvard University, Cambridge, USA, 1950. (Cited on page 235.)

[1144] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971. Reprinted by Dover (2003). (Cited on pages 3, 53, 54, 56, 78, 230, 235, 238, 243, and 259.)

[1145] D. M. Young. On the accelerated SSOR method for solving large linear systems. *Advances in Maths.*, 23:215–271, 1977. (Cited on page 240.)

[1146] D. M. Young and K. C. Jea. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980. (Cited on pages 329 and 360.)

[1147] H. Yserentant. Two preconditioners based on the multilevel splitting of finite element spaces. *Numer. Math.*, 58:163–184, 1990. (Cited on page 415.)

[1148] C. D. Yu, W. Wang, and D. Pierce. A CPU-GPU hybrid approach for the unsymmetric multifrontal method. *Parallel Comput.*, 37(12):759–770, 2011. (Cited on page 222.)

[1149] I. Zavorin. *Analysis of GMRES convergence by spectral factorization of the Krylov matrix*. PhD thesis, University of Maryland, USA, 2001. (Cited on pages 321 and 359.)

[1150] I. Zavorin, D. P. O'Leary, and H. C. Elman. Complete stagnation of GMRES. *Linear Algebra Appl.*, 367:165–183, 2003. (Cited on pages 321 and 359.)

[1151] J. P. M. Zemke. *Krylov subspace methods in finite precision: A unified approach*. PhD thesis, Technical University of Hamburg, Germany, 2003. (Cited on page 329.)

[1152] J. P. M. Zemke. Variants of IDR with partial orthonormalization. *Electron. Trans. Numer. Anal.*, 46:245–272, 2017. (Cited on pages 351 and 361.)

[1153] J. Zhang. On preconditioning Schur complement and Schur complement preconditioning. *Electron. Trans. Numer. Anal.*, 10:115–130, 2000. (Cited on page 417.)

[1154] J. Zhang. Sparse approximate inverse and multilevel block ILU preconditioning techniques for general sparse matrices. *Appl. Numer. Math.*, 35:67–86, 2000. (Cited on pages 415 and 417.)

[1155] L. Zhang and T. Nodera. A new adaptive restart for GMRES(m) method. *ANZIAM J.*, 46:409–425, 2005. (Cited on page 324.)

[1156] X. Zhang. Multilevel Schwarz methods. *Numer. Math.*, 63:521–539, 1992. (Cited on pages 415 and 416.)

[1157] Q. Zheng, Y. Xi, and Y. Saad. Multicolor low-rank preconditioner for general sparse linear systems. *Numer. Linear Algebra Appl.*, 27(4):e2316, 2020. (Cited on page 416.)

[1158] Y.-K. Zhu and W.B. Hayes. Correct rounding and a hybrid approach to exact floating-point summation. *SIAM J. Sci. Comput.*, 31(4):2981–3001, 2009. (Cited on page 31.)

[1159] Y.-K. Zhu and W.B. Hayes. Algorithm 908: Online exact summation of floating-point streams. *ACM Trans. Math. Soft. (TOMS)*, 37(3):1–13, 2010. (Cited on page 31.)

[1160] J. Zítko. Generalization of convergence conditions for a restarted GMRES. *Numer. Linear Algebra Appl.*, 7:117–131, 2000. (Cited on page 323.)

[1161] J. Zítko. Convergence conditions for a restarted GMRES method augmented with eigenspaces. *Numer. Linear Algebra Appl.*, 12(4):373–390, 2005. (Cited on page 323.)

[1162] J. Zítko. How residual bounds for restarted GMRES describe the real behaviour. In *PAMM: Proceedings in Applied Mathematics and Mechanics*, volume 8. Wiley-VCH Verlag, 2008. (Cited on page 323.)

[1163] J. Zítko. Some remarks on the restarted and augmented GMRES method. *Electron. Trans. Numer. Anal.*, 31:221–227, 2008. (Cited on page 323.)

[1164] J. Zítko and D. Nádhera. Behaviour of the augmented GMRES method. In *SNA'10, Nové Hrady*, Prague, Czech Republic, 2010. Charles University. (Cited on page 323.)

[1165] Z. Zlatev. On some pivotal strategies in Gaussian elimination by sparse technique. *SIAM J. Numer. Anal.*, 17(1):18–30, 1980. (Cited on page 217.)

[1166] Z. Zlatev. Use of iterative refinement in the solution of sparse linear systems. *SIAM J. Numer. Anal.*, 19(2):381–399, 1982. (Cited on page 129.)

[1167] K. Zorychta. Comparison of Purcell's method with the elimination method of Gauss. *Computatio*, 2:65–74, 1964. (Cited on page 156.)

[1168] M. Zounon, N. J. Higham, C. Lucas, and F. Tisseur. Performance impact of precision reduction in sparse linear systems solvers. *PeerJ Computer Science*, 8:e778, 2022. (Cited on page 223.)

Index