Iteratively Solving Large Sparse Linear Systems on Parallel Computers

H. Martin Bücker

published in

Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes,
J. Grotendorst, D. Marx, A. Muramatsu (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **10**, ISBN 3-00-009057-6, pp. 521-548, 2002.

Iteratively Solving Large Sparse Linear Systems on Parallel Computers

H. Martin Bücker

Institute for Scientific Computing
Aachen University of Technology, 52056 Aachen, Germany
E-mail: buecker@sc.rwth-aachen.de

Large systems of linear equations arise frequently as building blocks in many areas of scientific computing. Often, these linear systems are somehow structured or sparse, i.e., only a small number of the entries of the coefficient matrix are nonzero. In this note, numerical techniques for the solution of such linear systems are surveyed starting with a description of direct methods. When direct methods lead to excessive fill-in or when the coefficient matrix is not explicitly available, iterative methods enter the picture. These methods involve the coefficient matrix solely in the form of matrix-vector multiplications eliminating the problems of direct methods. After a summary of classical iterative methods based on relaxation of coordinates, the focus is on modern iterative methods making use of projection techniques. In particular, Krylov subspace methods are explained with an emphasis on their underlying structure rather than on their implementation details. Additional topics that are indispensable in the context of parallel computing such as reducing synchronization overhead and graph partitioning are also covered.

1 An Algorithmic Shift in Large-Scale Computations

Why would you want more than Gaussian elimination for the solution of systems of linear equations? The answer is that, sometimes, you *have* to use different techniques – simply to get a solution. In situations where the coefficient matrix is large and sparse, Gaussian elimination is often not applicable because of its excessive storage requirements. Of course, what is considered to be "large" varies with time. The meaning of "sparse" is somewhat vague too. A common definition is due to Wilkinson who called a matrix "sparse" *whenever it is possible to take advantage of the number and location of its nonzero entries*. In this survey, we assume that "sparse" means the usage of an appropriate storage scheme such that, given an $N \times N$ matrix A and some N-dimensional vector $\mathbf{x}$, then the number of arithmetic operations needed to compute the matrix-vector multiplication $A\mathbf{x}$ is small, say N or $N \log N$, compared to the N^2 operations of the conventional matrix-vector multiplication. Note that there are dense, but somehow structured matrices, for instance Toeplitz matrices, for which a matrix-vector multiplication can be carried out in $N \log N$ time or even better.

Under the assumption of being capable of efficiently computing a matrix-vector multiplication, we will survey numerical techniques for the solution of systems of linear equations

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where the coefficient matrix A is nonsingular. We will concentrate on nonsymmetric matrices and refer the reader to the book by Fischer[1] for the symmetric case where, among others, the well-known conjugate gradient (CG) method for symmetric positive definite systems is described.

In Sec. 2, we will show the reason why, in large-scale computations, there is a shift from direct methods whose most prominent representative is Gaussian elimination to iterative methods. The discussion of iterative methods begins with classical techniques summarized in Sec. 3. We will then lead over to the general framework of projection methods to start the survey of modern iterations in Sec. 4. Krylov subspace methods described in Sec. 5 fall under the class of projection methods. These methods are commonly considered to be among the most powerful iterative methods when combined with preconditioning techniques briefly mentioned in Sec. 6. On parallel computers, a number of additional issues are raised including the reduction of synchronization cost, explained in Sec. 7, and graph partitioning to efficiently compute a matrix-vector multiplication outlined in Sec. 8.

2 Difficulties with Direct Methods

Direct methods constitute one of the two classes of techniques for the solution of linear systems of type (1). In these methods, the exact solution $\mathbf{x}_* = A^{-1}\mathbf{b}$ is obtained after an a priori known, definite number of successive transformations. The storage for the coefficient matrix is usually overwritten during the course of the process by explicitly manipulating rows and columns of the matrix. Prominent examples of direct methods for nonsymmetric and symmetric positive definite systems are Gaussian elimination and Cholesky factorization, respectively.

2.1 Gaussian Elimination

Gaussian elimination is a typical direct approach for non-Hermitian linear systems. In the first phase of solving linear systems by Gaussian elimination, a decomposition or factorization of the form

$$A = PLU \tag{2}$$

where P is an $N \times N$ permutation matrix is computed. Furthermore, the factor

$$L = \begin{bmatrix} 1 & & & \\ * & 1 & & \\ \vdots & \ddots & \ddots & \\ * & \dots & * & 1 \end{bmatrix}$$

is a Lower triangular $N \times N$ matrix with unit diagonal entries and the factor

$$U = \begin{bmatrix} * & \dots & \dots & * \\ & * & & \vdots \\ & & \ddots & \vdots \\ & & & * \end{bmatrix}$$

is an Upper triangular matrix of the same size.

In the second phase of solving a linear system by means of Gaussian elimination, the original problem (1) is reformulated in terms of the factors L and U by using (2). Since permutation matrices are orthogonal, i.e., they satisfy $P^{-1} = P^T$, the result of the reformulation is given by the two linear systems

$$L\mathbf{y} = P^T\mathbf{b} \qquad \text{and} \qquad U\mathbf{x} = \mathbf{y}.$$

This reformulation, at first, appears unreasonable because a single linear system is replaced by two linear systems of the same size. The key idea behind the approach is that the two resulting systems are "extremely easy" to solve due to their tridiagonal structure.

Time complexity of Gaussian elimination is given by $2N^3/3 + \Theta(N^2)$ arithmetic operations. The computational dominant part of Gaussian elimination is the computation of the factorization (2); that is, the computation of L and U. Gaussian elimination can be implemented in-place meaning that the entries of A are overwritten by the entries of L and U during the course of the process. Thus, the storage requirement of Gaussian elimination is $N^2 + \Theta(N)$.

2.2 Cholesky Factorization

For Hermitian positive definite systems, the Gaussian elimination simplifies to a method known as Cholesky factorization. In the first phase of a Cholesky factorization, a decomposition

$$A = LL^H \tag{3}$$

is computed where, as before, L is a *L*ower triangular matrix

$$L = \begin{bmatrix} * & & & \\ \vdots & * & & \\ \vdots & & \ddots & \\ * & \dots & \dots & * \end{bmatrix}$$

but now with general, real diagonal entries $l_{ii} > 0$. When there is ambiguity, the so-called Cholesky triangle is indexed by the matrix to be decomposed; that is, the symbol L_A is used to denote the Cholesky triangle of a matrix A satisfying (3).

In the second phase of a Cholesky factorization, the original problem (1) is solved in terms of the Cholesky triangle:

$$L\mathbf{y} = \mathbf{b} \qquad \text{and} \qquad L^H\mathbf{x} = \mathbf{y}.$$

As in Gaussian elimination, the dominant part of the Cholesky factorization is the computation of the decomposition (3); that is, the computation of the Cholesky triangle L. The overall time complexity is $N^3/3 + \Theta(N^2)$ arithmetic operations. It is possible to arrange the Cholesky factorization so that L overwrites the lower triangle of A. Thus, the storage requirement of the Cholesky factorization is $N^2/2+\Theta(N)$. Note that, compared to Gaussian elimination for general matrices, the factorization for Hermitian positive definite matrices needs approximately half as much operations as well as half of the storage.

2.3 Additional Remarks

The situation in Gaussian elimination and Cholesky factorization is rather typical for direct methods in the following sense:

- Direct methods commonly proceed in two phases. During the first, computationally intensive phase, a decomposition of the coefficient matrix into factors is computed, either implicitly or explicitly. In the second phase, the original problem is reformulated and finally solved in terms of these factors.

- The number of arithmetic operations and the storage requirement of direct methods is often cubic and square, respectively, in the order of the coefficient matrix. Both properties are likely to be unacceptable in large-scale computations where the order of the matrix is rapidly increasing with time.

The discussion given in the preceding two subsections presents the direct solution of systems of linear equations in terms of matrices. The traditional implementations on a scalar level vary significantly and their performance on today's computers with deep memory hierarchies differ dramatically. Certain reorganizations of the algorithms in terms of matrix-vector and matrix-matrix operations rather than on scalar operations are known to substantially increase the performance as discussed by Dongarra et al.[2]. Rather than concentrating on these techniques of tuning performance, the focus here is on two serious and inherent weaknesses of direct methods when applied to large and sparse—as opposed to small and dense—systems.

2.4 The Problem of Fill-in

The following well-known example lucidly explains the difficulties of direct methods in the context of sparsity. Suppose that the task is to solve a symmetric positive definite system of order N with a sparse coefficient matrix

$$A = \begin{bmatrix} * & * & * & * & * & \cdots & * & * & * \\ * & * & & & & & & & \\ * & & * & & & & & & \\ * & & & * & & & & & \\ * & & & & * & & & & \\ \vdots & & & & & \ddots & & & \\ * & & & & & & * & & \\ * & & & & & & & * & \\ * & & & & & & & & * \end{bmatrix} \qquad (4)$$

whose sparsity pattern is given in the form of an arrow. Since A is symmetric a sparse storage scheme needs only to store $2N-1$ nonzero elements of A for its complete representation. The corresponding Cholesky triangle is given by

$$L_A = \begin{bmatrix} * & & & & & & & & \\ * & * & & & & & & & \\ * & * & * & & & & & & \\ * & * & * & * & & & & & \\ * & * & * & * & * & & & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & & & \\ * & * & * & * & * & \cdots & * & & \\ * & * & * & * & * & \cdots & * & * & \\ * & * & * & * & * & \cdots & * & * & * \end{bmatrix} \qquad (5)$$

and consists of $\Theta(N^2)$ nonzero entries. By comparing the sparsity of the lower triangular part of A and L_A, we find that L_A has a lot more nonzero entries than A. The phenomenon of turning a zero element of a sparse matrix into a nonzero element during a factorization is called fill-in. This kind of behavior is by no means restricted to the Cholesky factorization but applies to different factorization schemes as well. Fill-in is a general phenomenon of direct methods and may lead to severe storage problems in the context of high-performance computing. If A is large it may already be hard to keep its sparse representation within the limits of available storage capacity. Thus, fill-in is a measure of memory needed in addition to the extreme amount of storage used in high-performance applications anyway.

It is straight forward to ask whether the sparsity pattern of A has an influence on the amount of fill-in produced during a factorization. A different sparsity pattern results from renaming of the unknowns and reordering of the equations which can be represented by a particular kind of permutation defined as follows. Given a matrix A as well as a permutation P, the matrix P^TAP is said to be a symmetric permutation of A. A symmetric permutation of the matrix in (4) is given by

$$P^TAP = \begin{bmatrix}
* & & & & & & & & * \\
& * & & & & & & & * \\
& & * & & & & & & * \\
& & & * & & & & & * \\
& & & & * & & & & * \\
& & & & & \ddots & & & \vdots \\
& & & & & & * & & * \\
& & & & & & & * & * \\
* & * & * & * & * & \cdots & * & * & *
\end{bmatrix} \tag{6}$$

where only the first and last components and the two corresponding equations are permuted. For symmetric positive definite matrices A, it is possible to show that any symmetric permutation P^TAP is also symmetric positive definite. In other words, symmetric permutations preserve the property of symmetric positive definiteness. So, the Cholesky factorization can be applied to the matrix P^TAP in (6) leading to a Cholesky triangle

$$L_{P^TAP} = \begin{bmatrix}
* & & & & & & & & \\
& * & & & & & & & \\
& & * & & & & & & \\
& & & * & & & & & \\
& & & & * & & & & \\
& & & & & \ddots & & & \\
& & & & & & * & & \\
& & & & & & & * & \\
* & * & * & * & * & \cdots & * & * & *
\end{bmatrix} . \tag{7}$$

Comparing the Cholesky triangles L_A in (5) and L_{P^TAP} in (7), we observe that the sparsity pattern of the matrix to which a Cholesky factorization is applied does have a significant effect on the sparsity pattern of the matrix generated by the Cholesky factorization. The number of nonzero elements of L_A is $\Theta(N^2)$ whereas there are only $\Theta(N)$ nonzero elements in L_{P^TAP}. Unfortunately, the computation of a symmetric permutation leading to the minimum number of fill-in turns out to be a hard combinatorial optimization problem. More precisely, the so-called minimum fill-in problem is NP-complete meaning that, currently, there is no deterministic algorithm for its solution where the number of arithmetic operations scales polynomially with the order of the matrix. Moreover, from the point of view of theoretical computer science, it is very unlikely that one will ever find such an algorithm; see the book by Garey and Johnson[3] for more information on NP-completeness.

In summary, direct methods applied to sparse linear systems may lead to a dramatically high amount of fill-in prohibiting their use in large-scale applications. Furthermore, there is currently no computationally efficient technique to compute the minimum fill-in. Recently, an approximation algorithm for the minimum fill-in problem was developed[4] but its suitability for practical use is still open.

2.5 The Problem of Needing Explicit Access to the Matrix

Direct methods manipulate rows or columns of the coefficient matrix. Therefore, there is need to explicitly access entries of the coefficient matrix. In some applications, however, the matrix is not explicitly given. That is, the computation of the complete matrix is extremely expensive in terms of arithmetic operations whereas there is a computationally efficient procedure to compute the product of the coefficient matrix and some given vector.

An example of such a situation occurs in the solution of nonlinear systems of equations by Newton-type methods. Here, a subtask is to repeatedly solve linear systems of the form $J\mathbf{x} = \mathbf{b}$, where J is the Jacobian matrix of some function f. If there is a program in a high-level programming language like Fortran or C evaluating f for a given set of inputs, a technique called automatic differentiation is applicable to provide efficient code for computing Jacobian-vector multiplications. In contrast to numerical differentiation based on divided differencing delivering approximations to derivatives, automatic differentiation produces derivatives accurate up to machine precision. See the book by Griewank[5] or the web portal `http://www.autodiff.org` for an introduction to and more details on automatic differentiation.

The computation of all entries of J by automatic differentiation is more efficient than numerical differentiation under a wide range of circumstances[6–8]. The crucial point in the context of this note is the fact that, using automatic differentiation, Jacobian-vector multiplications are computationally even N times more efficient than computing all entries of J. From a conceptual point of view, one may explain the factor N by comparing a single matrix-vector multiplication and a sequence of N matrix-vector multiplications $J\mathbf{e}_1, J\mathbf{e}_2, \dots, J\mathbf{e}_N$, where $\mathbf{e}_i$ is the ith Cartesian unit vector, to compute all columns of J.

Since iterative methods make use of the coefficient matrix in the form of matrix-vector multiplications they do not suffer from neither the problem of fill-in nor from the problem of needing explicit access to the coefficient matrix.

3 Classical Iterations

Iterative methods enter the picture when direct methods produce excessive fill-in or the coefficient matrix is not explicitly available. By using the coefficient matrix in the form of matrix-vector multiplications, iterative methods are capable of handling these situations.

In their nth step, iterative methods compute approximations $\mathbf{x}_n$ to the exact solution $\mathbf{x}_* = A^{-1}\mathbf{b}$ of the linear system (1). The corresponding residual vector is defined by

$$\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n \tag{8}$$

and determines how far the approximation $\mathbf{x}_n$ is from the right hand side $\mathbf{b}$. The goal of any iterative method is to drive the residual vector to the zero vector because in this case $\mathbf{b} = A\mathbf{x}_n$ and thus the approximation $\mathbf{x}_n$ equals the exact solution $\mathbf{x}_*$. As a matter of fact, it is indispensable that, in order to beat the $\Theta(N^3)$ time complexity of direct methods, the approximations $\mathbf{x}_n$ should converge fast to the exact solution or a sufficiently accurate solution. To indicate this, the notation (big) N is used to denote the order of a (large) matrix and the symbol (little) n is used for a (small) iteration index expressing the desirable

[$\mathbf{x}_n$, $\mathbf{r}_n$] = RELAXATION(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$ with splitting $A = M - S$ with nonsingular M, this algorithm computes approximations $\mathbf{x}_n$ (with corresponding residuals $\mathbf{r}_n$) to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

```
1: Choose $\mathbf{x}_0 \in \mathbb{C}^N$, set $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$, and solve $M\mathbf{z}_0 = \mathbf{r}_0$
2: for $n = 1, 2, 3, \ldots$ do {until convergence}
3:     $\mathbf{x}_n \leftarrow \mathbf{z}_{n-1} + \mathbf{x}_{n-1}$
4:     $\mathbf{r}_n \leftarrow \mathbf{b} - A\mathbf{x}_n$
5:     Solve $M\mathbf{z}_n = \mathbf{r}_n$
6: end for
```

Figure 1. General form of a classical iterative method based on relaxation of coordinates.

relation $n \ll N$; that is, any viable iterative method should converge in a number of steps that is significantly smaller than the order of the matrix.

Classical iterative methods for the solution of linear systems date back at least to the 19th century. They are characterized by defining the approximations by a sequence of the form

$$M\mathbf{x}_n = \mathbf{b} + S\mathbf{x}_{n-1} \tag{9}$$

where

$$A = M - S \tag{10}$$

is a general matrix splitting. Given any starting vector $\mathbf{x}_0$, these iterative methods obtain the next approximation by modifying one or a few components of the current approximation. This class of methods is said to be based on relaxation of coordinates.

To derive a basic formulation of relaxation methods, observe that inserting S from (10) into (9) yields

$$M\mathbf{x}_n = \mathbf{b} - A\mathbf{x}_{n-1} + M\mathbf{x}_{n-1}.$$

If M is nonsingular an equivalent form is given by

$$\mathbf{x}_n = M^{-1}\mathbf{r}_{n-1} + \mathbf{x}_{n-1}.$$

The resulting process is depicted in Fig. 1. It is important to remark that linear systems with coefficient matrices M should be "easy" to solve because these systems are to be solved in each step of the iteration.

The Jacobi iteration and the Gauss–Seidel iteration are popular examples of these classical iterative methods. If D, L, and U denote the diagonal, the strict lower triangle, and the strict upper triangle, respectively, then the matrix splittings of the Jacobi and Gauss–Seidel iterations are given by

$$M_{\text{Jac}} = D, \qquad S_{\text{Jac}} = -(L+U), \tag{11}$$

$$\text{and} \qquad M_{\text{GS}} = D + L, \qquad S_{\text{GS}} = -U, \tag{12}$$

where the subscripts are used to identify the two methods.

A discussion of the convergence behavior of relaxation methods can be found in almost any introductory textbook on iterative methods. Typically, the analysis is formulated in terms of the spectral radius of the so-called iteration matrix, $M^{-1}S$. The spectral radius of a matrix B is defined by

$$\rho(B) := \max_{\lambda \text{ is eigenvalue of } B} |\lambda|.$$

The following theorem whose proof can be found in the book by Golub and van Loan[9] summarizes an important result.

Theorem 3.1. *An iterative scheme of the form*

$$M\mathbf{x}_n = S\mathbf{x}_{n-1} + \mathbf{b}$$

for the solution of $A\mathbf{x} = \mathbf{b}$ *with nonsingular coefficient matrix* $A = M - S$ *converges to the exact solution* $\mathbf{x}_* = A^{-1}\mathbf{b}$ *for any starting vector* $\mathbf{x}_0$*, if* M *is nonsingular and*

$$\rho(M^{-1}S) < 1.$$

Relaxation methods are not considered to be really efficient for solving large-scale problems. However, they are still in use as building blocks of multigrid methods or to construct preconditioners.

4 Projection Methods

A general framework to discuss iterative techniques for the solution of linear systems is a projection process. The idea of a projection method is to extract the next approximations $\mathbf{x}_n$ from a search subspace $\mathcal{K}$. If the dimension of $\mathcal{K}$ is given by m, then, in general, m restrictions or constraints are necessary to be able to extract $\mathbf{x}_n$ from $\mathcal{K}$. Typically, the constraints are imposed by orthogonalizing the residual vector $\mathbf{r}_n$ with respect to a subspace of constraints $\mathcal{L}$.

To illustrate the situation, let there be two subspaces, $\mathcal{K}$ and $\mathcal{L}$, of dimension m with two sets of basis vectors,

$$\mathcal{K} = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\} \qquad \text{and} \qquad \mathcal{L} = \operatorname{span}\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m\}.$$

Given some current approximation $\mathbf{x}_{n-1}$, the next approximation $\mathbf{x}_n$ is constructed in the search subspace, i.e.,

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mathcal{K} \tag{13}$$

subject to

$$\mathbf{r}_n \perp \mathcal{L}. \tag{14}$$

To proceed with the discussion we introduce two $N \times m$ matrices whose columns are given by the basis vectors of $\mathcal{K}$ and $\mathcal{L}$,

$$V_m = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_m] \qquad \text{and} \qquad W_m = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_m]\,,$$

respectively. Then, an equivalent form of the next approximation is given by

$$\mathbf{x}_n = \mathbf{x}_{n-1} + V_m\mathbf{y}_m, \tag{15}$$

$\mathbf{x}_n$ = PROJECTION(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{x}_0$ is a starting vector, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$.

```
1: Choose $\mathbf{x}_0 \in \mathbb{C}^N$
2: **for** $n = 1, 2, 3, \ldots$ **do** $\{$until convergence$\}$
3:     Choose subspaces $\mathcal{K}$ and $\mathcal{L}$
4:     Choose basis $V_m = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_m]$ of $\mathcal{K}$ and
       choose basis $W_m = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_m]$ of $\mathcal{L}$
5:     $\mathbf{r}_{n-1} \leftarrow \mathbf{b} - A\mathbf{x}_{n-1}$
6:     $\mathbf{y}_m \leftarrow (W_m^H A V_m)^{-1} W_m^H \mathbf{r}_{n-1}$
7:     $\mathbf{x}_n \leftarrow \mathbf{x}_{n-1} + V_m \mathbf{y}_m$
8: **end for**
```

Figure 2. General form of a projection method.

where $\mathbf{y}_m \in \mathbb{C}^m$ is determined by the constraint (14). More precisely, the governing equation of $\mathbf{y}_m$ follows from the reformulation of (14) in the form of

$$W_m^H \mathbf{r}_n = \mathbf{0}.$$

By inserting the residual vector corresponding to (15), an equivalent form is given by

$$W_m^H \mathbf{r}_{n-1} = W_m^H A V_m \mathbf{y}_m$$

which finally leads to

$$\mathbf{y}_m = \left(W_m^H A V_m\right)^{-1} W_m^H \mathbf{r}_{n-1}.$$

The preceding derivation leads to the general form of a projection method depicted in Fig. 2. Note that the algorithm still depends on the choice of $\mathcal{K}$, $\mathcal{L}$, and their bases.

In general, a projection method onto the search subspace $\mathcal{K}$ orthogonal to the subspace of constraints $\mathcal{L}$ is characterized by (13) and (14). If the search subspace $\mathcal{K}$ is the same as the subspace of constraints $\mathcal{L}$, the process is called an orthogonal projection method. In an oblique projection method, $\mathcal{K}$ is different from $\mathcal{L}$.

Several theoretical results are known under the general scenario of projection methods. For instance, without being specific about the subspaces $\mathcal{K}$ and $\mathcal{L}$, the following result holds if the subspace of constraints is chosen to satisfy $\mathcal{L} = A\mathcal{K}$.

Theorem 4.1 (Optimality of projection method with $\mathcal{L} = A\mathcal{K}$). *A vector $\mathbf{x}_n$ is the next approximation of a projection method onto the search subspace $\mathcal{K}$ along the subspace of constraints $\mathcal{L} = A\mathcal{K}$ if and only if*

$$\|\mathbf{b} - A\mathbf{x}_n\| = \min_{\mathbf{x}\in\mathbf{x}_{n-1}+\mathcal{K}} \|\mathbf{b} - A\mathbf{x}\|. \tag{16}$$

Here and in the sequel, the notation $\|\cdot\|$ is used to denote the Euclidean norm. The proof of the preceding theorem is given in the book by Saad[10] which also contains additional material on general optimality results.

5 Krylov Subspace Methods

In the canonical form of projection methods introduced in the preceding section, the particular choice of a search subspace $\mathcal{K}$ as well as of a subspace of constraints $\mathcal{L}$ is not specified. In this section, we will explicitly describe candidates for both, $\mathcal{K}$ and $\mathcal{L}$.

5.1 The Search Subspace

Krylov subspace methods are currently considered to be among the most powerful iterative methods for the solution of large sparse linear systems. A projection method onto the search subspace

$$\mathcal{K}_n(A, \mathbf{r}_0) := \operatorname{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{n-1}\mathbf{r}_0\}$$

is called a Krylov subspace method. The subspace $\mathcal{K}_n(A, \mathbf{r}_0)$ is referred to as the nth Krylov subspace generated by the matrix A and the vector $\mathbf{r}_0$. Any Krylov subspace method for the solution of $A\mathbf{x} = \mathbf{b}$ is characterized by constructing the vector $\mathbf{x}_n - \mathbf{x}_0$ in subspaces of the specific form $\mathcal{K}_n(A, \mathbf{r}_0)$ where $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$ is the initial residual vector associated with the initial guess $\mathbf{x}_0$. A key feature of any Krylov subspace method is to find accurate approximations $\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(A, \mathbf{r}_0)$ when $n \ll N$.

The straightforward approach to construct a basis of $\mathcal{K}_n(A, \mathbf{r}_0)$ is to repeatedly multiply the starting vector $\mathbf{r}_0$ by the matrix A. The resulting algorithm given in Fig. 3 is known as the power method but is a numerically useless process to span a basis of $\mathcal{K}_n(A, \mathbf{r}_0)$. The reason is that the power method converges to the largest eigenvalue (in absolute value) and, therefore, the vectors will soon become linear dependent in finite-precision arithmetic[11]. To emphasize the importance of a numerically stable process for the generation of the Krylov subspace, we abstract from the particular "application" of solving linear systems and use a general starting vector $\mathbf{v}_1$ in the following discussion.

V_n = POWERMETHOD(A, $\mathbf{v}_1$) If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{v}_1$ is a suitable starting vector, this algorithm computes a (numerically useless) basis $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N\times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$.

```
1: Choose $\mathbf{v}_1 \in \mathbb{C}^N$ such that $\|\mathbf{v}_1\| = 1$
2: for $n = 1, 2, 3, \ldots$ do {until invariance}
3:     $\widetilde{\mathbf{v}}_{n+1} \leftarrow A\mathbf{v}_n$
4:     $\alpha_{n+1} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
5:     $\mathbf{v}_{n+1} \leftarrow \frac{1}{\alpha_{n+1}}\widetilde{\mathbf{v}}_{n+1}$
6: end for
```

Figure 3. Power method.

Arnoldi Algorithm

A more promising approach for the generation of a basis of $\mathcal{K}_n(A, \mathbf{v}_1)$ is the Arnoldi process[12] which computes an orthonormal basis of $\mathcal{K}_n(A, \mathbf{v}_1)$. The process of orthogonalization works as follows. If $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ already is an orthonormal set of basis vectors, then the vector

$$\mathbf{z} = \widetilde{\mathbf{v}}_{n+1} - (\mathbf{v}_1^H \widetilde{\mathbf{v}}_{n+1})\mathbf{v}_1 - (\mathbf{v}_2^H \widetilde{\mathbf{v}}_{n+1})\mathbf{v}_2 - \cdots - (\mathbf{v}_n^H \widetilde{\mathbf{v}}_{n+1})\mathbf{v}_n$$

is orthogonal to $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. This can be derived from multiplication by $\mathbf{v}_i^H$ where $i = 1, 2, \ldots, n$ resulting in

$$\mathbf{v}_i^H \mathbf{z} = \mathbf{v}_i^H \widetilde{\mathbf{v}}_{n+1} - (\mathbf{v}_i^H \widetilde{\mathbf{v}}_{n+1})\mathbf{v}_i^H \mathbf{v}_i = 0$$

due to the orthonormality of the set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. A new set $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n, \mathbf{v}_{n+1}\}$ of orthonormal basis vectors is easily obtained by adding $\mathbf{v}_{n+1}$ as a scaled version of $\mathbf{z}$.

The resulting Arnoldi method is shown in Fig. 4 where the generated vectors $\mathbf{v}_i$ are called Arnoldi vectors. Step n of this algorithm computes the product of the previous Arnoldi vector $\mathbf{v}_n$ and A, i.e., $\widetilde{\mathbf{v}}_{n+1} = A\mathbf{v}_n$, and orthogonalizes $\widetilde{\mathbf{v}}_{n+1}$ towards all previous Arnoldi vectors $\mathbf{v}_i$ with $i = 1, 2, \ldots, n$ by the procedure explained above. Finally, $\widetilde{\mathbf{v}}_{n+1}$ is scaled to unity in the Euclidean norm. This kind of orthonormalization is called the standard Gram–Schmidt process.

The complete Arnoldi process can be written in matrix notation by first combining lines 3, 7 and 9 in the following vector equation

$$A\mathbf{v}_n = h_{n+1,n}\mathbf{v}_{n+1} + \sum_{i=1}^{n} h_{in}\mathbf{v}_i$$

V_n = BasicArnoldi(A, $\mathbf{v}_1$) If $A \in \mathbb{C}^{N \times N}$ and $\mathbf{v}_1$ is a suitable starting vector, this algorithm computes an orthonormal basis $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N \times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$ via the standard Gram–Schmidt process.

```
1: Choose $\mathbf{v}_1 \in \mathbb{C}^N$ such that $\|\mathbf{v}_1\| = 1$
2: for $n = 1, 2, 3, \ldots$ do {until invariance}
3:     $\widetilde{\mathbf{v}}_{n+1} \leftarrow A\mathbf{v}_n$
4:     for $i = 1, 2, \ldots, n$ do
5:         $h_{in} \leftarrow \mathbf{v}_i^H \widetilde{\mathbf{v}}_{n+1}$
6:     end for
7:     $\widetilde{\mathbf{v}}_{n+1} \leftarrow \widetilde{\mathbf{v}}_{n+1} - \sum_{i=1}^{n} h_{in}\mathbf{v}_i$
8:     $h_{n+1,n} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
9:     $\mathbf{v}_{n+1} \leftarrow \frac{1}{h_{n+1,n}}\widetilde{\mathbf{v}}_{n+1}$
10: end for
```

Figure 4. Arnoldi method via standard Gram–Schmidt orthogonalization.

representing an $(n+1)$-term recurrence for the computation of the Arnoldi vectors. Collecting the recurrence coefficients in an upper Hessenberg matrix

$$H_n = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ & \ddots & \ddots & \vdots \\ & & h_{n,n-1} & h_{nn} \end{bmatrix} \in \mathbb{C}^{n\times n},$$

a matrix with zeros below the first subdiagonal, leads to the matrix form summarized in the following result.

Theorem 5.1 (Arnoldi). *In exact arithmetic the Arnoldi vectors* $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ *generated during the course of Fig. 4 form an orthonormal basis*

$$V_n^H V_n = I_n \tag{17}$$

of the Krylov subspace

$$\mathcal{K}_n\left(A, \mathbf{v}_1\right) = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}. \tag{18}$$

Successive Arnoldi vectors are related by

$$AV_n = V_n H_n + h_{n+1,n}\mathbf{v}_{n+1}\mathbf{e}_n^T, \tag{19}$$

and the matrix A *is reduced to (full) upper Hessenberg form*

$$V_n^H A V_n = H_n \tag{20}$$

by means of unitary transformations V_n*.*

The very best of the Arnoldi method is the orthogonality of its basis. Orthogonality is a highly-desired feature from the point of view of numerical stability. Moreover, it can be exploited to the advantage of minimizing the Euclidean norm of the residual in an iterative method referred to as GMRES as is shown in the next subsection. The main disadvantage of the Arnoldi method is that its computation is expensive in terms of both arithmetic operations and storage requirement. The Arnoldi process is based on $(n+1)$-term recurrences as is reflected in line 7 of Fig. 4 where all previous Arnoldi vectors are involved or, equivalently, by the fact that the upper Hessenberg matrix H_n is full. Due to this property the Arnoldi method is said to be based on long recurrences. The nth iteration of the Arnoldi method requires $\Theta(n \cdot N)$ arithmetic operations as well as $\Theta(n \cdot N)$ storage. The most unpleasant feature in practical applications when large sparse matrices are involved is the storage requirement of the Arnoldi process that grows linearly with the iteration number. The use of the Arnoldi process may therefore sometimes be prohibited by its long recurrences.

A more stable formulation of the Arnoldi method results from replacing the standard Gram–Schmidt process by a modified Gram–Schmidt orthogonalization; see Saad[10] for details.

Lanczos Algorithm

Another process for the generation of a basis of $\mathcal{K}_n\left(A, \mathbf{v}_1\right)$ is the Lanczos algorithm[13]. In contrast to the long recurrences of the Arnoldi method, the Lanczos algorithm is based on three-term recurrences. However, the basis is no longer unitary. Furthermore, the Lanczos algorithm not only computes a basis of $\mathcal{K}_n(A, \mathbf{v}_1)$ for some starting vector $\mathbf{v}_1$ but also an additional basis of $\mathcal{K}_n\left(A^H, \mathbf{w}_1\right)$ for a second starting vector $\mathbf{w}_1$. The process is depicted in Fig. 5 and summarized in the following theorem.

$[V_n,\ W_n]$ = BioLanczos(A, $\mathbf{v}_1$, $\mathbf{w}_1$) If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{v}_1, \mathbf{w}_1$ are suitable starting vectors, this algorithm computes biorthogonal bases $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N\times n}$ and $W_n = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_n] \in \mathbb{C}^{N\times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$ and $\mathcal{K}_n\left(A^H, \mathbf{w}_1\right)$, respectively.

```
1:  Choose $\mathbf{v}_1, \mathbf{w}_1 \in \mathbb{C}^N$ such that $\|\mathbf{v}_1\| = \|\mathbf{w}_1\| \leftarrow 1$ and $\delta_1 \leftarrow \mathbf{w}_1^H \mathbf{v}_1 \neq 0$
2:  Set $\mathbf{v}_0 = \mathbf{w}_0 \leftarrow \mathbf{0}$ and $\gamma_1 = \rho_1 \leftarrow 0, \xi_1 \neq 0$
3:  for $n = 1, 2, 3, \ldots$ do $\{$until invariance$\}$
4:      $\alpha_n \leftarrow \mathbf{w}_n^H A \mathbf{v}_n / \delta_n$
5:      $\widetilde{\mathbf{v}}_{n+1} \leftarrow A\mathbf{v}_n - \alpha_n \mathbf{v}_n - \gamma_n \mathbf{v}_{n-1}$
6:      $\widetilde{\mathbf{w}}_{n+1} \leftarrow A^H \mathbf{w}_n - \overline{\alpha}_n \mathbf{w}_n - \frac{\overline{\gamma}_n \overline{\rho}_n}{\xi_n} \mathbf{w}_{n-1}$
7:      $\rho_{n+1} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
8:      $\xi_{n+1} \leftarrow \|\widetilde{\mathbf{w}}_{n+1}\|$
9:      $\mathbf{v}_{n+1} \leftarrow \frac{1}{\rho_{n+1}} \widetilde{\mathbf{v}}_{n+1}$
10:     $\mathbf{w}_{n+1} \leftarrow \frac{1}{\xi_{n+1}} \widetilde{\mathbf{w}}_{n+1}$
11:     $\delta_{n+1} \leftarrow \mathbf{w}_{n+1}^H \mathbf{v}_{n+1}$
12:     $\gamma_{n+1} \leftarrow \overline{\xi}_{n+1} \delta_{n+1} / \delta_n$
13: end for
```

Figure 5. Biorthogonal Lanczos method.

Theorem 5.2 (Lanczos). *In exact arithmetic the Lanczos vectors* $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ *and* $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n$ *generated during the course of Fig. 5 are scaled to unity in the Euclidean norm and form a biorthogonal system, i.e.,*

$$W_n^H V_n = D_n := \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_n), \tag{21}$$

as well as a pair of bases of the Krylov subspaces

$$\mathcal{K}_n(A, \mathbf{v}_1) = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}, \tag{22}$$

$$\mathcal{K}_n\left(A^H, \mathbf{w}_1\right) = \operatorname{span}\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\}. \tag{23}$$

Successive Lanczos vectors are related by

$$AV_n = V_n T_n + \rho_{n+1} \mathbf{v}_{n+1} \mathbf{e}_n^T, \tag{24}$$

$$A^H W_n = W_n D_n^{-H} T_n^H D_n^H + \xi_{n+1} \mathbf{w}_{n+1} \mathbf{e}_n^T, \tag{25}$$

and the matrix A is reduced to tridiagonal form

$$W_n^H A V_n = D_n T_n \tag{26}$$

by means of similarity transformations V_n.

The Lanczos algorithm computes the Lanczos vectors $\mathbf{v}_i$ and $\mathbf{w}_i$ by means of three-term recurrences as is reflected in a tridiagonal matrix T_n rather than a full upper Hessenberg matrix as in the Arnoldi process. Compared to the Arnoldi algorithm, the price to be paid for these short recurrences is the computation of a second sequence $\mathbf{w}_i$ in addition to the sequence $\mathbf{v}_i$. The sequence $\mathbf{w}_i$ is based on repeatedly multiplying vectors by A^H rather than by A as for the sequence $\mathbf{v}_i$. So, each iteration of the Lanczos algorithm needs two matrix-by-vector products and thus is computationally twice as expensive as the Arnoldi method.

A professional implementation of the Lanczos algorithm is based on look-ahead techniques; see Gutknecht[14] and the references given therein.

Additional Remark

While the Arnoldi process corresponds to a Hessenberg orthogonalization, the Lanczos algorithm is summarized as a tridiagonal biorthogonalization. Unfortunately, it is not possible to define algorithms making use of a tridiagonal orthogonalization, i.e., combining optimal projection techniques and short recurrences[15].

5.2 The Subspace of Constraints

A Krylov subspace method is a projection method onto the Krylov subspace $\mathcal{K}_n(A, \mathbf{r}_0)$ along a subspace of constraints $\mathcal{L}_n$. More formally, the iterates of a Krylov subspace method are of the form

$$\mathbf{x}_n \in \mathbf{x}_0 + \mathcal{K}_n(A, \mathbf{r}_0) \tag{27}$$

subject to

$$\mathbf{r}_n \perp \mathcal{L}_n. \tag{28}$$

That is, the iterates are constructed in the particular search space $\mathcal{K}_n(A, \mathbf{r}_0)$ while their actual definition is based on the restriction that the associated residual vector

$$\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n \tag{29}$$

is orthogonal to a subspace $\mathcal{L}_n$. In this subsection, we will discuss different candidates for this subspace of constraints. The Arnoldi and Lanczos methods started with

$$\mathbf{v}_1 = \frac{\mathbf{r}_0}{\|\mathbf{r}_0\|} \tag{30}$$

will then be used as underlying processes of different Krylov subspace methods derived by applying these different approaches for the choice of $\mathcal{L}_n$.

Using (15), an equivalent representation of the iterates (27) is given by

$$\mathbf{x}_n = \mathbf{x}_0 + V_n\mathbf{y} \quad \text{for some} \quad \mathbf{y} \in \mathbb{C}^n. \tag{31}$$

Here, $\mathbf{y}$ is a free parameter vector that is fixed by imposing the condition (28). Note from (29) that, since the coefficient matrix is nonsingular, there is a bijection between $\mathbf{x}_n$

and $\mathbf{r}_n$. Thus, imposing a condition on $\mathbf{r}_n$ corresponds to fixing the free parameter vector $\mathbf{y}$ in the representation (31) of $\mathbf{x}_n$. Let $\mathbf{y}_n \in \mathbb{C}^n$ denote the corresponding vector determining the actual definition of the iterates, i.e.,

$$\mathbf{x}_n = \mathbf{x}_0 + V_n \mathbf{y}_n. \tag{32}$$

Then, the resulting Krylov subspace methods differ in

- the choice of the basis V_n of the search subspace $\mathcal{K}_n(A, \mathbf{r}_0)$ and
- the definition of $\mathbf{y}_n$ by the choice of the subspace of constraints $\mathcal{L}_n$.

An interpretation is as follows. A designer of a Krylov subspace method has two degrees of freedom. Firstly, there is the choice of how a basis of the underlying Krylov subspaces is generated in a numerically stable way, for instance, the Arnoldi and Lanczos methods may be used. Secondly, there is the option to define the actual iterates by different choices of $\mathbf{y}_n$ four of which are described in this subsection.

Fixing the free parameter vector $\mathbf{y}$ in (31) by imposing the condition (28) relates the vector $\mathbf{y}_n$ to a basis

$$W_n = [\mathbf{w}_1 \; \mathbf{w}_2 \; \cdots \; \mathbf{w}_n]$$

of the subspace $\mathcal{L}_n$ as follows. Rewriting (28) in terms of the basis W_n of $\mathcal{L}_n$ yields

$$W_n^H \mathbf{r}_n = \mathbf{0}.$$

Inserting the residual vector associated to the iterate (32) which is given by

$$\mathbf{r}_n = \mathbf{r}_0 - A V_n \mathbf{y}_n$$

results in

$$W_n^H A V_n \mathbf{y}_n = W_n^H \mathbf{r}_0. \tag{33}$$

In the following paragraphs, different bases W_n of $\mathcal{L}_n$ are used in this equation leading to different vectors $\mathbf{y}_n$.

The Ritz–Galerkin Approach

There are two broad classes of projection methods. The classification is based on whether or not the search subspace $\mathcal{K}_n$ is the same as the subspace of constraints $\mathcal{L}_n$. Recall that, in an orthogonal projection method, both subspaces are equal while they are different in an oblique projection method. An orthogonal Krylov subspace method takes

$$\mathcal{K}_n = \mathcal{L}_n = \mathcal{K}_n(A, \mathbf{r}_0).$$

This approach is called *Ritz–Galerkin approach*. To derive an orthogonal Krylov subspace method this approach is applied to an underlying process for the generation of $\mathcal{K}_n(A, \mathbf{r}_0)$. When applying the Ritz–Galerkin approach to a process like the Arnoldi method there is no need for the generation of a second basis W_n because the two subspaces $\mathcal{K}_n$ and $\mathcal{L}_n$ are the same and the underlying process already generates a basis V_n of the Krylov subspace $\mathcal{K}_n(A, \mathbf{r}_0)$. Thus, inserting $W_n = V_n$ in (33) leads to

$$V_n^H A V_n \mathbf{y}_n = V_n^H \mathbf{r}_0.$$

$\mathbf{x}_n$ = FOM(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

```
1: $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$
2: $\mathbf{v}_1 \leftarrow \mathbf{r}_0/\|\mathbf{r}_0\|$
3: for $n = 1, 2, 3, \ldots$ do {until convergence}
4:     Step $n$ of Arnoldi method producing $H_n$ and $V_n$
5:     $\mathbf{y}_n \leftarrow H_n^{-1}\|\mathbf{r}_0\|\mathbf{e}_1$
6:     $\mathbf{x}_n \leftarrow \mathbf{x}_0 + V_n\mathbf{y}_n$
7: end for
```

Figure 6. Highlevel Full Orthogonalization Method (FOM).

Assume that the Ritz–Galerkin approach is applied to the Arnoldi method started with (30). Then, using Theorem 5.1 results in

$$H_n\mathbf{y}_n = \|\mathbf{r}_0\|\mathbf{e}_1, \tag{34}$$

where $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^T$ is the first Cartesian unit vector of appropriate dimension. That is, from a conceptual point of view, the vector $\mathbf{y}_n$ defining the iterate $\mathbf{x}_n$ is available by solving a linear system whose coefficient matrix is given by the upper Hessenberg matrix H_n generated by the Arnoldi process. Note that a Hessenberg system of type (34) is to be solved in each iteration, n, of the resulting process for the solution of the original system $A\mathbf{x} = \mathbf{b}$. Fortunately, the system (34) is a "small" $n \times n$ system as opposed to the original large $N \times N$ system. The actual implementation of the resulting method called Full Orthogonalization Method (FOM) is beyond the scope of this note. A highlevel description of FOM is given in Fig. 6. There are many possible variations including those dealing with practical issues such as reducing the high computational and memory cost incurred by the underlying long recurrences of the Arnoldi process; see Saad[10] for more details on restarting FOM and on truncation of the underlying orthogonalization process.

The Petrov–Galerkin Approach

In contrast to an orthogonal projection method where the search subspace $\mathcal{K}_n$ and the subspace of constraints $\mathcal{L}_n$ are the same, $\mathcal{K}_n$ and $\mathcal{L}_n$ are different in an oblique projection method. An oblique Krylov subspace method can be derived by taking

$$\mathcal{L}_n = \mathcal{K}_n\left(A^H, \mathbf{w}_1\right),$$

where $\mathbf{w}_1 \in \mathbb{C}^N$ is a nonzero starting vector for the generation of a basis of $\mathcal{L}_n$. An approach where $\mathcal{K}_n$ and $\mathcal{L}_n$ are different is called *Petrov–Galerkin approach*. Recall that the Lanczos algorithm not only generates a basis $\mathbf{V}_n$ of $\mathcal{K}_n(A, \mathbf{r}_0)$ but also computes another basis $\mathbf{W}_n$ of $\mathcal{K}_n\left(A^H, \mathbf{w}_1\right)$. Thus, a basis of $\mathcal{L}_n$ is available by means of the Lanczos process suggesting the application of the Petrov–Galerkin approach to the Lanczos algorithm. To this end, start the Lanczos algorithm with two starting vectors $\mathbf{v}_1$ as given by (30)

$\mathbf{x}_n$ = BI-LANCZOS(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

```
1: $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$
2: $\mathbf{v}_1 \leftarrow \mathbf{r}_0/\|\mathbf{r}_0\|$
3: Choose $\mathbf{w}_1$ such that $\|\mathbf{w}_1\| \leftarrow 1$ and $\mathbf{v}_1^H\mathbf{w}_1 \neq 0$
4: for $n = 1, 2, 3, \ldots$ do {until convergence}
5:     Step $n$ of biorthogonal Lanczos algorithm producing $T_n$ and $V_n$
6:     $\mathbf{y}_n \leftarrow T_n^{-1}\|\mathbf{r}_0\|\mathbf{e}_1$
7:     $\mathbf{x}_n \leftarrow \mathbf{x}_0 + V_n\mathbf{y}_n$
8: end for
```

Figure 7. Highlevel Bi-Lanczos Method.

and $\mathbf{w}_1$ satisfying

$$\|\mathbf{w}_1\| = 1 \quad \text{and} \quad \mathbf{v}_1^H\mathbf{w}_1 \neq 0. \tag{35}$$

Then, (33) and Theorem 5.2 lead to

$$D_nT_n\mathbf{y}_n = \delta_1\|\mathbf{r}_0\|\mathbf{e}_1.$$

Since D_n is a nonsingular diagonal matrix an equivalent form is given by

$$T_n\mathbf{y}_n = \|\mathbf{r}_0\|\mathbf{e}_1. \tag{36}$$

This situation is quite similar to the Ritz–Galerkin approach described in the previous section in the sense that, in every iteration n, a small $n \times n$ systems has to be solved in order to obtain the vector $\mathbf{y}_n$. Here, however, the coefficient matrix T_n generated by the Lanczos process is tridiagonal whereas the coefficient matrix in (34) is a (full) upper Hessenberg matrix. The resulting method is called Bi-Lanczos method[16] and its highlevel description is depicted in Fig. 7.

The Minimum Residual Approach

In Theorem 4.1, an optimality result for oblique projection methods is given under quite general assumptions. In particular, without being specific about the subspaces $\mathcal{K}_n$ and $\mathcal{L}_n$, an oblique projection method minimizes the Euclidean norm of the residual vector over the space $\mathbf{x}_0 + \mathcal{K}_n$ if and only if the subspace of constraints is defined by $\mathcal{L}_n = A\mathcal{K}_n$. This result is applied here to Krylov subspace methods where $\mathcal{K}_n = \mathcal{K}_n(A, \mathbf{r}_0)$.

Suppose that, in a Krylov subspace method, the subspace of constraints is chosen as

$$\mathcal{L}_n = A\mathcal{K}_n(A, \mathbf{r}_0).$$

Then according to Theorem 4.1, the iterate $\mathbf{x}_n$ is given by the vector whose associated residual is minimal in the Euclidean norm over all vectors $\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_n(A, \mathbf{r}_0)$, i.e.,

$$\|\mathbf{b} - A\mathbf{x}_n\| = \min_{\mathbf{x}\in\mathbf{x}_0+\mathcal{K}_n(A,\mathbf{r}_0)} \|\mathbf{b} - A\mathbf{x}\|. \tag{37}$$

The approach is known as the *minimum residual approach* and can be applied to the Arnoldi process as follows. Recall that every vector $\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_n(A, \mathbf{r}_0)$ can be represented in the form (31) involving the free parameter vector $\mathbf{y}$. Therefore, rather than using $\mathbf{x}$, the minimization problem (37) can be reformulated in terms of the free parameter vector $\mathbf{y}$. That is, solving the minimization problem (37) in iteration n implies fixing the vector $\mathbf{y}$ to a particular vector $\mathbf{y}_n$ or, equivalently, choosing the subspace $\mathcal{L}_n$. More precisely, if the Arnoldi process is started with $\mathbf{v}_1$ as in (30), the residual vector associated to (31) is given by

$$\mathbf{r}_n = \|\mathbf{r}_0\|\mathbf{v}_1 - AV_n\mathbf{y} \quad \text{for some} \quad \mathbf{y} \in \mathbb{C}^n.$$

Using the fact that $\mathbf{v}_1$ is the first column of V_{n+1} and making use of Theorem 5.1 yields

$$\mathbf{r}_n = V_{n+1}(\|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y}) \quad \text{for some} \quad \mathbf{y} \in \mathbb{C}^n, \tag{38}$$

where the symbol

$$\underline{H}_{n+1} := \begin{bmatrix} & H_n & \\ 0\ 0\ \dots\ 0 & h_{n+1,n} \end{bmatrix} \in \mathbb{C}^{(n+1)\times n}$$

denotes the matrix obtained from H_n in (19) by adding another row at the bottom. This representation is used to rewrite (37) in the form

$$\|V_{n+1}(\|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y}_n)\| = \min_{\mathbf{y}\in\mathbb{C}^n} \|V_{n+1}(\|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y})\|.$$

Since the Arnoldi process generates a unitary matrix V_{n+1} and the Euclidean norm is invariant under unitary transformations, an equivalent form is given by

$$\|V_{n+1}(\|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y}_n)\| = \min_{\mathbf{y}\in\mathbb{C}^n} \|\ \|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y}\|. \tag{39}$$

Thus, in each iteration n of the resulting method, an $(n+1)\times n$ least-squares problem of type (39) is to be solved. Since the Hessenberg matrix $\underline{H}_{n+1}$ has full rank, there is a unique solution $\mathbf{y}_n$. This method is known as the Generalized Minimum RESidual method (GMRES)[17] whose highlevel description is given in Fig. 8. Finally, note that (39) is actually not a single least-squares problem, but defines a sequence of least-squares problems where in each step a row and a column in $\underline{H}_{n+1}$ are appended. Moreover, its Hessenberg structure can be exploited to efficiently solve the sequence of least-squares problems with less computational cost than solving a new least-squares problem in every iteration from scratch. The corresponding technique is detailed in Saad[10].

The Quasi-Minimal Residual Approach

Suppose the Lanczos algorithm is started with two nonzero starting vectors $\mathbf{v}_1$ and $\mathbf{w}_1$ satisfying (30) and (35), respectively. Then, similar to the derivation of (38), the Lanczos algorithm summarized by Theorem 5.2 leads to

$$\mathbf{r}_n = V_{n+1}(\|\mathbf{r}_0\|\mathbf{e}_1 - \underline{T}_{n+1}\mathbf{y}) \quad \text{for some} \quad \mathbf{y} \in \mathbb{C}^n, \tag{40}$$

where

$$\underline{T}_{n+1} := \begin{bmatrix} & T_n & \\ 0\ 0\ \dots\ 0 & \rho_{n+1} \end{bmatrix} \in \mathbb{C}^{(n+1)\times n}.$$

$\mathbf{x}_n$ = GMRES(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

```
1: $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$
2: $\mathbf{v}_1 \leftarrow \mathbf{r}_0/\|\mathbf{r}_0\|$
3: **for** $n = 1, 2, 3, \ldots$ **do** $\{$until convergence$\}$
4:     Step $n$ of Arnoldi method producing $\underline{H}_{n+1}$ and $V_n$
5:     Compute minimizer $\mathbf{y}_n$ of $\|\ \|\mathbf{r}_0\|\mathbf{e}_1 - \underline{H}_{n+1}\mathbf{y}\|$
6:     $\mathbf{x}_n \leftarrow \mathbf{x}_0 + V_n\mathbf{y}_n$
7: **end for**
```

Figure 8. Highlevel Generalized Minimum Residual Method (GMRES).

Since the matrix V_{n+1} is no longer unitary in the Lanczos process, it is not possible to minimize $\|\mathbf{r}_n\|$ by an equivalent "small" problem of type (39) in a similar way as in the minimum residual approach. Rather than minimizing $\|\mathbf{r}_n\|$ which would be desirable but computationally expensive, the *quasi-minimal residual approach* minimizes a factor of the representation (40) of the residual. More precisely, the free parameter vector $\mathbf{y}$ is fixed by

$$\|\ \|\mathbf{r}_0\|\mathbf{e}_1 - \underline{T}_{n+1}\mathbf{y}_n\| = \min_{\mathbf{y}\in\mathbb{C}^n} \|\ \|\mathbf{r}_0\|\mathbf{e}_1 - \underline{T}_{n+1}\mathbf{y}\|.$$

So, instead of $\|\mathbf{r}_n\|$, only the norm of the factor of (40) given in parentheses is minimized. The complete highlevel description of the resulting Quasi-Minimal Residual method (QMR)[18] is given in Fig. 9.

$\mathbf{x}_n$ = QMR(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

```
1: $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$
2: $\mathbf{v}_1 \leftarrow \mathbf{r}_0/\|\mathbf{r}_0\|$
3: Choose $\mathbf{w}_1$ such that $\|\mathbf{w}_1\| \leftarrow 1$ and $\mathbf{v}_1^H\mathbf{w}_1 \neq 0$
4: **for** $n = 1, 2, 3, \ldots$ **do** $\{$until convergence$\}$
5:     Step $n$ of biorthogonal Lanczos algorithm producing $\underline{T}_{n+1}$ and $V_n$
6:     Compute minimizer $\mathbf{y}_n$ of $\|\ \|\mathbf{r}_0\|\mathbf{e}_1 - \underline{T}_{n+1}\mathbf{y}\|$
7:     $\mathbf{x}_n \leftarrow \mathbf{x}_0 + V_n\mathbf{y}_n$
8: **end for**
```

Figure 9. Highlevel Quasi-Minimal Residual Method (QMR).

Additional Remark

The presentations of the algorithms given in this survey concentrate on the underlying principles of Krylov subspace methods. These principles are useful in understanding most of the other Krylov subspace methods. The highlevel presentations are not meant to replace the study of the original papers such as the ones by Hestenes and Stiefel[19] for CG, Saad and Schulz[17] for GMRES, Freund and Nachtigal[18] for QMR, to name just a few. Important implementation details given in these articles are vast and indispensable for efficient and professional software. In particular, a statement of the form $\mathbf{x}_n \leftarrow \mathbf{x}_0 + V_n\mathbf{y}_n$ in the highlevel presentations does *not* necessarily mean to store the complete matrix V_n, i.e., all vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$.

6 Preconditioning

The convergence of an iterative method applied to a linear system depends on the properties of the coefficient matrix. With the exception of GMRES, little is known about the details of the convergence behavior for general linear systems. To achieve or accelerate the convergence of an iterative method, a given linear system $A\mathbf{x} = \mathbf{b}$ is often transformed into an equivalent system called preconditioned system. There are right preconditioning techniques of the form

$$AM\mathbf{y} = \mathbf{b} \qquad \text{and} \qquad M\mathbf{y} = \mathbf{x}$$

and left preconditioners of type

$$MA\mathbf{x} = M\mathbf{b}.$$

Convergence will be fast if AM or MA are, in some sense, "close to" the identity I for right and left preconditioning, respectively. Preconditioning involves the additional work of computing the preconditioner M and the repeated solution of linear systems with M as the coefficient matrix. The requirements for an effective preconditioner are as follows:

- Linear systems with coefficient matrix M should be easy to solve. An extreme case is when $M = I$, but then convergence is not accelerated at all; essentially, M is no proper preconditioner.

- The acceleration of the convergence should be fast. The extreme case here is when $M = A^{-1}$ in which the process converges in a single step but the construction of M is as hard as solving the original system.

Effective preconditioners lie between these two extremes. Some examples of preconditioning techniques are as follows.

Diagonal scaling is a simple preconditioner where $M = \operatorname{diag}^{-1}(A)$. Another popular preconditioning class is based on incomplete Cholesky or LU factorizations. Numerous variants of ILU preconditioners are in use. In problems arising from partial differential equations, preconditioners are constructed from coarse-grid approximations. One of the key ideas of multigrid methods is to use, as a preconditioner, one or more steps of a classical iteration such as Jacobi or Gauss–Seidel. Another well-known strategy is based on domain decomposition techniques where the idea is to have solvers for certain local subdomains used to form a preconditioner for the overall global problem. Here, the local subdomains can be handled in parallel. Polynomial preconditioners are also interesting with respect to parallelism. Approximate inverse preconditioners may also offer a high degree of parallelism. Preconditioning is described in the books by Saad[10], Greenbaum[20], Axelsson[21], and Meurant[22].

7 Reducing Synchronization

The parallelization of Krylov subspace methods on distributed memory processors is straightforward[23] and consists in parallelizing the three kinds of operations: vector updates, matrix-vector products, and inner products. Vector updates are perfectly parallelizable and, for large sparse matrices, matrix-vector products can be implemented with communication between only nearby processors. The bottleneck is usually due to inner products enforcing global communication, i.e., communication of all processors at the same time.

There are two strategies to remedy the performance degradation which, of course, can be combined. The first is to restructure the code such that most communication is overlapped with useful computation. The second is to eliminate data dependencies such that several inner products can be computed simultaneously. In this section an example of the latter strategy is given where the number of global synchronization points is reduced. A global synchronization point is defined as the locus of an algorithm at which all local information has to be globally available in order to continue the computation.

Consider once more the Lanczos process depicted in Fig. 5. In a parallel implementation of the main loop, global communication is necessary for the inner products in lines 4 and 11 as well as for the norms in lines 7 and 8. Because the computation of the norms can be computed simultaneously, there are three global synchronization points per iteration.

For this algorithm, a simple reorganization of the statements is used to eliminate two of these global synchronization points. The idea is to delay the computation of the vectors $\mathbf{v}_{n+1}$ and $\mathbf{w}_{n+1}$. This is easily accomplished by observing that lines 9 and 10 lead to an equivalent form of line 11 given by

$$\delta_{n+1} = \mathbf{w}_{n+1}^H \mathbf{v}_{n+1} = \frac{\widetilde{\mathbf{w}}_{n+1}^H \widetilde{\mathbf{v}}_{n+1}}{\xi_{n+1}\rho_{n+1}}. \tag{41}$$

Similarly, line 4 is reformulated as

$$\alpha_n = \frac{\widetilde{\mathbf{w}}_n^H A \widetilde{\mathbf{v}}_n}{\xi_n \rho_n \delta_n}.$$

Then, a new variant of the algorithm is given by replacing all δ_n's by a new quantity

$$\widetilde{\delta}_{n+1} := \widetilde{\mathbf{w}}_{n+1}^H \widetilde{\mathbf{v}}_{n+1} = \delta_{n+1}\xi_{n+1}\rho_{n+1}$$

where the last equation results from (41). This parallel variant is depicted in Fig. 10. It is more scalable than the original algorithm of Fig. 5 because the computations of the two inner products in lines 4 and 12 and the two norms in lines 10 and 11 are all independent of each other and can be computed simultaneously. Thus, in this variant of the algorithm, there is only a single global synchronization point per iteration.

Reducing synchronization cost by a simple rearrangement of the statements is possible for the Lanczos algorithm based on three-term recurrences with the option to scale both Lanczos vectors, i.e., the algorithm given in Fig. 5. However, there is a corresponding

$[\, V_n, W_n \,]$ = ScaBioLanczos(A, $\widetilde{\mathbf{v}}_1$, $\widetilde{\mathbf{w}}_1$) If $A \in \mathbb{C}^{N\times N}$ and $\widetilde{\mathbf{v}}_1, \widetilde{\mathbf{w}}_1$ are suitable starting vectors, this algorithm computes biorthogonal bases $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N\times n}$ and $W_n = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_n] \in \mathbb{C}^{N\times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$ and $\mathcal{K}_n(A^H, \mathbf{w}_1)$, respectively. A single synchronization point is used.

```
1: Choose $\widetilde{\mathbf{v}}_1, \widetilde{\mathbf{w}}_1 \in \mathbb{C}^N$ such that $\widetilde{\delta}_1 \leftarrow \widetilde{\mathbf{w}}_1^H \widetilde{\mathbf{v}}_1 \neq 0$
2: Set $\mathbf{v}_0 = \mathbf{w}_0 \leftarrow \mathbf{0}$ and $\rho_0 = \xi_0 \leftarrow 0$, $\rho_1 \leftarrow \|\widetilde{\mathbf{v}}_1\|$, $\xi_1 \leftarrow \|\widetilde{\mathbf{w}}_1\|$, $\widetilde{\delta}_0 \neq 0$
3: for $n = 1, 2, 3, \ldots$ do {until invariance}
4:     $\alpha_n \leftarrow \widetilde{\mathbf{w}}_n^H A \widetilde{\mathbf{v}}_n / \widetilde{\delta}_n$
5:     $\gamma_n \leftarrow \overline{\xi}_n \xi_{n-1} \rho_{n-1} \widetilde{\delta}_n / (\xi_n \rho_n \widetilde{\delta}_{n-1})$
6:     $\mathbf{v}_n \leftarrow \frac{1}{\rho_n} \widetilde{\mathbf{v}}_n$
7:     $\mathbf{w}_n \leftarrow \frac{1}{\xi_n} \widetilde{\mathbf{w}}_n$
8:     $\widetilde{\mathbf{v}}_{n+1} \leftarrow \frac{1}{\rho_n} A \widetilde{\mathbf{v}}_n - \alpha_n \mathbf{v}_n - \gamma_n \mathbf{v}_{n-1}$
9:     $\widetilde{\mathbf{w}}_{n+1} \leftarrow A^H \mathbf{w}_n - \overline{\alpha}_n \mathbf{w}_n - \frac{\overline{\gamma}_n \overline{\rho}_n}{\xi_n} \mathbf{w}_{n-1}$
10:    $\rho_{n+1} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
11:    $\xi_{n+1} \leftarrow \|\widetilde{\mathbf{w}}_{n+1}\|$
12:    $\widetilde{\delta}_{n+1} \leftarrow \widetilde{\mathbf{w}}_{n+1}^H \widetilde{\mathbf{v}}_{n+1}$
13: end for
```

Figure 10. A scalable variant of the Bi-Lanczos method.

coupled two-term formulation[24] of the Lanczos algorithm that has a better reputation with respect to numerical stability where such a rearrangement is not immediately available. Here, algorithms have to be redesigned with parallelism in mind[25,26]. Synchronization is also reduced in algorithms different from the Lanczos algorithm[27–29]. It is possible to combine global synchronization points not only within a single iteration but also within multiple iterations[30,31]. The performance of Krylov subspace methods on parallel computers is modeled by de Sturler[32], Gupta et al.[33], and Bücker[34].

8 Matrix-Vector Multiplications and Graph Partitioning

Among the basic computational kernels of a Krylov subspace method, the most computationally expensive operation is typically the matrix-vector multiplication. A careful implementation of this operation is therefore important. On a parallel computer with distributed memory, a distribution of the data to processors has to be carried out where it is desirable to balance the computational load on each processor while minimizing the interprocessor communication. This can be modeled by a graph partitioning problem as follows.

Consider a matrix-vector multiplication of the form $\mathbf{y} = A\mathbf{x}$ where the N-dimensional vector $\mathbf{y}$ is the result of applying the $N \times N$ coefficient matrix A to some given N-dimensional vector $\mathbf{x}$. Assume that the sparsity is exploited by computing the ith entry of $\mathbf{y}$

$$\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & \times & \times & & \times & \times & & & & & \\
2 & \times & \times & \times & & & & & & & \times \\
3 & & \times & \times & & & \times & & & \times & \\
4 & \times & & & \times & \times & \times & & & & \\
5 & \times & & & \times & \times & & \times & & & \\
6 & & & \times & \times & & \times & \times & & & \\
7 & & & & & \times & \times & \times & \times & & \\
8 & & & & & & & \times & \times & \times & \times \\
9 & & & \times & & & & & \times & \times & \times \\
10 & & \times & & & & & & \times & \times & \times
\end{array}$$

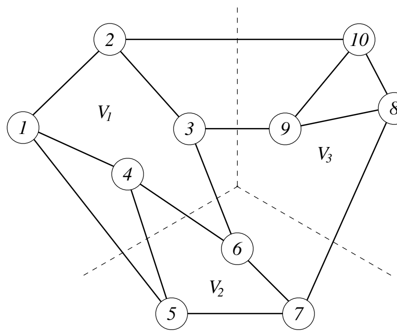

Figure 11. A nonsymmetric 10×10 matrix with a symmetric nonzero pattern (left) and its associated graph with partitions V_1, V_2 and V_3 (right).

via

$$y_i = \sum_{j \text{ with } A(i,j) \neq 0} A(i,j) \cdot x_j, \tag{42}$$

where the summation is over the nonzero elements of the ith row of A.

A graph representation of a nonsymmetric matrix with symmetric nonzero pattern is given by a set of nodes $V = \{1, 2, \ldots, N\}$ where a node is associated to every row of A and a set of edges

$$E = \{(i,j) \mid A(i,j) \neq 0 \text{ for } i \neq j\}$$

used to describe the nonzero entries. An example of a matrix and its associated graph is given in Fig. 11 for $N = 10$. A data distribution on p processors, where x_i, y_i, and the ith row of A are stored on the same processor for all $1 \leq i \leq N$, may be expressed by a partition $P : V \rightarrow \{1, 2, \ldots, p\}$ which decomposes the set of nodes into p subsets $V = V_1 \cup V_2 \cup \cdots \cup V_p$ with $V_i \cap V_j = \emptyset$ for $i \neq j$. In Fig. 11, the partition of the graph on $p = 3$ processors is shown by three dashed lines. Here, the subset describing the data stored on processor 1 is given by $V_1 = \{1, 2, 3, 4\}$. The data distribution of the remaining processors is represented by $V_2 = \{5, 6, 7\}$ and $V_3 = \{8, 9, 10\}$.

A reformulation of (42) in terms of graph terminology is

$$y_i = A(i,i) \cdot x_i + \sum_{\substack{(i,j) \in E \\ P(i) = P(j)}} A(i,j) \cdot x_j + \sum_{\substack{(i,j) \in E \\ P(i) \neq P(j)}} A(i,j) \cdot x_j .$$

Here, the first two terms can be computed on processor i without communication to any other processor. The condition $P(i) \neq P(j)$ in the last term shows that its computation requires communication between processor i and processor j.

Suppose that the number of nonzeros is roughly the same for each row of A. Then, the number of arithmetic operations of a matrix-vector multiplication $A\mathrm{x}$ is well-balanced among the p processors if the partition P satisfies

$$|V_1| \approx |V_2| \approx \cdots \approx |V_p|.$$

Moreover, a rough measure of minimizing interprocessor communication is to find a partition P that minimizes the number of edges (i, j) with $P(i) \neq P(j)$, i.e., those edges connecting nodes in different subsets of the partitions. The number of edges whose end nodes have been mapped to different processors is called the cut size.

Finding a partition balancing the number of nodes while minimizing its cut size is known as the graph partitioning problem. This problem is a hard combinatorial problem known as NP-hard[3]. Thus, it is unlikely that there is a deterministic polynomial-time algorithm that always finds an optimal partition. Since the graph partitioning problem—or more general formulations involving node and edge weights—is of interest in a variety of areas, a large number of heuristics have been developed; see the surveys by Fjällström[35], Schloegel et al.[36], and Hendrickson and Kolda[37].

9 Concluding Remarks

Large sparse linear systems arise frequently in different areas of scientific computing. Due to their sheer size, parallel computing is often mandatory. With the ever-increasing computational performance and storage capacity of the computer technology, the order of the linear systems also increase at a noticeable speed: What today seems a large system is likely to be considered to be small in a few years. Therefore, the capability of exploiting structure and/or sparsity of the coefficient matrix is a crucial ingredient to any computational technique for the solution of linear systems from real-world applications.

Direct methods such as Gaussian elimination or Cholesky factorization may lead to excessive fill-in for large sparse matrices. By making use of the coefficient matrix solely in the form of matrix-vector multiplication, iterative methods do not suffer from the fill-in problem. Classical iterative methods such as Jacobi or Gauss–Seidel iteration typically do not converge fast enough but are useful as building blocks in more efficient techniques. Krylov subspace methods are currently considered to be among the most powerful iterative techniques. Prominent examples include the conjugate gradient (CG) method and the generalized minimum residual (GMRES) method. Preconditioning is an important mechanism to accelerate the convergence of Krylov subspace methods.

When large sparse systems are iteratively solved on parallel computers, a number of additional issues arise. If the number of processors is large, performance is usually decreased by synchronization involved in the computation of inner product-like operations. The cost of synchronization may sometimes be significantly reduced by small rearrangements of some given parallel implementation of a serial algorithm. However, there is more improvement to be expected if a new algorithm is designed from scratch with parallelism already in mind. Graph partitioning can be used to efficiently perform a matrix-vector multiplication in parallel. Here, the idea behind graph partitioning is to balance the computational work while minimizing interprocessor communication.

10 Bibliographic Comments

Thousands of papers have been written on direct methods for the solution of linear systems. The classic book by Wilkinson[38] is an early reference including a careful study of Gaussian elimination with respect to rounding errors. The more general field of matrix computations is treated in a standard textbook by Golub and van Loan[9]. Direct methods exploiting sparsity are described in the books by George and Liu[39], Duff et al.[40], Osterby and Zlatev[41], and Pissanetzky[42]. The books by Varga[43] and Young[44] started the study of classical iterative methods. Modern iterative methods including preconditioning are described in the books by Fischer[1], Greenbaum[20], Saad[10], Axelsson[21], and Meurant[22]. Iterative methods are surveyed in the papers by Freund et al.[45] and by Gutknecht[14].

Multigrid methods are an important class of modern techniques for the solution of linear systems. They are omitted in this survey simply to limit the discussion. Multigrid methods can be viewed as a combination of an iterative scheme and a preconditioner. A seminal paper in the area of multigrid methods is the one by Brandt[46]. A short introduction to multigrid methods is given in the book by Briggs[47]. Additional material is described in the books by Hackbusch[48], Hackbusch and Trottenberg[49], and Briggs et al.[50].

An excellent starting point to parallel computing in general is the book by Kumar et al.[51]. The two articles by Demmel[52] and Demmel et al.[53] give a survey on parallel numerical algorithms. A more recent survey on parallel techniques for the solution of linear systems, both direct and iterative, is given by Duff and van der Vorst[54]. The paper by Saad[55] concentrates on parallel iterative methods.

Acknowledgments

The author would like to thank the organizing team at Forschungszentrum Jülich as well as the Scientific Programme Committee for carrying out a successful Winter School at Rolduc. I am also indebted to F. Hoßfeld and the whole team at the Central Institute for Applied Mathematics, Forschungszentrum Jülich, for making available a professional environment for parts of my research.

References

1. B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*. Advances in Numerical Mathematics. Wiley and Teubner, Chichester, 1996.
2. J. J. Dongarra, I. S. Duff, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, 1998.
3. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. Freeman, San Francisco, 1979.
4. A. Natanzon, R. Shamir, and R. Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30(4):1067–1079, 2000.
5. Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2000.
6. M. Berz, C. Bischof, G. Corliss, and A. Griewank. *Computational Differentiation: Techniques, Applications, and Tools*. SIAM, Philadelphia, 1996.

7. George Corliss, Christèle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann, editors. *Automatic Differentiation of Algorithms: From Simulation to Optimization*. Springer, 2002. (to appear).
8. A. Griewank and G. Corliss. *Automatic Differentiation of Algorithms*. SIAM, Philadelphia, 1991.
9. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, third edition, 1996.
10. Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, 1996.
11. N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Publications, Philadelphia, PA, 1996.
12. W. E. Arnoldi. The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.
13. C. Lanczos. An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, 1950.
14. M. H. Gutknecht. Lanczos-Type Solvers for Nonsymmetric Linear Systems of Equations. In *Acta Numerica 1997*, pages 271–397. Cambridge University Press, Cambridge, 1997.
15. V. Faber and T. Manteuffel. Necessary and Sufficient Conditions for the Existence of a Conjugate Gradient Method. *SIAM Journal on Numerical Analysis*, 21(2):352–362, 1984.
16. C. Lanczos. Solutions of Systems of Linear Equations by Minimized Iterations. *Journal of Research of the National Bureau of Standards*, 49(1):33–53, 1952.
17. Y. Saad and M. H. Schulz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.
18. R. W. Freund and N. M. Nachtigal. QMR: A Quasi-Minimal Residual Method for Non-Hermitian Linear Systems. *Numerische Mathematik*, 60(3):315–339, 1991.
19. M. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.
20. A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, PA, 1997.
21. O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Cambridge, 1994.
22. G. Meurant. *Computer Solution of Large Linear Systems*, volume 28 of *Studies in Mathematics and Its Applications*. North-Holland, Amsterdam, 1999.
23. Y. Saad. Krylov Subspace Methods on Supercomputers. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1200–1232, 1989.
24. R. W. Freund and N. M. Nachtigal. An Implementation of the QMR Method Based on Coupled Two-Term Recurrences. *SIAM Journal on Scientific Computing*, 15(2):313–337, 1994.
25. H. M. Bücker and M. Sauren. A Parallel Version of the Quasi-Minimal Residual Method Based on Coupled Two-Term Recurrences. In J. Waśniewski, J. Dongarra, K. Madsen, and D. Olesen, editors, *Applied Parallel Computing: Industrial Compu-*

tation and Optimization, Proceedings of the Third International Workshop, PARA '96, Lyngby, Denmark, August 18–21, 1996, volume 1184 of *Lecture Notes in Computer Science*, pages 157–165, Berlin, 1996. Springer.

26. H. M. Bücker and M. Sauren. A Variant of the Biconjugate Gradient Method Suitable for Massively Parallel Computing. In G. Bilardi, A. Ferreira, R. Lüling, and J. Rolim, editors, *Solving Irregularly Structured Problems in Parallel, Proceedings of the Fourth International Symposium, IRREGULAR'97, Paderborn, Germany, June 12–13, 1997*, volume 1253 of *Lecture Notes in Computer Science*, pages 72–79, Berlin, 1997. Springer.
27. E.F. D'Azevedo, V.L. Eijkhout, and C.H. Romine. Lapack Working Note 56: Reducing Communication Costs in the Conjugate Gradient Algorithm on Distributed Memory Multiprocessors. Technical Report CS–93–185, University of Tennessee, Knoxville, 1993.
28. G. Meurant. Multitasking the Conjugate Gradient Method on the CRAY X-MP/48. *Parallel Computing*, 5:267–280, 1987.
29. Y. Saad. Practical Use of Polynomial Preconditionings for the Conjugate Gradient Method. *SIAM Journal on Scientific and Statistical Computing*, 6(4):865–881, 1985.
30. A. T. Chronopoulos and C. W. Gear. s-Step Iterative Methods for Symmetric Linear Systems. *Journal of Computational and Applied Mathematics*, 25:153–168, 1989.
31. A. T. Chronopoulos and C. D. Swanson. Parallel Iterative S-step Methods for Unsymmetric Linear Systems. *Parallel Computing*, 22(5):623–641, 1997.
32. E. de Sturler. A Performance Model for Krylov Subspace Methods on Mesh-Based Parallel Computers. *Parallel Computing*, 22:57–74, 1996.
33. A. Gupta, V. Kumar, and A. Sameh. Performance and Scalability of Preconditioned Conjugate Gradient Methods on Parallel Computers. Technical Report TR 92–64, Department of Computer Science, University of Minnesota, Minneapolis, MN – 55455, November 1992. Revised April 1994.
34. H. M. Bücker. Using the Isoefficiency Concept for the Design of Krylov Subspace Methods. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference on Applied Simulation and Modelling, Marbella, Spain, September 4–7, 2001*, pages 404–411, Anaheim, CA, USA, 2001. ACTA Press.
35. P.-O. Fjällström. Algorithms for graph partitioning: a survey. *Linköping Electronic Articles in Computer and Information Science*, 3(10), 1998.
36. K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high-performance scientific simulations. In J. Dongarra, I. Foster, G. Fox, K. Kennedy, L. Torczon, and A. White, editors, *CRPC Parallel Computing Handbook*. Morgan Kaufmann, to appear.
37. B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(2):1519–1534, 2000.
38. J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.
39. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
40. I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.
41. O. Osterby and Z. Zlatev. *Direct Methods for Sparse Matrices*. Springer, New York, 1983.

42. S. Pissanetzky. *Sparse Matrix Technology*. Academic Press, New York, 1984.
43. R. S. Varga. *Matrix Iterative Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1962.
44. D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971.
45. R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative Solution of Linear Systems. In *Acta Numerica 1992*, pages 1–44. Cambridge University Press, Cambridge, 1992.
46. A. Brandt. Multilevel adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.
47. W. L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1987.
48. W. Hackbusch. *Multigrid Methods and Applications*. Springer, Berlin, 1985.
49. W. Hackbusch and U. Trottenberg. *Multigrid Methods*. Springer, Berlin, 1982.
50. W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, Philadelphia, second edition, 2000.
51. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, 1994.
52. J. W. Demmel. Trading Off Parallelism and Numerical Stability. In M. S. Moonen, G. H. Golub, and B. L. R. De Moor, editors, *Linear Algebra for Large Scale and Real-Time Applications*, volume 232 of *NATO ASI Series E: Applied Sciences*, pages 49–68. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993. Proceedings of the NATO Advanced Study Institute on Linear Algebra for Large Scale and Real-Time Applications, Leuven, Belgium, August 1992.
53. J. W. Demmel, M. T. Heath, and H. A. van der Vorst. Parallel Numerical Linear Algebra. In *Acta Numerica 1993*, pages 111–197. Cambridge University Press, Cambridge, 1993.
54. I. S. Duff and H. A. van der Vorst. Developments and Trends in the Parallel Solution of Linear Systems. *Parallel Computing*, 25(13–14):1931–1970, 1999.
55. Y. Saad. Parallel Iterative Methods for Sparse Linear Systems. In D. Butnariu, Y. Censor, and S. Reich, editors, *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, volume 8 of *Studies in Computational Mathematics*, pages 423–440. Elsevier Science, Amsterdam, The Netherlands, 2001.

There are several libraries to (iteratively) solve large sparse linear equation systems in parallel on a number of CPUs. Our parallel cluster also has attached powerful GPUs, but so far, I did not find any solver that can use CPUs and GPUs in parallel to solve a sparse linear equation system. MAIN QUESTION: Are there actually no solvers which use CPUs and GPUs or could you recommend me one?Â Linear system solvers are generally limited by memory access, so in parallel systems communication becomes the bottleneck. Good scaling can be usually achieved only for very large systems (millions of unknowns), for smaller systems you might be better off with a single-node or single-GPU solver.