CHAPTER

4

BASIC ITERATIVE METHODS

The first iterative methods used for solving large linear systems were based on *relaxation of the coordinates*. Beginning with a given approximate solution, these methods modify the components of the approximation, one or a few at a time and in a certain order, until convergence is reached. Each of these modifications, called relaxation steps, is aimed at annihilating one or a few components of the residual vector. Now, these techniques are rarely used separately. However, when combined with the more efficient methods described in later chapters, they can be quite successful. Moreover, there are a few application areas where variations of these methods are still quite popular.

4.1 JACOBI, GAUSS-SEIDEL, AND SOR

This chapter begins by reviewing the basic iterative methods for solving linear systems. Given an $n \times n$ real matrix A and a real n-vector b, the problem considered is: Find x belonging to $\mathbb{R}^n$ such that

$$Ax = b \tag{4.1}$$

Equation (4.1) is a *linear system*, A is the *coefficient matrix*, b is the *right-hand side* vector, and x is the *vector of unknowns*. Most of the methods covered in this chapter involve passing from one iterate to the next by modifying one or a few components of an approximate vector solution at a time. This is natural since there are simple criteria when modifying a component in order to improve an iterate. One example is to annihilate some component(s) of the residual vector $b - Ax$. The convergence of these methods is rarely guaranteed for all matrices, but a large body of theory exists for the case where the coefficient matrix arises from the finite difference discretization of Elliptic Partial Differential Equations.

We begin with the decomposition

$$A = D - E - F, \tag{4.2}$$

in which D is the diagonal of A, $-E$ its strict lower part, and $-F$ its strict upper part, as illustrated in Figure 4.1. It is always assumed that the diagonal entries of A are all nonzero.

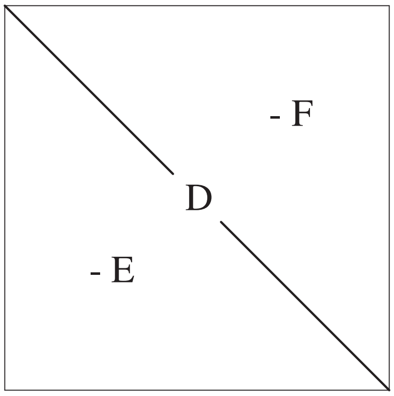


Figure 4.1 *Initial partitioning of matrix A.*

The Jacobi iteration determines the i-th component of the next approximation so as to annihilate the i-th component of the residual vector. In the following, $\xi_i^{(k)}$ denotes the i-th component of the iterate x_k and β_i the i-th component of the right-hand side b. Thus, writing

$$(b - Ax_{k+1})_i = 0, \tag{4.3}$$

in which $(y)_i$ represents the i-th component of the vector y, yields

$$a_{ii}\xi_i^{(k+1)} = -\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}\xi_j^{(k)} + \beta_i,$$

or

$$\xi_i^{(k+1)} = \frac{1}{a_{ii}} \left(\beta_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}\xi_j^{(k)} \right) \quad i = 1, \ldots, n. \tag{4.4}$$

This is a component-wise form of the Jacobi iteration. All components of the next iterate can be grouped into the vector x_{k+1}. The above notation can be used to rewrite the Jacobi iteration (4.4) in vector form as

$$x_{k+1} = D^{-1}(E + F)x_k + D^{-1}b. \tag{4.5}$$

Similarly, the Gauss-Seidel iteration corrects the i-th component of the current approximate solution, in the order $i = 1, 2, \ldots, n$, again to annihilate the i-th component of the residual. However, this time the approximate solution is updated immediately after the new component is determined. The newly computed components $\xi_i^{(k)}$, $i = 1, 2, \ldots, n$ can be changed within a working vector which is redefined at each relaxation step. Thus, since the order is $i = 1, 2, \ldots$, the result at the i-th step is

$$\beta_i - \sum_{j=1}^{i-1} a_{ij}\xi_j^{(k+1)} - a_{ii}\xi_i^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\xi_j^{(k)} = 0, \tag{4.6}$$

which leads to the iteration,

$$\xi_i^{(k+1)} = \frac{1}{a_{ii}}\left(-\sum_{j=1}^{i-1} a_{ij}\xi_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}\xi_j^{(k)} + \beta_i\right),\ i = 1, \ldots, n. \tag{4.7}$$

The defining equation (4.6) can be written as

$$b + Ex_{k+1} - Dx_{k+1} + Fx_k = 0,$$

which leads immediately to the vector form of the Gauss-Seidel iteration

$$x_{k+1} = (D - E)^{-1}Fx_k + (D - E)^{-1}b. \tag{4.8}$$

Computing the new approximation in (4.5) requires multiplying by the inverse of the diagonal matrix D. In (4.8) a triangular system must be solved with $D - E$, the lower triangular part of A. Thus, the new approximation in a Gauss-Seidel step can be determined either by solving a triangular system with the matrix $D - E$ or from the relation (4.7).

A *backward* Gauss-Seidel iteration can also be defined as

$$(D - F)x_{k+1} = Ex_k + b, \tag{4.9}$$

which is equivalent to making the coordinate corrections in the order $n, n-1, \ldots, 1$. A Symmetric Gauss-Seidel Iteration consists of a forward sweep followed by a backward sweep.

The Jacobi and the Gauss-Seidel iterations are both of the form

$$Mx_{k+1} = Nx_k + b = (M - A)x_k + b, \tag{4.10}$$

in which

$$A = M - N \tag{4.11}$$

is a *splitting* of A, with $M = D$ for Jacobi, $M = D - E$ for forward Gauss-Seidel, and $M = D - F$ for backward Gauss-Seidel. An iterative method of the form (4.10) can be defined for any splitting of the form (4.11) where M is nonsingular. *Overrelaxation* is based on the splitting

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega)D),$$

and the corresponding *Successive Over Relaxation* (SOR) method is given by the recursion

$$(D - \omega E)x_{k+1} = [\omega F + (1 - \omega)D]x_k + \omega b. \tag{4.12}$$

The above iteration corresponds to the relaxation sequence

$$\xi_i^{(k+1)} = \omega\xi_i^{GS} + (1 - \omega)\xi_i^{(k)}, i = 1, 2, \ldots, n,$$

in which ξ_i^{GS} is defined by the expression in the right-hand side of (4.7). A backward SOR sweep can be defined analogously to the backward Gauss-Seidel sweep (4.9).

A Symmetric SOR (SSOR) step consists of the SOR step (4.12) followed by a backward SOR step,

$$\begin{aligned}(D-\omega E)x_{k+1/2} &= [\omega F + (1-\omega)D]x_k + \omega b\\ (D-\omega F)x_{k+1} &= [\omega E + (1-\omega)D]x_{k+1/2} + \omega b\end{aligned}$$

This gives the recurrence

$$x_{k+1} = G_\omega x_k + f_\omega,$$

where

$$\begin{aligned}G_\omega &= (D-\omega F)^{-1}(\omega E + (1-\omega)D)\ \times\\ &\qquad\qquad (D-\omega E)^{-1}(\omega F + (1-\omega)D), \qquad (4.13)\\ f_\omega &= \omega(D-\omega F)^{-1}\left(I + [\omega E + (1-\omega)D](D-\omega E)^{-1}\right)\ b. \qquad (4.14)\end{aligned}$$

Observing that

$$\begin{aligned}[\omega E + (1-\omega)D](D-\omega E)^{-1} &= [-(D-\omega E) + (2-\omega)D](D-\omega E)^{-1}\\ &= -I + (2-\omega)D(D-\omega E)^{-1},\end{aligned}$$

f_ω can be rewritten as

$$f_\omega = \omega(2-\omega)\ (D-\omega F)^{-1}D(D-\omega E)^{-1}b.$$

4.1.1 BLOCK RELAXATION SCHEMES

Block relaxation schemes are generalizations of the "point" relaxation schemes described above. They update a whole set of components at each time, typically a subvector of the solution vector, instead of only one component. The matrix A and the right-hand side and solution vectors are partitioned as follows:

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1p}\\ A_{21} & A_{22} & A_{23} & \cdots & A_{2p}\\ A_{31} & A_{32} & A_{33} & \cdots & A_{3p}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ A_{p1} & A_{p2} & \cdots & \cdots & A_{pp}\end{pmatrix},\ x = \begin{pmatrix}\xi_1\\ \xi_2\\ \xi_3\\ \vdots\\ \xi_p\end{pmatrix},\ b = \begin{pmatrix}\beta_1\\ \beta_2\\ \beta_3\\ \vdots\\ \beta_p\end{pmatrix}, \qquad (4.15)$$

in which the partitionings of b and x into subvectors β_i and ξ_i are identical and compatible with the partitioning of A. Thus, for any vector x partitioned as in (4.15),

$$(Ax)_i = \sum_{j=1}^{p} A_{ij}\xi_j,$$

in which $(y)_i$ denotes the i-th component of the vector i according to the above partitioning. The diagonal blocks in A are square and assumed nonsingular.

Now define, similarly to the scalar case, the splitting

$$A = D - E - F$$

with

$$D = \begin{pmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{pp} \end{pmatrix}, \tag{4.16}$$

$$E = -\begin{pmatrix} O & & & \\ A_{21} & O & & \\ \vdots & \vdots & \ddots & \\ A_{p1} & A_{p2} & \cdots & O \end{pmatrix}, \quad F = -\begin{pmatrix} O & A_{12} & \cdots & A_{1p} \\ & O & \cdots & A_{2p} \\ & & \ddots & \vdots \\ & & & O \end{pmatrix}.$$

With these definitions, it is easy to generalize the previous three iterative procedures defined earlier, namely, Jacobi, Gauss-Seidel, and SOR. For example, the block Jacobi iteration is now defined as a technique in which the new subvectors $\xi_i^{(k)}$ are all replaced according to

$$A_{ii}\xi_i^{(k+1)} = ((E+F)x_k)_i + \beta_i$$

or,

$$\xi_i^{(k+1)} = A_{ii}^{-1}\,((E+F)x_k)_i + A_{ii}^{-1}\beta_i, \quad i = 1, \ldots, p,$$

which leads to the same equation as before,

$$x_{k+1} = D^{-1}(E+F)x_k + D^{-1}b,$$

except that the meanings of D, E, and F have changed to their block analogues.

With finite difference approximations of PDEs, it is standard to block the variables and the matrix by partitioning along whole lines of the mesh. For example, for the two-dimensional mesh illustrated in Figure 2.5, this partitioning is

$$\xi_1 = \begin{pmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{14} \\ u_{15} \end{pmatrix}, \quad \xi_2 = \begin{pmatrix} u_{21} \\ u_{22} \\ u_{23} \\ u_{24} \\ u_{25} \end{pmatrix}, \quad \xi_3 = \begin{pmatrix} u_{31} \\ u_{32} \\ u_{33} \\ u_{34} \\ u_{35} \end{pmatrix}.$$

This corresponds to the mesh 2.5 of Chapter 2, whose associated matrix pattern is shown in Figure 2.6. A relaxation can also be defined along the vertical instead of the horizontal lines. Techniques of this type are often known as *line relaxation* techniques.

In addition, a block can also correspond to the unknowns associated with a few consecutive lines in the plane. One such blocking is illustrated in Figure 4.2 for a 6×6 grid. The corresponding matrix with its block structure is shown in Figure 4.3. An important difference between this partitioning and the one corresponding to the single-line partitioning is that now the matrices A_{ii} are block-tridiagonal instead of tridiagonal. As a result, solving linear systems with A_{ii} may be much more expensive. On the other hand, the number of iterations required to achieve convergence often decreases rapidly as the block-size increases.

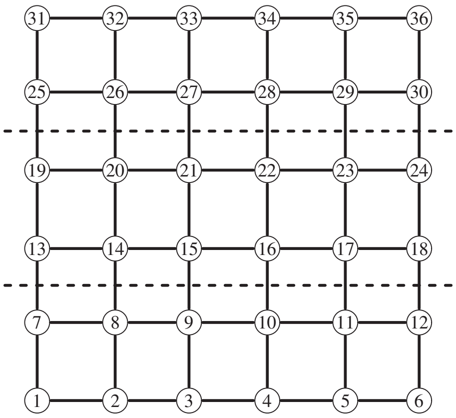


Figure 4.2 *Partitioning of a* 6×6 *square mesh into three subdomains.*

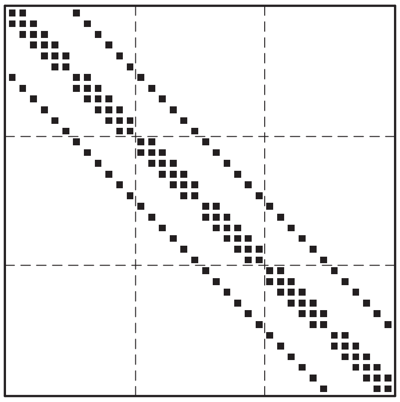


Figure 4.3 *Matrix associated with the mesh of Figure 4.2.*

Finally, block techniques can be defined in more general terms. First, by using blocks that allow us to update arbitrary groups of components, and second, by allowing the blocks to overlap. Since this is a form of the domain-decomposition method which will be seen later, we define the approach carefully. So far, our partition has been based on an actual *set-partition* of the variable set $S = \{1, 2, \ldots, n\}$ into subsets $S_1, S_2, \ldots, S_p$, with the condition that two distinct subsets are disjoint. In set theory, this is called a *partition* of S. More generally, a *set-decomposition* of S removes the constraint of disjointness. In other words it is required that the union of the subsets S_i's be equal to S:

$$S_i \subseteq S, \qquad \bigcup_{i=1,\cdots,p} S_i = S.$$

In the following, n_i denotes the size of S_i and the subset S_i is of the form,

$$S_i = \{m_i(1), m_i(2), \ldots m_i(n_i)\}.$$

A general block Jacobi iteration can be defined as follows. Let V_i be the $n \times n_i$ matrix

$$V_i = [e_{m_i(1)}, e_{m_i(2)}, \dots e_{m_i(n_i)}]$$

and

$$W_i = [\eta_{m_i(1)} e_{m_i(1)}, \eta_{m_i(2)} e_{m_i(2)}, \dots, \eta_{m_i(n_i)} e_{m_i(n_i)}],$$

where each e_j is the j-th column of the $n \times n$ identity matrix, and $\eta_{m_i(j)}$ represents a weight factor chosen so that

$$W_i^T V_i = I.$$

When there is no overlap, i.e., when the S_i's form a partition of the whole set $\{1, 2, \dots, n\}$, then define $\eta_{m_i(j)} = 1$.

Let A_{ij} be the $n_i \times n_j$ matrix

$$A_{ij} = W_i^T A V_j$$

and define similarly the partitioned vectors

$$\xi_i = W_i^T x, \quad \beta_i = W_i^T b.$$

Note that $V_i W_i^T$ is a projector from $\mathbb{R}^n$ to the subspace K_i spanned by the columns $m_i(1)$, $\dots$, $m_i(n_i)$. In addition, we have the relation

$$x = \sum_{i=1}^{s} V_i \xi_i.$$

The n_i-dimensional vector $W_i^T x$ represents the projection $V_i W_i^T x$ of x with respect to the basis spanned by the columns of V_i. The action of V_i performs the reverse operation. That means $V_i y$ is an extension operation from a vector y in K_i (represented in the basis consisting of the columns of V_i) into a vector $V_i y$ in $\mathbb{R}^n$. The operator W_i^T is termed a *restriction operator* and V_i is an *prolongation operator*.

Each component of the Jacobi iteration can be obtained by imposing the condition that the projection of the residual in the span of S_i be zero, i.e.,

$$W_i^T \left[b - A \left(V_i W_i^T x_{k+1} + \sum_{j \neq i} V_j W_j^T x_k \right) \right] = 0.$$

Remember that $\xi_j = W_j^T x$, which can be rewritten as

$$\xi_i^{(k+1)} = \xi_i^{(k)} + A_{ii}^{-1} W_i^T (b - A x_k). \tag{4.17}$$

This leads to the following algorithm:

ALGORITHM 4.1: General Block Jacobi Iteration

1. *For* $k = 0, 1, \dots,$ *until convergence Do:*
2. *For* $i = 1, 2, \dots, p$ *Do:*
3. *Solve* $A_{ii} \delta_i = W_i^T (b - A x_k)$
4. *Set* $x_{k+1} := x_k + V_i \delta_i$
5. *EndDo*
6. *EndDo*

As was the case with the scalar algorithms, there is only a slight difference between the Jacobi and Gauss-Seidel iterations. Gauss-Seidel immediately updates the component to be corrected at step i, and uses the updated approximate solution to compute the residual vector needed to correct the next component. However, the Jacobi iteration uses the same previous approximation x_k for this purpose. Therefore, the block Gauss-Seidel iteration can be defined algorithmically as follows:

ALGORITHM 4.2: General Block Gauss-Seidel Iteration

1. *Until convergence Do:*
2. *For* $i = 1, 2, \ldots, p$ *Do:*
3. *Solve* $A_{ii}\delta_i = W_i^T(b - Ax)$
4. *Set* $x := x + V_i\delta_i$
5. *EndDo*
6. *EndDo*

From the point of view of storage, Gauss-Seidel is more economical because the new approximation can be overwritten over the same vector. Also, it typically converges faster. On the other hand, the Jacobi iteration has some appeal on parallel computers since the second *Do* loop, corresponding to the p different blocks, can be executed in parallel. Although the point Jacobi algorithm by itself is rarely a successful technique for real-life problems, its block Jacobi variant, when using large enough overlapping blocks, can be quite attractive especially in a parallel computing environment.

4.1.2 ITERATION MATRICES AND PRECONDITIONING

The Jacobi and Gauss-Seidel iterations are of the form

$$x_{k+1} = Gx_k + f, \tag{4.18}$$

in which

$$G_{JA}(A) = I - D^{-1}A, \tag{4.19}$$

$$G_{GS}(A) = I - (D - E)^{-1}A, \tag{4.20}$$

for the Jacobi and Gauss-Seidel iterations, respectively. Moreover, given the matrix splitting

$$A = M - N, \tag{4.21}$$

where A is associated with the linear system (4.1), a *linear fixed-point iteration* can be defined by the recurrence

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b, \tag{4.22}$$

which has the form (4.18) with

$$G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A, \quad f = M^{-1}b. \tag{4.23}$$

For example, for the Jacobi iteration, $M = D, N = A - D$, while for the Gauss-Seidel iteration, $M = D - E, N = M - A = F$.

The iteration $x_{k+1} = Gx_k + f$ can be viewed as a technique for solving the system

$$(I - G)x = f.$$

Since G has the form $G = I - M^{-1}A$, this system can be rewritten as

$$M^{-1}Ax = M^{-1}b.$$

The above system which has the same solution as the original system is called a *preconditioned system* and M is the *preconditioning matrix* or *preconditioner*. In other words, *a relaxation scheme is equivalent to a fixed-point iteration on a preconditioned system.*

For example, for the Jacobi, Gauss-Seidel, SOR, and SSOR iterations, these preconditioning matrices are, respectively,

$$M_{JA} = D, \tag{4.24}$$

$$M_{GS} = D - E, \tag{4.25}$$

$$M_{SOR} = \frac{1}{\omega}(D - \omega E), \tag{4.26}$$

$$M_{SSOR} = \frac{1}{\omega(2-\omega)}\,(D - \omega E)D^{-1}(D - \omega F). \tag{4.27}$$

Thus, the Jacobi preconditioner is simply the diagonal of A, while the Gauss-Seidel preconditioner is the lower triangular part of A. The constant coefficients in front of the matrices M_{SOR} and M_{SSOR} only have the effect of scaling the equations of the preconditioned system uniformly. Therefore, they are unimportant in the preconditioning context.

Note that the "preconditioned" system may be a full system. Indeed, there is no reason why M^{-1} should be a sparse matrix (even though M may be sparse), since the inverse of a sparse matrix is not necessarily sparse. This limits the number of techniques that can be applied to solve the preconditioned system. Most of the iterative techniques used only require matrix-by-vector products. In this case, to compute $w = M^{-1}Av$ for a given vector v, first compute $r = Av$ and then solve the system $Mw = r$:

$$r = Av,$$
$$w = M^{-1}r.$$

In some cases, it may be advantageous to exploit the splitting $A = M - N$ and compute $w = M^{-1}Av$ as $w = (I - M^{-1}N)v$ by the procedure

$$r = Nv,$$
$$w = M^{-1}r,$$
$$w := v - w.$$

The matrix N may be sparser than A and the matrix-by-vector product Nv may be less expensive than the product Av. A number of similar but somewhat more complex ideas have been exploited in the context of preconditioned iterative methods. A few of these will be examined in Chapter 9.

4.2 CONVERGENCE

All the methods seen in the previous section define a sequence of iterates of the form

$$x_{k+1} = Gx_k + f, \tag{4.28}$$

in which G is a certain *iteration matrix*. The questions addressed in this section are: (a) if the iteration converges, then is the limit indeed a solution of the original system? (b) under which conditions does the iteration converge? (c) when the iteration does converge, how fast is it?

If the above iteration converges, its limit x satisfies

$$x = Gx + f. \tag{4.29}$$

In the case where the above iteration arises from the splitting $A = M - N$, it is easy to see that the solution x to the above system is identical to that of the original system $Ax = b$. Indeed, in this case the sequence (4.28) has the form

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b$$

and its limit satisfies

$$Mx = Nx + b,$$

or $Ax = b$. This answers question (a). Next, we focus on the other two questions.

4.2.1 GENERAL CONVERGENCE RESULT

If $I - G$ is nonsingular then there is a solution x_* to the equation (4.29). Subtracting (4.29) from (4.28) yields

$$x_{k+1} - x_* = G(x_k - x_*) = \cdots = G^{k+1}(x_0 - x_*). \tag{4.30}$$

Standard results seen in Chapter 1 imply that if the spectral radius of the iteration matrix G is less than unity, then $x_k - x_*$ converges to zero and the iteration (4.28) converges toward the solution defined by (4.29). Conversely, the relation

$$x_{k+1} - x_k = G(x_k - x_{k-1}) = \cdots = G^k(f - (I - G)x_0).$$

shows that if the iteration converges for *any* x_0 *and* f then $G^k v$ converges to zero for any vector v. As a result, $\rho(G)$ must be less than unity and the following theorem is proved:

THEOREM 4.1 *Let G be a square matrix such that $\rho(G) < 1$. Then $I - G$ is nonsingular and the iteration (4.28) converges for any f and x_0. Conversely, if the iteration (4.28) converges for for any f and x_0, then $\rho(G) < 1$.*

Since it is expensive to compute the spectral radius of a matrix, sufficient conditions that guarantee convergence can be useful in practice. One such sufficient condition could be obtained by utilizing the inequality, $\rho(G) \leq \|G\|$, for any matrix norm.

COROLLARY 4.1 *Let G be a square matrix such that $\|G\| < 1$ for some matrix norm $\|.\|$. Then $I - G$ is nonsingular and the iteration (4.28) converges for any initial vector x_0.*

Apart from knowing that the sequence (4.28) converges, it is also desirable to know *how fast* it converges. The error $d_k = x_k - x_*$ at step k satisfies

$$d_k = G^k d_0.$$

The matrix G can be expressed in the Jordan canonical form as $G = XJX^{-1}$. Assume for simplicity that there is only one eigenvalue of G of largest modulus and call it λ. Then

$$d_k = \lambda^k X \left(\frac{J}{\lambda}\right)^k X^{-1} d_0.$$

A careful look at the powers of the matrix J/λ shows that all its blocks, except the block associated with the eigenvalue λ, converge to zero as k tends to infinity. Let this Jordan block be of size p and of the form

$$J_\lambda = \lambda I + E,$$

where E is nilpotent of index p, i.e., $E^p = 0$. Then, for $k \geq p$,

$$J_\lambda^k = (\lambda I + E)^k = \lambda^k (I + \lambda^{-1} E)^k = \lambda^k \left(\sum_{i=0}^{p-1} \lambda^{-i} \binom{k}{i} E^i \right).$$

If k is large enough, then for any λ the dominant term in the above sum is the last term, i.e.,

$$J_\lambda^k \approx \lambda^{k-p+1} \binom{k}{p-1} E^{p-1}.$$

Thus, the norm of $d_k = G^k d_0$ has the asymptotical form

$$\|d_k\| \approx C \times |\lambda^{k-p+1}| \binom{k}{p-1},$$

where C is some constant. The *convergence factor* of a sequence is the limit

$$\rho = \lim_{k \to \infty} \left(\frac{\|d_k\|}{\|d_0\|} \right)^{1/k}.$$

It follows from the above analysis that $\rho = \rho(G)$. The *convergence rate* τ is the (natural) logarithm of the inverse of the convergence factor

$$\tau = -\ln \rho.$$

The above definition depends on the initial vector x_0, so it may be termed a *specific* convergence factor. A *general* convergence factor can also be defined by

$$\phi = \lim_{k \to \infty} \left(\max_{x_0 \in \mathbb{R}^n} \frac{\|d_k\|}{\|d_0\|} \right)^{1/k}.$$

This factor satisfies

$$\phi = \lim_{k\to\infty} \left(\max_{d_0 \in \mathbb{R}^n} \frac{\|G^k d_0\|}{\|d_0\|} \right)^{1/k}$$
$$= \lim_{k\to\infty} \left(\|G^k\| \right)^{1/k} = \rho(G).$$

Thus, the global asymptotic convergence factor is equal to the spectral radius of the iteration matrix G. The *general* convergence rate differs from the *specific* rate only when the initial error does not have any components in the invariant subspace associated with the dominant eigenvalue. Since it is hard to know this information in advance, the *general* convergence factor is more useful in practice.

Example 4.1 Consider the simple example of *Richardson's Iteration*,

$$x_{k+1} = x_k + \alpha\,(b - Ax_k)\,, \tag{4.31}$$

where α is a nonnegative scalar. This iteration can be rewritten as

$$x_{k+1} = (I - \alpha A)x_k + \alpha b. \tag{4.32}$$

Thus, the iteration matrix is $G_\alpha = I - \alpha A$ and the convergence factor is $\rho(I - \alpha A)$. Assume that the eigenvalues $\lambda_i, i = 1, \ldots, n$, are all real and such that,

$$\lambda_{min} \le \lambda_i \le \lambda_{max}.$$

Then, the eigenvalues μ_i of G_α are such that

$$1 - \alpha\lambda_{max} \le \mu_i \le 1 - \alpha\lambda_{min}.$$

In particular, if $\lambda_{min} < 0$ and $\lambda_{max} > 0$, at least one eigenvalue is > 1, and so $\rho(G_\alpha) > 1$ for any α. In this case the method will always diverge for some initial guess. Let us assume that all eigenvalues are positive, i.e., $\lambda_{min} > 0$. Then, the following conditions must be satisfied in order for the method to converge:

$$1 - \alpha\lambda_{min} < 1,$$
$$1 - \alpha\lambda_{max} > -1.$$

The first condition implies that $\alpha > 0$, while the second requires that $\alpha \le 2/\lambda_{max}$. In other words, the method converges for any scalar α which satisfies

$$0 < \alpha < \frac{2}{\lambda_{max}}.$$

The next question is: What is the best value α_{opt} for the parameter α, i.e., the value of α which minimizes $\rho(G_\alpha)$? The spectral radius of G_α is

$$\rho(G_\alpha) = \max\{|1 - \alpha\lambda_{min}|,\ |1 - \alpha\lambda_{max}|\}.$$

This function of α is depicted in Figure 4.4. As the curve shows, the best possible α is reached at the point where the curve $|1 - \lambda_{max}\alpha|$ with positive slope crosses the curve $|1 - \lambda_{min}\alpha|$ with negative slope, i.e., when

$$-1 + \lambda_{max}\alpha = 1 - \lambda_{min}\alpha.$$

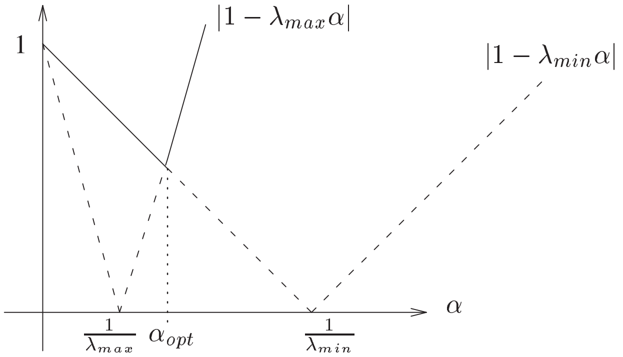


Figure 4.4 *The curve $\rho(G_\alpha)$ as a function of α.*

This gives

$$\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}}. \tag{4.33}$$

Replacing this in one of the two curves gives the corresponding optimal spectral radius

$$\rho_{opt} = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}.$$

This expression shows the difficulty with the presence of small and large eigenvalues. The convergence rate can be extremely small for realistic problems. In addition, to achieve good convergence, eigenvalue estimates are required in order to obtain the optimal or a near-optimal α, and this may cause difficulties. Finally, since λ_{max} can be very large, the curve $\rho(G_\alpha)$ can be extremely sensitive near the optimal value of α. These observations are common to many iterative methods that depend on an acceleration parameter.

4.2.2 REGULAR SPLITTINGS

DEFINITION 4.1 *Let A, M, N be three given matrices satisfying $A = M - N$. The pair of matrices M, N is a regular splitting of A, if M is nonsingular and M^{-1} and N are nonnegative.*

With a regular splitting, we associate the iteration

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b. \tag{4.34}$$

The question asked is: Under which conditions does such an iteration converge? The following result, which generalizes Theorem 1.15, gives the answer.

THEOREM 4.2 *Let M, N be a regular splitting of a matrix A. Then $\rho(M^{-1}N) < 1$ if and only if A is nonsingular and A^{-1} is nonnegative.*

Proof. Define $G = M^{-1}N$. From the fact that $\rho(G) < 1$, and the relation

$$A = M(I - G) \tag{4.35}$$

it follows that A is nonsingular. The assumptions of Theorem 1.15 are satisfied for the matrix G since $G = M^{-1}N$ is nonnegative and $\rho(G) < 1$. Therefore, $(I - G)^{-1}$ is nonnegative as is $A^{-1} = (I - G)^{-1} M^{-1}$.

To prove the sufficient condition, assume that A is nonsingular and that its inverse is nonnegative. Since A and M are nonsingular, the relation (4.35) shows again that $I - G$ is nonsingular and in addition,

$$\begin{aligned} A^{-1}N &= \left(M(I - M^{-1}N)\right)^{-1} N \\ &= (I - M^{-1}N)^{-1} M^{-1}N \\ &= (I - G)^{-1}G. \end{aligned} \tag{4.36}$$

Clearly, $G = M^{-1}N$ is nonnegative by the assumptions, and as a result of the Perron-Frobenius theorem, there is a nonnegative eigenvector x associated with $\rho(G)$ which is an eigenvalue, such that

$$Gx = \rho(G)x.$$

From this and by virtue of (4.36), it follows that

$$A^{-1}Nx = \frac{\rho(G)}{1 - \rho(G)} x.$$

Since x and $A^{-1}N$ are nonnegative, this shows that

$$\frac{\rho(G)}{1 - \rho(G)} \geq 0$$

and this can be true only when $0 \leq \rho(G) \leq 1$. Since $I - G$ is nonsingular, then $\rho(G) \neq 1$, which implies that $\rho(G) < 1$. ∎

This theorem establishes that the iteration (4.34) always converges, if M, N is a regular splitting and A is an M-matrix.

4.2.3 DIAGONALLY DOMINANT MATRICES

We begin with a few standard definitions.

DEFINITION 4.2 *A matrix A is*

- *(weakly) diagonally dominant if*

$$|a_{jj}| \geq \sum_{\substack{i=1 \\ i \neq j}}^{i=n} |a_{ij}|, \quad j = 1, \ldots, n.$$

- *strictly diagonally dominant if*

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

- *irreducibly diagonally dominant if A is irreducible, and*

$$|a_{jj}| \geq \sum_{\substack{i=1\\ i\neq j}}^{i=n} |a_{ij}|, \quad j = 1, \ldots, n.$$

with strict inequality for at least one j.

Often the term diagonally dominant is used instead of *weakly* diagonally dominant.

Diagonal dominance is related to an important result in Numerical Linear Algebra known as Gershgorin's theorem. This theorem allows rough locations for all the eigenvalues of A to be determined. In some situations, it is desirable to determine these locations in the complex plane by directly exploiting some knowledge of the entries of the matrix A. The simplest such result is the bound

$$|\lambda_i| \;\leq\; \|A\|$$

for any matrix norm. Gershgorin's theorem provides a more precise localization result.

THEOREM 4.3 ***(Gershgorin)*** *Any eigenvalue λ of a matrix A is located in one of the closed discs of the complex plane centered at a_{ii} and having the radius*

$$\rho_i = \sum_{\substack{j=1\\ j\neq i}}^{j=n} |a_{ij}|.$$

In other words,

$$\forall\ \lambda\ \in \sigma(A), \quad \exists\, i \quad \text{such that} \quad |\lambda - a_{ii}| \leq \sum_{\substack{j=1\\ j\neq i}}^{j=n} |a_{ij}|. \tag{4.37}$$

Proof. Let x be an eigenvector associated with an eigenvalue λ, and let m be the index of the component of largest modulus in x. Scale x so that $|\xi_m| = 1$, and $|\xi_i| \leq 1$, for $i \neq m$. Since x is an eigenvector, then

$$(\lambda - a_{mm})\xi_m = -\sum_{\substack{j=1\\ j\neq m}}^{n} a_{mj}\xi_j,$$

which gives

$$|\lambda - a_{mm}| \leq \sum_{\substack{j=1\\ j\neq m}}^{n} |a_{mj}||\xi_j| \leq \sum_{\substack{j=1\\ j\neq m}}^{n} |a_{mj}| = \rho_m. \tag{4.38}$$

This completes the proof. ■

Since the result also holds for the transpose of A, a version of the theorem can also be formulated based on column sums instead of row sums.

The n discs defined in the theorem are called Gershgorin discs. The theorem states that the union of these n discs contains the spectrum of A. It can also be shown that if there are m Gershgorin discs whose union S is disjoint from all other discs, then S contains exactly m eigenvalues (counted with their multiplicities). For example, when one disc is disjoint from the others, then it must contain exactly one eigenvalue.

An additional refinement which has important consequences concerns the particular case when A is irreducible.

THEOREM 4.4 *Let A be an irreducible matrix, and assume that an eigenvalue λ of A lies on the boundary of the union of the n Gershgorin discs. Then λ lies on the boundary of all Gershgorin discs.*

Proof. As in the proof of Gershgorin's theorem, let x be an eigenvector associated with λ, with $|\xi_m| = 1$, and $|\xi_i| \leq 1$, for $i \neq m$. Start from equation (4.38) in the proof of Gershgorin's theorem which states that the point λ belongs to the m-th disc. In addition, λ belongs to the boundary of the union of all the discs. As a result, it cannot be an interior point to the disc $D(\lambda, \rho_m)$. This implies that $|\lambda - a_{mm}| = \rho_m$. Therefore, the inequalities in (4.38) both become equalities:

$$|\lambda - a_{mm}| = \sum_{\substack{j=1 \\ j \neq m}}^{n} |a_{mj}||\xi_j| = \sum_{\substack{j=1 \\ j \neq m}}^{n} |a_{mj}| = \rho_m. \tag{4.39}$$

Let j be any integer $1 \leq j \leq n$. Since A is irreducible, its graph is connected and, therefore, there exists a path from node m to node j in the adjacency graph. Let this path be

$$m, m_1, m_2, \ldots, m_k = j.$$

By definition of an edge in the adjacency graph, $a_{m,m_1} \neq 0$. Because of the equality in (4.39), it is necessary that $|\xi_j| = 1$ for any nonzero ξ_j. Therefore, $|\xi_{m_1}|$ must be equal to one. Now repeating the argument with m replaced by m_1 shows that the following equality holds:

$$|\lambda - a_{m_1,m_1}| = \sum_{\substack{j=1 \\ j \neq m_1}}^{n} |a_{m_1,j}||\xi_j| = \sum_{\substack{j=1 \\ j \neq m_1}}^{n} |a_{m_1,j}| = \rho_{m_1}. \tag{4.40}$$

The argument can be continued showing each time that

$$|\lambda - a_{m_i,m_i}| = \rho_{m_i}, \tag{4.41}$$

and this is valid for $i = 1, \ldots, k$. In the end, it will be proved that λ belongs to the boundary of the j-th disc for an arbitrary j. ∎

An immediate corollary of the Gershgorin theorem and the above theorem follows.

COROLLARY 4.2 *If a matrix A is strictly diagonally dominant or irreducibly diagonally dominant, then it is nonsingular.*

Proof. If a matrix is strictly diagonally dominant, then the union of the Gershgorin disks excludes the origin, so $\lambda = 0$ cannot be an eigenvalue. Assume now that it is only irreducibly diagonal dominant. Then if it is singular, the zero eigenvalue lies on the boundary of the union of the Gershgorin disks. In this situation, according to the previous theorem, this eigenvalue should lie on the boundary of all the disks. This would mean that

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

which contradicts the assumption of irreducible diagonal dominance. ∎

The following theorem can now be stated.

THEOREM 4.5 *If A is a strictly diagonally dominant or an irreducibly diagonally dominant matrix, then the associated Jacobi and Gauss-Seidel iterations converge for any x_0.*

Proof. We first prove the results for strictly diagonally dominant matrices. Let λ be the dominant eigenvalue of the iteration matrix $M_J = D^{-1}(E + F)$ for Jacobi and $M_G = (D - E)^{-1}F$ for Gauss-Seidel. As in the proof of Gershgorin's theorem, let x be an eigenvector associated with λ, with $|\xi_m| = 1$, and $|\xi_i| \leq 1$, for $i \neq 1$. Start from equation (4.38) in the proof of Gershgorin's theorem which states that for M_J,

$$|\lambda| \leq \sum_{\substack{j=1 \\ j \neq m}}^{n} \frac{|a_{mj}|}{|a_{mm}|} |\xi_j| \leq \sum_{\substack{j=1 \\ j \neq m}}^{n} \frac{|a_{mj}|}{|a_{mm}|} < 1.$$

This proves the result for Jacobi's method.

For the Gauss-Seidel iteration, write the m-th row of the equation $Fx = \lambda(D - E)x$ in the form

$$\sum_{j<m} a_{mj}\xi_j = \lambda \left(a_{mm}\xi_m + \sum_{j>m} a_{mj}\xi_j \right),$$

which yields the inequality

$$|\lambda| \leq \frac{\sum_{j<m} |a_{mj}||\xi_j|}{|a_{mm}| - \sum_{j>m} |a_{mj}||\xi_j|} \leq \frac{\sum_{j<m} |a_{mj}|}{|a_{mm}| - \sum_{j>m} |a_{mj}|}.$$

The last term in the above equation has the form $\sigma_2/(d - \sigma_1)$ with d, σ_1, σ_2 all nonnegative and $d - \sigma_1 - \sigma_2 > 0$. Therefore,

$$|\lambda| \leq \frac{\sigma_2}{\sigma_2 + (d - \sigma_2 - \sigma_1)} < 1.$$

In the case when the matrix is only irreducibly diagonally dominant, the above proofs only show that $\rho(M^{-1}N) \leq 1$, where $M^{-1}N$ is the iteration matrix for either Jacobi or Gauss-Seidel. A proof by contradiction will be used to show that in fact $\rho(M^{-1}N) < 1$. Assume that λ is an eigenvalue of $M^{-1}N$ with $|\lambda| = 1$. Then the matrix $M^{-1}N - \lambda I$ would be singular and, as a result, $A' = N - \lambda M$ would also be singular. Since $|\lambda| = 1$, it is clear that A' is also an irreducibly diagonally dominant matrix. This would contradict Corollary 4.2. ∎

4.2.4 SYMMETRIC POSITIVE DEFINITE MATRICES

It is possible to show that when A is Symmetric Positive Definite, then SOR will converge for any ω in the open interval $(0, 2)$ and for any initial guess x_0. In fact, the reverse is also true under certain assumptions.

THEOREM 4.6 *If A is symmetric with positive diagonal elements and for $0 < \omega < 2$, SOR converges for any x_0 if and only if A is positive definite.*

4.2.5 PROPERTY A AND CONSISTENT ORDERINGS

A number of properties which are related to the graph of a finite difference matrix are now defined. The first of these properties is called Property A. A matrix has Property A if its graph is *bipartite*. This means that the graph is two-colorable in the sense defined in Chapter 3: Its vertices can be partitioned in two sets in such a way that no two vertices in the same set are connected by an edge. Note that, as usual, the self-connecting edges which correspond to the diagonal elements are ignored.

DEFINITION 4.3 *A matrix has Property A if the vertices of its adjacency graph can be partitioned in two sets S_1 and S_2, so that any edge in the graph links a vertex of S_1 to a vertex of S_2.*

In other words, nodes from the first set are connected only to nodes from the second set and vice versa. This definition is illustrated in Figure 4.5.

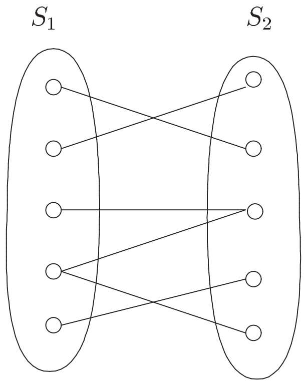


Figure 4.5 *Graph illustration of Property A.*

An alternative definition is that a matrix has Property A if it can be permuted into a matrix with the following structure:

$$A' = \begin{pmatrix} D_1 & -F \\ -E & D_2 \end{pmatrix}, \tag{4.42}$$

where D_1 and D_2 are diagonal matrices. This structure can be obtained by first labeling all the unknowns in S_1 from 1 to n_1, in which $n_1 = |S_1|$ and the rest from $n_1 + 1$ to n. Note that the Jacobi iteration matrix will have the same structure except that the D_1, D_2 blocks will be replaced by zero blocks. These Jacobi iteration matrices satisfy an important property stated in the following proposition.

PROPOSITION 4.1 *Let B be a matrix with the following structure:*

$$B = \begin{pmatrix} O & B_{12} \\ B_{21} & O \end{pmatrix}, \tag{4.43}$$

and let L and U be the lower and upper triangular parts of B, respectively. Then

1. *If μ is an eigenvalue of B, then so is $-\mu$.*
2. *The eigenvalues of the matrix*

$$B(\alpha) = \alpha L + \frac{1}{\alpha} U$$

defined for $\alpha \neq 0$ are independent of α.

Proof. The first property is shown by simply observing that if $\binom{x}{v}$ is an eigenvector associated with μ, then $\begin{pmatrix} x \\ -v \end{pmatrix}$ is an eigenvector of B associated with the eigenvalue $-\mu$.

Consider the second property. For any α, the matrix $B(\alpha)$ is similar to B, i.e., $B(\alpha) = XBX^{-1}$ with X defined by

$$X = \begin{pmatrix} 1 & O \\ O & \alpha \end{pmatrix}.$$

This proves the desired result ■

A definition which generalizes this important property is *consistently ordered matrices*. Varga [213] calls a consistently ordered matrix one for which the eigenvalues of $B(\alpha)$ are independent of α. Another definition given by Young [232] considers a specific class of matrices which generalize this property. We will use this definition here. Unlike Property A, the consistent ordering property depends on the initial ordering of the unknowns.

DEFINITION 4.4 *A matrix is said to be* consistently ordered *if the vertices of its adjacency graph can be partitioned in p sets $S_1, S_2, \ldots, S_p$ with the property that any two adjacent vertices i and j in the graph belong to two consecutive partitions S_k and $S_{k'}$, with $k' = k - 1$, if $j < i$, and $k' = k + 1$, if $j > i$.*

It is easy to show that consistently ordered matrices satisfy property A: the first color is made up of all the partitions S_i with odd i and the second with the partitions S_i with even i.

Example 4.2 Block tridiagonal matrices of the form

$$T = \begin{pmatrix} D_1 & T_{12} & & & \\ T_{21} & D_2 & T_{23} & & \\ & T_{32} & D_3 & \ddots & \\ & & \ddots & \ddots & T_{p-1,p} \\ & & & T_{p,p-1} & D_p \end{pmatrix}$$

whose diagonal blocks D_i are diagonal matrices are called T-matrices. Clearly, such matrices are consistently ordered. Note that matrices of the form (4.42) are a particular case with $p = 2$.

Consider now a general, consistently ordered matrix. By definition, there is permutation π of $\{1, 2, \ldots, n\}$ which is the union of p disjoint subsets

$$\pi = \pi_1 \bigcup \pi_2 \ldots \bigcup \pi_p \tag{4.44}$$

with the property that if $a_{ij} \neq 0$, $j \neq i$ and i belongs to π_k, then j belongs to $\pi_{k\pm1}$ depending on whether $i < j$ or $i > j$. This permutation π can be used to permute A symmetrically. If P is the permutation matrix associated with the permutation π, then clearly

$$A' = P^T A P$$

is a T-matrix.

Not every matrix that can be symmetrically permuted into a T-matrix is consistently ordered. The important property here is that the partition $\{\pi_i\}$ *preserves the order of the indices i, j of nonzero elements*. In terms of the adjacency graph, there is a partition of the graph with the property that an oriented edge i, j from i to j always points to a set with a larger index if $j > i$, or a smaller index otherwise. In particular, a very important consequence is that edges corresponding to the lower triangular part will remain so in the permuted matrix. The same is true for the upper triangular part. Indeed, if a nonzero element in the permuted matrix is $a'_{i',j'} = a_{\pi^{-1}(i),\pi^{-1}(j)} \neq 0$ with $i' > j'$, then by definition of the permutation $\pi(i') > \pi(j')$, or $i = \pi(\pi^{-1}(i)) > j = \pi(\pi^{-1}(j))$. Because of the order preservation, it is necessary that $i > j$. A similar observation holds for the upper triangular part. Therefore, this results in the following proposition.

PROPOSITION 4.2 *If a matrix A is consistently ordered, then there exists a permutation matrix P such that $P^T A P$ is a T-matrix and*

$$(P^T AP)_L = P^T A_L P, \quad (P^T AP)_U = P^T A_U P \tag{4.45}$$

in which X_L represents the (strict) lower part of X and X_U the (strict) upper part of X.

With the above property it can be shown that for consistently ordered matrices the eigenvalues of $B(\alpha)$ as defined in Proposition 4.1 are also invariant with respect to α.

PROPOSITION 4.3 *Let B be the Jacobi iteration matrix associated with a consistently ordered matrix A, and let L and U be the lower and upper triangular parts of B, respec-*

tively. Then the eigenvalues of the matrix

$$B(\alpha) = \alpha L + \frac{1}{\alpha} U$$

defined for $\alpha \neq 0$ do not depend on α.

Proof. First transform $B(\alpha)$ into a T-matrix using the permutation π in (4.44) provided by the previous proposition

$$P^T B(\alpha) P = \alpha P^T L P + \frac{1}{\alpha} P^T U P.$$

From the previous proposition, the lower part of $P^T B P$ is precisely $L' = P^T L P$. Similarly, the upper part is $U' = P^T U P$, the lower and upper parts of the associated T-matrix. Therefore, we only need to show that the property is true for a T-matrix.

In this case, for any α, the matrix $B(\alpha)$ is similar to B. This means that $B(\alpha) = X B X^{-1}$ with X being equal to

$$X = \begin{pmatrix} 1 & & & & \\ & \alpha I & & & \\ & & \alpha^2 I & & \\ & & & \ddots & \\ & & & & \alpha^{p-1} I \end{pmatrix},$$

where the partitioning is associated with the subsets $\pi_1, \ldots, \pi_p$ respectively. ∎

Note that T-matrices and matrices with the structure (4.42) are two particular cases of matrices which fulfill the assumptions of the above proposition. There are a number of well known properties related to Property A and consistent orderings. For example, it is possible to show that,

- Property A is invariant under symmetric permutations.
- A matrix has Property A if and only if there is a permutation matrix P such that $A' = P^{-1} A P$ is consistently ordered.

Consistently ordered matrices satisfy an important property which relates the eigenvalues of the corresponding SOR iteration matrices to those of the Jacobi iteration matrices. The main theorem regarding the theory for SOR is a consequence of the following result proved by Young [232]. Remember that

$$\begin{aligned} M_{SOR} &= (D - \omega E)^{-1} (\omega F + (1-\omega) D) \\ &= (I - \omega D^{-1} E)^{-1} \left(\omega D^{-1} F + (1-\omega) I \right). \end{aligned}$$

THEOREM 4.7 *Let A be a consistently ordered matrix such that $a_{ii} \neq 0$ for $i = 1, \ldots, n$, and let $\omega \neq 0$. Then if λ is a nonzero eigenvalue of the SOR iteration matrix M_{SOR}, any scalar μ such that*

$$(\lambda + \omega - 1)^2 = \lambda \omega^2 \mu^2 \tag{4.46}$$

is an eigenvalue of the Jacobi iteration matrix B. Conversely, if μ is an eigenvalue of the Jacobi matrix B and if a scalar λ satisfies (4.46), then λ is an eigenvalue of M_{SOR}.

Proof. Denote $D^{-1}E$ by L and $D^{-1}F$ by U, so that

$$M_{SOR} = (I - \omega L)^{-1} (\omega U + (1 - \omega)I)$$

and the Jacobi iteration matrix is merely $L + U$. Writing that λ is an eigenvalue yields

$$\det \left(\lambda I - (I - \omega L)^{-1}(\omega U + (1 - \omega)I)\right) = 0$$

which is equivalent to

$$\det \left(\lambda(I - \omega L) - (\omega U + (1 - \omega)I)\right) = 0$$

or

$$\det \left((\lambda + \omega - 1)I - \omega(\lambda L + U)\right) = 0.$$

Since $\omega \neq 0$, this can be rewritten as

$$\det \left(\frac{\lambda + \omega - 1}{\omega} I - (\lambda L + U)\right) = 0,$$

which means that $(\lambda + \omega - 1)/\omega$ is an eigenvalue of $\lambda L + U$. Since A is consistently ordered, the eigenvalues of $\lambda L + U$ which are equal to $\lambda^{1/2}(\lambda^{1/2}L + \lambda^{-1/2}U)$ are the same as those of $\lambda^{1/2}(L + U)$, where $L + U$ is the Jacobi iteration matrix. The proof follows immediately. ∎

This theorem allows us to compute an optimal value for ω, which can be shown to be equal to

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(B)^2}}. \tag{4.47}$$

A typical SOR procedure starts with some ω, for example, $\omega = 1$, then proceeds with a number of SOR steps with this ω. The convergence rate for the resulting iterates is estimated providing an estimate for $\rho(B)$ using Theorem 4.7. A better ω is then obtained from the formula (4.47), and the iteration restarted. Further refinements of the optimal ω are calculated and retrofitted in this manner as the algorithm progresses.

4.3 ALTERNATING DIRECTION METHODS

The Alternating Direction Implicit (ADI) method was introduced in the mid-1950s by Peaceman and Rachford [162] specifically for solving equations arising from finite difference discretizations of elliptic and parabolic Partial Differential Equations. Consider a partial differential equation of elliptic type

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u(x,y)}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u(x,y)}{\partial y}\right) = f(x,y) \tag{4.48}$$

on a rectangular domain with Dirichlet boundary conditions. The equations are discretized with centered finite differences using $n + 2$ points in the x direction and $m + 2$ points in the y direction, This results in the system of equations

$$Hu + Vu = b, \tag{4.49}$$

in which the matrices H and V represent the three-point central difference approximations to the operators

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial}{\partial x}\right) \quad \text{and} \quad \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial}{\partial y}\right),$$

respectively. In what follows, the same notation is used to represent the discretized version of the unknown function u.

The ADI algorithm consists of iterating by solving (4.49) in the x and y directions alternatively as follows.

ALGORITHM 4.3: Peaceman-Rachford (PR) ADI

1. *For* $k = 0., 1, \ldots,$ *until convergence Do:*
2. *Solve:* $(H + \rho_k I)u_{k+\frac{1}{2}} = (\rho_k I - V)u_k + b$
3. *Solve:* $(V + \rho_k I)u_{k+1} = (\rho_k I - H)u_{k+\frac{1}{2}} + b$
4. *EndDo*

Here, $\rho_k, k = 1, 2, \ldots$, is a sequence of positive acceleration parameters.

The specific case where ρ_k is chosen to be a constant ρ deserves particular attention. In this case, we can formulate the above iteration in the usual form of (4.28) with

$$G = (V + \rho I)^{-1}(H - \rho I)(H + \rho I)^{-1}(V - \rho I), \tag{4.50}$$

$$f = (V + \rho I)^{-1}\left[I - (H - \rho I)(H + \rho I)^{-1}\right] b \tag{4.51}$$

or, when $\rho > 0$, in the form (4.22) with

$$M = \frac{1}{2\rho}(H + \rho I)(V + \rho I), \quad N = \frac{1}{2\rho}(H - \rho I)(V - \rho I). \tag{4.52}$$

Note that (4.51) can be rewritten in a simpler form; see Exercise 5.

The ADI algorithm is often formulated for solving the time-dependent Partial Differential Equation

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right) \tag{4.53}$$

on the domain $(x, y, t) \in \Omega \times [0, T] \equiv (0, 1) \times (0, 1) \times [0, T]$. The initial and boundary conditions are:

$$u(x, y, 0) = x_0(x, y), \ \ \forall (x, y) \in \Omega, \tag{4.54}$$

$$u(\bar{x}, \bar{y}, t) = g(\bar{x}, \bar{y}, t), \ \ \forall (\bar{x}, \bar{y}) \in \partial\Omega, \ \ t > 0, \tag{4.55}$$

where $\partial\Omega$ is the boundary of the unit square Ω. The equations are discretized with respect to the space variables x and y as before, resulting in a system of Ordinary Differential Equations:

$$\frac{du}{dt} = Hu + Vu, \tag{4.56}$$

in which the matrices H and V have been defined earlier. The Alternating Direction Implicit algorithm advances the relation (4.56) forward in time alternately in the x and y directions as follows:

$$(I - \frac{1}{2}\Delta t\ H)u_{k+\frac{1}{2}} = (I + \frac{1}{2}\Delta t\ V)u_k\ ,$$

$$(I - \frac{1}{2}\Delta t\ V)u_{k+1} = (I + \frac{1}{2}\Delta t\ H)u_{k+\frac{1}{2}}.$$

The acceleration parameters ρ_k of Algorithm 4.3 are replaced by a natural time-step.

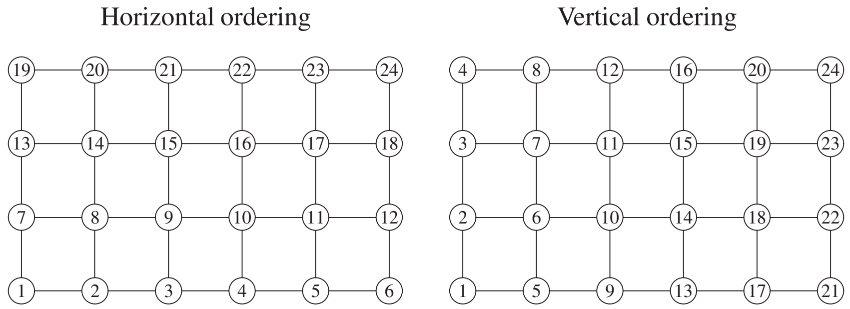


Figure 4.6 *The horizontal and vertical orderings for the unknowns in ADI.*

Assuming that the mesh-points are ordered by lines in the x-direction, then the first step of Algorithm 4.3 constitutes a set of m independent tridiagonal linear systems of size n each. However, the second step constitutes a large tridiagonal system whose three diagonals are offset by $-m$, 0, and m, respectively. This second system can also be rewritten as a set of n independent tridiagonal systems of size m each by reordering the grid points by lines, this time in the y direction. The natural (horizontal) and vertical orderings are illustrated in Figure 4.6. Whenever moving from one half step of ADI to the next, we must implicitly work with the transpose of the matrix representing the solution on the $n \times m$ grid points. This data operation may be an expensive task on parallel machines and often it is cited as one of the drawbacks of Alternating Direction Methods in this case.

ADI methods were extensively studied in the 1950s and 1960s for the particular case of positive definite systems. For such systems, H and V have real eigenvalues and the following is a summary of the main results in this situation. First, when H and V are Symmetric Positive Definite, then the stationary iteration ($\rho_k = \rho > 0$, for all k) converges. For the model problem, the asymptotic rate of convergence of the stationary ADI iteration using the optimal ρ is the same as that of SSOR using the optimal ω. However, each ADI step is more expensive than one SSOR step. One of the more important results in the ADI theory is that the rate of convergence of ADI can be increased appreciably by using a cyclic sequence of parameters, ρ_k. A theory for selecting the best sequence of ρ_k's is well understood in the case when H and V commute [26]. For the model problem, the parameters can be selected so that the time complexity is reduced to $O(n^2 \log n)$, for details see [162].

EXERCISES

1. Consider an $n \times n$ tridiagonal matrix of the form

$$T_\alpha = \begin{pmatrix} \alpha & -1 & & & & \\ -1 & \alpha & -1 & & & \\ & -1 & \alpha & -1 & & \\ & & -1 & \alpha & -1 & \\ & & & -1 & \alpha & -1 \\ & & & & -1 & \alpha \end{pmatrix}, \tag{4.57}$$

where α is a real parameter.

a. Verify that the eigenvalues of T_α are given by

$$\lambda_j = \alpha - 2\cos(j\theta) \quad j = 1, \ldots, n,$$

where

$$\theta = \frac{\pi}{n+1}$$

and that an eigenvector associated with each λ_j is

$$q_j = [\sin(j\theta), \sin(2j\theta), \ldots, \sin(nj\theta)]^T .$$

Under what condition on α does this matrix become positive definite?

b. Now take $\alpha = 2$. How does this matrix relate to the matrices seen in Chapter 2 for one-dimensional problems?

i. Will the Jacobi iteration converge for this matrix? If so, what will its convergence factor be?

ii. Will the Gauss-Seidel iteration converge for this matrix? If so, what will its convergence factor be?

iii. For which values of ω will the SOR iteration converge?

2. Prove that the iteration matrix G_ω of SSOR, as defined by (4.13), can be expressed as

$$G_\omega = I - \omega(2-\omega)(D - \omega F)^{-1} D (D - \omega E)^{-1} A.$$

Deduce the expression (4.27) for the preconditioning matrix associated with the SSOR iteration.

3. Let A be a matrix with a positive diagonal D.

a. Obtain an expression equivalent to that of (4.13) for G_ω but which involves the matrices $S_E \equiv D^{-1/2} E D^{-1/2}$ and $S_F \equiv D^{-1/2} F D^{-1/2}$.

b. Show that

$$D^{1/2} G_\omega D^{-1/2} = (I - \omega S_F)^{-1} (I - \omega S_E)^{-1} (\omega S_E + (1-\omega) I)(\omega S_F + (1-\omega) I)$$

c. Now assume that in addition to having a positive diagonal, A is symmetric. Prove that the eigenvalues of the SSOR iteration matrix G_ω are real and nonnegative.

4. Let

$$A = \begin{pmatrix} D_1 & -F_2 & & & \\ -E_2 & D_2 & -F_3 & & \\ & -E_3 & D_3 & \ddots & \\ & & \ddots & \ddots & -F_m \\ & & & -E_m & D_m \end{pmatrix},$$

where the D_i blocks are nonsingular matrices which are not necessarily diagonal.

a. What are the *block Jacobi* and *block Gauss-Seidel* iteration matrices?

b. Show a result similar to that in Proposition 4.3 for the Jacobi iteration matrix.

c. Show also that for $\omega = 1$ (1) the block Gauss-Seidel and block Jacobi iterations either both converge or both diverge, and (2) when they both converge, then the block Gauss-Seidel iteration is (asymptotically) twice as fast as the block Jacobi iteration.

5. According to formula (4.23), the f vector in iteration (4.22) should be equal to $M^{-1}b$, where b is the right-hand side and M is given in (4.52). Yet, formula (4.51) gives a different expression for f. Reconcile the two results, i.e., show that the expression (4.51) can also be rewritten as

$$f = 2\rho(V + \rho I)^{-1}(H + \rho I)^{-1}b.$$

6. Show that a matrix has Property A if and only if there is a permutation matrix P such that $A' = P^{-1}AP$ is consistently ordered.

7. Consider a matrix A which is consistently ordered. Show that the asymptotic convergence rate for Gauss-Seidel is double that of the Jacobi iteration.

8. A matrix of the form

$$B = \begin{pmatrix} 0 & E & 0 \\ 0 & 0 & F \\ H & 0 & 0 \end{pmatrix}$$

is called a three-cyclic matrix.

a. What are the eigenvalues of B? (Express them in terms of eigenvalues of a certain matrix which depends on E, F, and H.)

b. Assume that a matrix A has the form $A = D + B$, where D is a nonsingular diagonal matrix, and B is three-cyclic. How can the eigenvalues of the Jacobi iteration matrix be related to those of the Gauss-Seidel iteration matrix? How does the asymptotic convergence rate of the Gauss-Seidel iteration compare with that of the Jacobi iteration matrix in this case?

c. Answer the same questions as in (2) for the case when SOR replaces the Gauss-Seidel iteration.

d. Generalize the above results to p-cyclic matrices, i.e., matrices of the form

$$B = \begin{pmatrix} 0 & E_1 & & & \\ & 0 & E_2 & & \\ & & 0 & \ddots & \\ & & & 0 & E_{p-1} \\ E_p & & & & 0 \end{pmatrix}.$$

NOTES AND REFERENCES. Two good references for the material covered in this chapter are Varga [213] and and Young [232]. Although relaxation-type methods were very popular up to the 1960s, they are now mostly used as preconditioners, a topic which will be seen in detail in Chapters 9 and 10. One of the main difficulties with these methods is finding an optimal relaxation factor for general matrices. Theorem 4.4 is due to Ostrowski. For details on the use of Gershgorin's theorem in eigenvalue problems, see [180]. The original idea of the ADI method is described in [162] and those results on the optimal parameters for ADI can be found in [26]. A comprehensive text on this class of techniques can be found in [220]. Not covered in this book is the related class of multigrid methods; see the reference [115] for a detailed exposition. Closely related to the multigrid approach is the Aggregation-Disaggregation technique which is popular in Markov chain modeling. A recommended book for these methods and others used in the context of Markov chain modeling is [203]. ■

CHAPTER

5

PROJECTION METHODS

Most of the existing practical iterative techniques for solving large linear systems of equations utilize a projection process in one way or another. A projection process represents a *canonical* way for extracting an approximation to the solution of a linear system from a subspace. This chapter describes these techniques in a very general framework and presents some theory. The one-dimensional case is covered in detail at the end of the chapter, as it provides a good preview of the more complex projection processes to be seen in later chapters.

5.1 BASIC DEFINITIONS AND ALGORITHMS

Consider the linear system

$$Ax = b, \tag{5.1}$$

where A is an $n \times n$ real matrix. In this chapter, the same symbol A is often used to denote the matrix and the linear mapping in $\mathbb{R}^n$ that it represents. The idea of *projection techniques* is to extract an approximate solution to the above problem from a subspace of $\mathbb{R}^n$. If $\mathcal{K}$ is this subspace of *candidate approximants*, or *search subspace*, and if m is its dimension, then, in general, m constraints must be imposed to be able to extract such an approximation. A typical way of describing these constraints is to impose m (independent) orthogonality conditions. Specifically, the residual vector $b - Ax$ is constrained to be orthogonal to m linearly independent vectors. This defines another subspace $\mathcal{L}$ of dimension m which will be called the *subspace of constraints* or *left subspace* for reasons that will be explained below. This simple framework is common to many different mathematical methods and is known as the Petrov-Galerkin conditions.

There are two broad classes of projection methods: *orthogonal* and *oblique*. In an orthogonal projection technique, the subspace $\mathcal{L}$ is the same as $\mathcal{K}$. In an oblique projection method, $\mathcal{L}$ is different from $\mathcal{K}$ and may be totally unrelated to it. This distinction is rather important and gives rise to different types of algorithms.

5.1.1 GENERAL PROJECTION METHODS

Let A be an $n \times n$ real matrix and $\mathcal{K}$ and $\mathcal{L}$ be two m-dimensional subspaces of $\mathbb{R}^n$. A projection technique onto the subspace $\mathcal{K}$ and orthogonal to $\mathcal{L}$ is a process which finds an approximate solution $\tilde{x}$ to (5.1) by imposing the conditions that $\tilde{x}$ belong to $\mathcal{K}$ and that the new residual vector be orthogonal to $\mathcal{L}$,

$$\text{Find } \tilde{x} \in \mathcal{K}, \quad \text{such that} \quad b - A\tilde{x} \perp \mathcal{L}. \tag{5.2}$$

If we wish to exploit the knowledge of an initial guess x_0 to the solution, then the approximation must be sought in the affine space $x_0 + \mathcal{K}$ instead of the homogeneous vector space $\mathcal{K}$. This requires a slight modification to the above formulation. The approximate problem should be redefined as

$$\text{Find} \quad \tilde{x} \in x_0 + \mathcal{K}, \quad \text{such that} \quad b - A\tilde{x} \perp \mathcal{L}. \tag{5.3}$$

Note that if $\tilde{x}$ is written in the form $\tilde{x} = x_0 + \delta$, and the initial residual vector r_0 is defined as

$$r_0 = b - Ax_0, \tag{5.4}$$

then the above equation becomes $b - A(x_0 + \delta) \perp \mathcal{L}$ or

$$r_0 - A\delta \perp \mathcal{L}.$$

In other words, the approximate solution can be defined as

$$\tilde{x} = x_0 + \delta, \quad \delta \in \mathcal{K}, \tag{5.5}$$

$$(r_0 - A\delta, w) = 0, \quad \forall w \in \mathcal{L}. \tag{5.6}$$

The orthogonality condition (5.6) imposed on the new residual $r_{new} = r_0 - A\delta$ is illustrated in Figure 5.1.

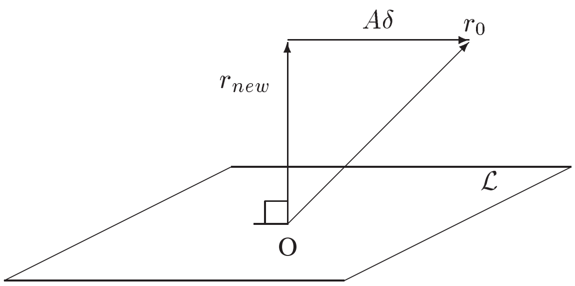


Figure 5.1 *Interpretation of the orthogonality condition.*

This is a basic projection step, in its most general form. Most standard techniques use a succession of such projections. Typically, a new projection step uses a new pair of subspace $\mathcal{K}$ and $\mathcal{L}$ and an initial guess x_0 equal to the most recent approximation obtained from the previous projection step. Projection methods form a unifying framework for many of the well known methods in scientific computing. In fact, virtually all of the basic iterative techniques seen in the previous chapter can be considered projection techniques. Whenever an approximation is defined via m degrees of freedom (subspace $\mathcal{K}$) and m constraints (Subspace $\mathcal{L}$), a projection process results.

Example 5.1 In the simplest case, an elementary Gauss-Seidel step as defined by (4.6) is nothing but a projection step with $\mathcal{K} = \mathcal{L} = \text{span}\{e_i\}$. These projection steps are cycled for $i = 1, \ldots, n$ until convergence. See Exercise 1 for an alternative way of selecting the sequence of e_i's.

Orthogonal projection methods correspond to the particular case when the two subspaces $\mathcal{L}$ and $\mathcal{K}$ are identical. The distinction is particularly important in the Hermitian case since we are guaranteed that the projected problem will be Hermitian in this situation, as will be seen shortly. In addition, a number of helpful theoretical results are true for the orthogonal case. When $\mathcal{L} = \mathcal{K}$, the Petrov-Galerkin conditions are called the Galerkin conditions.

5.1.2 MATRIX REPRESENTATION

Let $V = [v_1, \ldots, v_m]$, an $n \times m$ matrix whose column-vectors form a basis of $\mathcal{K}$ and, similarly, $W = [w_1, \ldots, w_m]$, an $n \times m$ matrix whose column-vectors form a basis of $\mathcal{L}$. If the approximate solution is written as

$$x = x_0 + Vy,$$

then the orthogonality condition leads immediately to the following system of equations for the vector y:

$$W^T AVy = W^T r_0.$$

If the assumption is made that the $m \times m$ matrix $W^T AV$ is nonsingular, the following expression for the approximate solution $\tilde{x}$ results,

$$\tilde{x} = x_0 + V(W^T AV)^{-1} W^T r_0. \tag{5.7}$$

In many algorithms, the matrix $W^T AV$ does not have to be formed since it is available as a by-product of the algorithm. A prototype projection technique is represented by the following algorithm.

ALGORITHM 5.1: Prototype Projection Method

1. *Until convergence, Do:*
2. *Select a pair of subspaces $\mathcal{K}$ and $\mathcal{L}$*
3. *Choose bases $V = [v_1, \ldots, v_m]$ and $W = [w_1, \ldots, w_m]$ for $\mathcal{K}$ and $\mathcal{L}$*
4. $r := b - Ax$
5. $y := (W^T AV)^{-1} W^T r$
6. $x := x + Vy$
7. *EndDo*

The approximate solution is defined only when the matrix $W^T AV$ is nonsingular, which is not guaranteed to be true even when A is nonsingular.

Example 5.2 As an example, consider the matrix

$$A = \begin{pmatrix} O & I \\ I & I \end{pmatrix},$$

where I is the $m \times m$ identity matrix and O is the $m \times m$ zero matrix, and let $V = W = [e_1, e_2, \ldots, e_m]$. Although A is nonsingular, the matrix $W^T AV$ is precisely the O block in the upper-left corner of A and is therefore singular.

There are two important particular cases where the nonsingularity of $W^T AV$ is guaranteed. These are discussed in the following proposition.

PROPOSITION 5.1 *Let A, $\mathcal{L}$, and $\mathcal{K}$ satisfy either one of the two following conditions,*

i. *A is positive definite and $\mathcal{L} = \mathcal{K}$, or*

ii. *A is nonsingular and $\mathcal{L} = A\mathcal{K}$.*

Then the matrix $B = W^T AV$ is nonsingular for any bases V and W of $\mathcal{K}$ and $\mathcal{L}$, respectively.

Proof. Consider first the case (i). Let V be any basis of $\mathcal{K}$ and W be any basis of $\mathcal{L}$. In fact, since $\mathcal{L}$ and $\mathcal{K}$ are the same, W can always be expressed as $W = VG$, where G is a nonsingular $m \times m$ matrix. Then

$$B = W^T AV = G^T V^T AV.$$

Since A is positive definite, so is $V^T AV$, see Chapter 1, and this shows that B is nonsingular.

Consider now case (ii). Let V be any basis of $\mathcal{K}$ and W be any basis of $\mathcal{L}$. Since $\mathcal{L} = A\mathcal{K}$, W can be expressed in this case as $W = AVG$, where G is a nonsingular $m \times m$ matrix. Then

$$B = W^T AV = G^T (AV)^T AV. \tag{5.8}$$

Since A is nonsingular, the $n \times m$ matrix AV is of full rank and as a result, $(AV)^T AV$ is nonsingular. This, along with (5.8), shows that B is nonsingular. ■

Now consider the particular case where A is symmetric (real) and an orthogonal projection technique is used. In this situation, the same basis can be used for $\mathcal{L}$ and $\mathcal{K}$, which are identical subspaces, and the projected matrix, which is $B = V^T AV$, is symmetric. In addition, if the matrix A is Symmetric Positive Definite, then so is B.

5.2 GENERAL THEORY

This section gives some general theoretical results without being specific about the subspaces $\mathcal{K}$ and $\mathcal{L}$ which are used. The goal is to learn about the quality of the approximation obtained from a general projection process. Two main tools are used for this. The first is to exploit optimality properties of projection methods. These properties are induced from those properties of projectors seen in Section 1.12.4 of Chapter 1. The second tool consists of interpreting the projected problem with the help of projection operators in an attempt to extract residual bounds.

5.2.1 TWO OPTIMALITY RESULTS

In this section, two important optimality results will be established that are satisfied by the approximate solutions in some cases. Consider first the case when A is SPD.

PROPOSITION 5.2 *Assume that A is Symmetric Positive Definite and $\mathcal{L} = \mathcal{K}$. Then a vector $\tilde{x}$ is the result of an (orthogonal) projection method onto $\mathcal{K}$ with the starting vector x_0 if and only if it minimizes the A-norm of the error over $x_0 + \mathcal{K}$, i.e., if and only if*

$$E(\tilde{x}) = \min_{x \in x_0 + \mathcal{K}} E(x),$$

where

$$E(x) \equiv (A(x_* - x), x_* - x)^{1/2}.$$

Proof. As was seen in Section 1.12.4, for $\tilde{x}$ to be the minimizer of $E(x)$, it is necessary and sufficient that $x_* - \tilde{x}$ be A-orthogonal to all the subspace $\mathcal{K}$. This yields

$$(A(x_* - \tilde{x}), v) = 0, \quad \forall v \quad \in \mathcal{K},$$

or, equivalently,

$$(b - A\tilde{x}, v) = 0, \quad \forall v \quad \in \mathcal{K},$$

which is the Galerkin condition defining an orthogonal projection process for the approximation $\tilde{x}$. ∎

We now take up the case when $\mathcal{L}$ is defined by $\mathcal{L} = A\mathcal{K}$.

PROPOSITION 5.3 *Let A be an arbitrary square matrix and assume that $\mathcal{L} = A\mathcal{K}$. Then a vector $\tilde{x}$ is the result of an (oblique) projection method onto $\mathcal{K}$ orthogonally to $\mathcal{L}$ with the starting vector x_0 if and only if it minimizes the 2-norm of the residual vector $b - Ax$ over $x \in x_0 + \mathcal{K}$, i.e., if and only if*

$$R(\tilde{x}) = \min_{x \in x_0 + \mathcal{K}} R(x),$$

where $R(x) \equiv \|b - Ax\|_2$.

Proof. As was seen in Section 1.12.4, for $\tilde{x}$ to be the minimizer of $R(x)$, it is necessary and sufficient that $b - A\tilde{x}$ be orthogonal to all vectors of the form $v = Ay$, where y belongs to $\mathcal{K}$, i.e.,

$$(b - A\tilde{x}, v) = 0, \quad \forall v \quad \in A\mathcal{K},$$

which is precisely the Petrov-Galerkin condition that defines the approximate solution $\tilde{x}$. ∎

It is worthwhile to point out that A need not be nonsingular in the above proposition. When A is singular there may be infinitely many vectors $\tilde{x}$ satisfying the optimality condition.

5.2.2 INTERPRETATION IN TERMS OF PROJECTORS

We now return to the two important particular cases singled out in the previous section, namely, the cases $\mathcal{L} = \mathcal{K}$ and $\mathcal{L} = A\mathcal{K}$. In these cases, the result of the projection process can be interpreted easily in terms of actions of orthogonal projectors on the initial residual or initial error. Consider the second case first, as it is slightly simpler. Let r_0 be the initial residual $r_0 = b - Ax_0$, and $\tilde{r} = b - A\tilde{x}$ the residual obtained after the projection process with $\mathcal{L} = A\mathcal{K}$. Then,

$$\tilde{r} = b - A(x_0 + \delta) = r_0 - A\delta. \tag{5.9}$$

In addition, δ is obtained by enforcing the condition that $r_0 - A\delta$ be orthogonal to $A\mathcal{K}$. Therefore, the vector $A\delta$ is *the orthogonal projection of the vector r_0 onto the subspace $A\mathcal{K}$*. This is illustrated in Figure 5.2. Hence, the following proposition can be stated.

PROPOSITION 5.4 *Let $\tilde{x}$ be the approximate solution obtained from a projection process onto $\mathcal{K}$ orthogonally to $\mathcal{L} = A\mathcal{K}$, and let $\tilde{r} = b - A\tilde{x}$ be the associated residual. Then,*

$$\tilde{r} = (I - P)r_0, \tag{5.10}$$

where P denotes the orthogonal projector onto the subspace $A\mathcal{K}$.

A result of the proposition is that the 2-norm of the residual vector obtained after one projection step will not exceed the initial 2-norm of the residual, i.e.,

$$\|\tilde{r}\|_2 \leq \|r_0\|_2,$$

a result which has been established already. This class of methods may be termed *residual projection* methods.

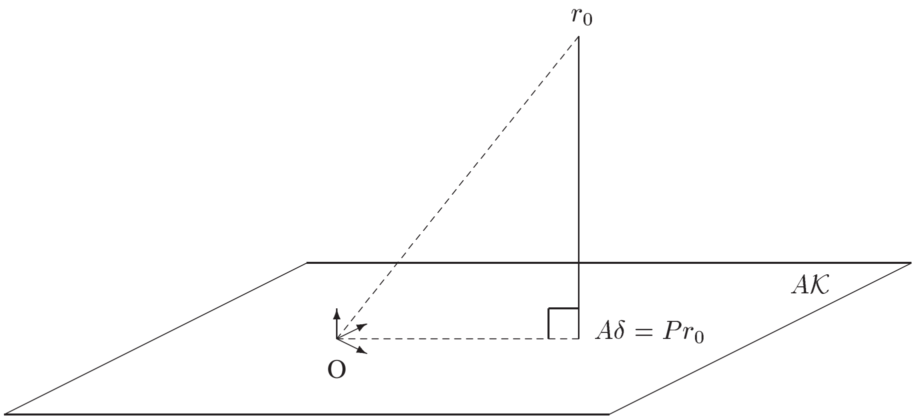


Figure 5.2 *Interpretation of the projection process for the case when $\mathcal{L} = A\mathcal{K}$.*

Now consider the case where $\mathcal{L} = \mathcal{K}$ and A is Symmetric Positive Definite. Let $d_0 = x_* - x_0$ be the initial error, where x_* denotes the exact solution to the system and, similarly, let $\tilde{d} = x_* - \tilde{x}$ where $\tilde{x} = x_0 + \delta$ is the approximate solution resulting from the projection step. Then (5.9) yields the relation

$$A\tilde{d} = \tilde{r} = A(d_0 - \delta),$$

where δ is now obtained by constraining the residual vector $r_0 - A\delta$ to be orthogonal to $\mathcal{K}$:

$$(r_0 - A\delta, w) = 0, \quad \forall\, w \in \mathcal{K}.$$

The above condition is equivalent to

$$(A(d_0 - \delta), w) = 0, \quad \forall\, w \in \mathcal{K}.$$

Since A is SPD, it defines an inner product (see Section 1.11) which is usually denoted by $(.,.)_A$ and the above condition becomes

$$(d_0 - \delta,\ w)_A = 0, \quad \forall\, w \in \mathcal{K}.$$

The above condition is now easy to interpret: *The vector δ is the A-orthogonal projection of the initial error d_0 onto the subspace $\mathcal{K}$.*

PROPOSITION 5.5 *Let $\tilde{x}$ be the approximate solution obtained from an orthogonal projection process onto $\mathcal{K}$ and let $\tilde{d} = x_* - \tilde{x}$ be the associated error vector. Then,*

$$\tilde{d} = (I - P_A)d_0,$$

where P_A denotes the projector onto the subspace $\mathcal{K}$, which is orthogonal with respect to the A-inner product.

A result of the proposition is that the A-norm of the error vector obtained after one projection step does not exceed the initial A-norm of the error, i.e.,

$$\|\tilde{d}\|_A \leq \|d_0\|_A,$$

which is expected because it is known that the A-norm of the error is minimized in $x_0 + \mathcal{K}$. This class of methods may be termed *error projection methods*.

5.2.3 GENERAL ERROR BOUND

If no vector of the subspace $\mathcal{K}$ comes close to the exact solution x, then it is impossible to find a good approximation $\tilde{x}$ to x from $\mathcal{K}$. Therefore, the approximation obtained by any projection process based on $\mathcal{K}$ will be poor. On the other hand, if there is some vector in $\mathcal{K}$ which is a small distance ϵ away from x, then the question is: How good can the approximate solution be? The purpose of this section is to try to answer this question.

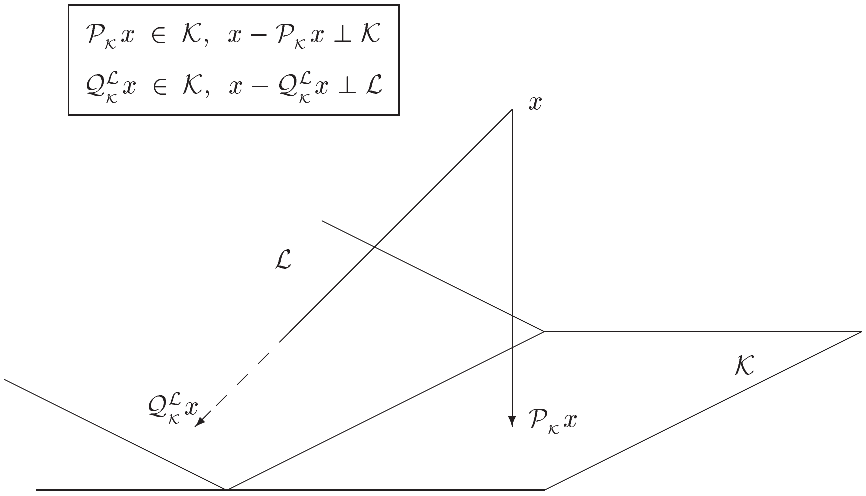


Figure 5.3 *Orthogonal and oblique projectors.*

Let $\mathcal{P}_{\mathcal{K}}$ be the orthogonal projector onto the subpace $\mathcal{K}$ and let $\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}}$ be the (oblique) projector onto $\mathcal{K}$ and orthogonally to $\mathcal{L}$. These projectors are defined by

$$\mathcal{P}_{\mathcal{K}} x \in \mathcal{K}, \; x - \mathcal{P}_{\mathcal{K}} x \perp \mathcal{K},$$
$$\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} x \in \mathcal{K}, \; x - \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} x \perp \mathcal{L},$$

and are illustrated in Figure 5.3. The symbol A_m is used to denote the operator

$$A_m = \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A \mathcal{P}_{\mathcal{K}},$$

and it is assumed, without loss of generality, that $x_0 = 0$. Then according to the property (1.54), the approximate problem defined in (5.5 – 5.6) can be reformulated as follows: find $\tilde{x} \in \mathcal{K}$ such that

$$\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} (b - A\tilde{x}) = 0,$$

or, equivalently,

$$A_m \tilde{x} = \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} b, \quad \tilde{x} \in \mathcal{K}.$$

Thus, an n-dimensional linear system is approximated by an m-dimensional one.

The following proposition examines what happens in the particular case when the subspace $\mathcal{K}$ is invariant under A. This is a rare occurrence in practice, but the result helps in understanding the breakdown behavior of the methods to be considered in later chapters.

PROPOSITION 5.6 *Assume that $\mathcal{K}$ is invariant under A, $x_0 = 0$, and b belongs to $\mathcal{K}$. Then the approximate solution obtained from any (oblique or orthogonal) projection method onto $\mathcal{K}$ is exact.*

Proof. An approximate solution $\tilde{x}$ is defined by

$$\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}}(b - A\tilde{x}) = 0,$$

where $\tilde{x}$ is a nonzero vector in $\mathcal{K}$. The right-hand side b is in $\mathcal{K}$, so we have $\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} b = b$. Similarly, $\tilde{x}$ belongs to $\mathcal{K}$ which is invariant under A, and therefore, $\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A\tilde{x} = A\tilde{x}$. Then the above equation becomes

$$b - A\tilde{x} = 0,$$

showing that $\tilde{x}$ is an exact solution. ∎

The result can be extended trivially to the case where $x_0 \neq 0$. The required assumption in this case is that the initial residual $r_0 = b - Ax_0$ belongs to the invariant subspace $\mathcal{K}$.

An important quantity for the convergence properties of projection methods is the distance $\|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2$ of the exact solution x_* from the subspace $\mathcal{K}$. This quantity plays a key role in the analysis of projection methods. Note that the solution x_* cannot be well approximated from $\mathcal{K}$, if $\|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2$ is not small because

$$\|\tilde{x} - x_*\|_2 \geq \|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2.$$

The fundamental quantity $\|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2 / \|x_*\|_2$ is the *sine* of the acute angle between the solution x_* and the subspace $\mathcal{K}$. The following theorem establishes an upper bound for the residual norm of the *exact* solution with respect to the approximate operator A_m.

THEOREM 5.1 *Let $\gamma = \|\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A(I - \mathcal{P}_{\mathcal{K}})\|_2$ and assume that b is a member of $\mathcal{K}$ and $x_0 = 0$. Then the exact solution x_* of the original problem is such that*

$$\|b - A_m x_*\|_2 \leq \gamma \|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2. \tag{5.11}$$

Proof. Since $b \in \mathcal{K}$, then

$$\begin{aligned} b - A_m x_* &= \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}}(b - A\mathcal{P}_{\mathcal{K}} x_*) \\ &= \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}}(Ax_* - A\mathcal{P}_{\mathcal{K}} x_*) \\ &= \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A(x_* - \mathcal{P}_{\mathcal{K}} x_*) \\ &= \mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A(I - \mathcal{P}_{\mathcal{K}})x_*. \end{aligned}$$

Noting that $I - \mathcal{P}_{\mathcal{K}}$ is a projector, it follows that

$$\begin{aligned}\|b - A_m x_*\|_2 &= \|\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A(I - \mathcal{P}_{\mathcal{K}})(I - \mathcal{P}_{\mathcal{K}})x_*\|_2 \\ &\leq \|\mathcal{Q}_{\mathcal{K}}^{\mathcal{L}} A(I - \mathcal{P}_{\mathcal{K}})\|_2 \|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2,\end{aligned}$$

which completes the proof. ■

It is useful to consider a matrix interpretation of the theorem. We consider only the particular case of orthogonal projection methods ($\mathcal{L} = \mathcal{K}$). Assume that V is unitary, i.e., that the basis $\{v_1, \ldots, v_m\}$ is orthonormal, and that $W = V$. Observe that $b = VV^T b$. Equation (5.11) can be represented in the basis V as

$$\|b - V(V^T AV)V^T x_*\|_2 \leq \gamma \|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2.$$

However,

$$\begin{aligned}\|b - V(V^T AV)V^T x_*\|_2 &= \|V(V^T b - (V^T AV)V^T x_*\|_2 \\ &= \|V^T b - (V^T AV)V^T x_*\|_2.\end{aligned}$$

Thus, the projection of the exact solution has a residual norm with respect to the matrix $B = V^T AV$, which is of the order of $\|(I - \mathcal{P}_{\mathcal{K}})x_*\|_2$.

5.3 ONE-DIMENSIONAL PROJECTION PROCESSES

This section examines simple examples provided by one-dimensional projection processes. In what follows, the vector r denotes the residual vector $r = b - Ax$ for the current approximation x. To avoid subscripts, arrow notation is used to denote *vector updates*. Thus, "$x \leftarrow x + \alpha r$" means "compute $x + \alpha r$ and overwrite the result on the current x." (This is known as a SAXPY operation.)

One-dimensional projection processes are defined when

$$\mathcal{K} = span\{v\} \quad \text{and} \quad \mathcal{L} = span\{w\},$$

where v and w are two vectors. In this case, the new approximation takes the form $x \leftarrow x + \alpha v$ and the Petrov-Galerkin condition $r - A\delta \perp w$ yields

$$\alpha = \frac{(r, w)}{(Av, w)}. \tag{5.12}$$

Following are three popular choices to be considered.

5.3.1 STEEPEST DESCENT

The steepest descent algorithm is defined for the case where the matrix A is Symmetric Positive Definite. It consists of taking at each step $v = r$ and $w = r$. This yields an iteration described by the following algorithm.

ALGORITHM 5.2: Steepest Descent Algorithm

1. *Until convergence, Do:*
2. $r \leftarrow b - Ax$
3. $\alpha \leftarrow (r, r)/(Ar, r)$
4. $x \leftarrow x + \alpha r$
5. *EndDo*

Each step of the above iteration minimizes

$$f(x) = \|x - x_*\|_A^2 = (A(x - x_*), (x - x_*)),$$

over all vectors of the form $x + \alpha d$, where d is the negative of the gradient direction $-\nabla f$. The negative of the gradient direction is *locally* the direction that yields the fastest rate of decrease for f. Next, we prove that convergence is guaranteed when A is SPD. The result is a consequence of the following lemma known as the Kantorovich inequality.

LEMMA 5.1 ***(Kantorovich inequality)*** *Let B be any Symmetric Positive Definite real matrix and λ_{max}, λ_{min} its largest and smallest eigenvalues. Then,*

$$\frac{(Bx, x)(B^{-1}x, x)}{(x, x)^2} \le \frac{(\lambda_{max} + \lambda_{min})^2}{4\,\lambda_{max}\lambda_{min}}, \quad \forall x \neq 0. \tag{5.13}$$

Proof. Clearly, it is equivalent to show that the result is true for any unit vector x. Since B is symmetric, it is unitarily similar to a diagonal matrix, $B = Q^T DQ$, and

$$\begin{aligned}(Bx, x)(B^{-1}x, x) &= (Q^T DQx, x)(Q^T D^{-1}Qx, x) \\ &= (DQx, Qx)(D^{-1}Qx, Qx).\end{aligned}$$

Setting $y = Qx = (y_1, \ldots, y_n)^T$, and $\beta_i = y_i^2$, note that

$$\lambda \equiv (Dy, y) = \sum_{i=1}^{n} \beta_i \lambda_i$$

is a convex combination of the eigenvalues $\lambda_i, i = 1, \ldots, n$. The following relation holds,

$$(Bx, x)(B^{-1}x, x) = \lambda\psi(y)$$

with

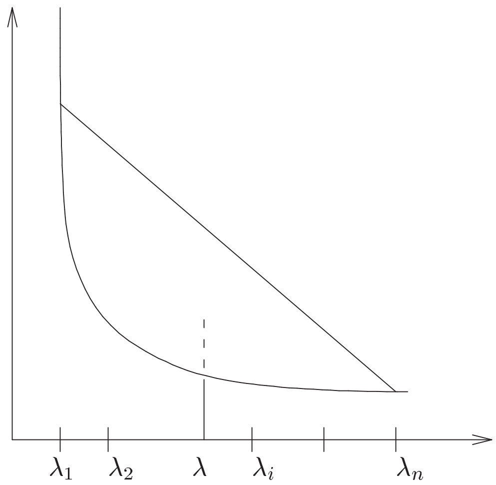
$$\psi(y) = (D^{-1}y, y) = \sum_{i=1}^{n} \beta_i \frac{1}{\lambda_i}.$$

Noting that the function $1/x$ is convex, $\psi(y)$ is bounded from above by the linear curve that joins the points $(\lambda_1, 1/\lambda_1)$ and $(\lambda_n, 1/\lambda_n)$, i.e.,

$$\psi(y) \le \frac{1}{\lambda_1} + \frac{1}{\lambda_n} - \frac{\lambda}{\lambda_1\lambda_n}.$$

Therefore,

$$(Bx, x)(B^{-1}x, x) = \lambda\psi(y) \le \lambda\left(\frac{1}{\lambda_1} + \frac{1}{\lambda_n} - \frac{\lambda}{\lambda_1\lambda_n}\right).$$



The maximum of the right-hand side is reached for $\lambda = \frac{1}{2}(\lambda_1 + \lambda_n)$ yielding,

$$(Bx, x)(B^{-1}x, x) = \lambda\psi(y) \le \frac{(\lambda_1 + \lambda_n)^2}{4\lambda_1 \lambda_n}$$

which gives the desired result. ■

This lemma helps to establish the following result regarding the convergence rate of the method.

THEOREM 5.2 *Let A be a Symmetric Positive Definite matrix. Then, the A-norms of the error vectors $d_k = x_* - x_k$ generated by Algorithm 5.2 satisfy the relation*

$$\|d_{k+1}\|_A \le \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \|d_k\|_A, \tag{5.14}$$

and Algorithm 5.2 converges for any initial guess x_0.

Proof. Start by observing that $\|d_{k+1}\|_A^2 = (Ad_{k+1}, d_{k+1}) = (r_{k+1}, d_{k+1})$ and then by simple substitution,

$$\|d_{k+1}\|_A^2 = (r_{k+1}, d_k - \alpha_k r_k)$$

Since by construction the new residual vector r_{k+1} must be orthogonal to the search direction r_k, the second term in the right-hand side of the above equation is zero. Thus,

$$\|d_{k+1}\|_A^2 = (r_k - \alpha_k A r_k, d_k) \tag{5.15}$$

$$= (r_k, A^{-1} r_k) - \alpha_k (r_k, r_k) \tag{5.16}$$

$$= \|d_k\|_A^2 \left(1 - \frac{(r_k, r_k)}{(r_k, Ar_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)}\right). \tag{5.17}$$

The result follows by applying the Kantorovich inequality (5.13). ■

5.3.2 MINIMAL RESIDUAL (MR) ITERATION

We now assume that A is not necessarily symmetric but only positive definite, i.e., its symmetric part $A + A^T$ is Symmetric Positive Definite. Taking at each step $v = r$ and $w = Ar$, the following iterative procedure results.

ALGORITHM 5.3: Minimal Residual Iteration

1. *Until convergence, Do:*
2. $\quad r \leftarrow b - Ax$
3. $\quad \alpha \leftarrow (Ar, r)/(Ar, Ar)$
4. $\quad x \leftarrow x + \alpha r$
5. *EndDo*

Here, each step minimizes $f(x) = \|b - Ax\|_2^2$ in the direction r. The iteration converges under the condition that A is positive definite as is stated in the next theorem.

THEOREM 5.3 *Let A be a real positive definite matrix, and let*

$$\mu = \lambda_{min}(A + A^T)/2, \quad \sigma = \|A\|_2.$$

Then the residual vectors generated by Algorithm 5.3 satisfy the relation

$$\|r_{k+1}\|_2 \le \left(1 - \frac{\mu^2}{\sigma^2}\right)^{1/2} \|r_k\|_2 \tag{5.18}$$

and Algorithm (5.3) converges for any initial guess x_0.

Proof. We proceed similarly to the steepest descent method, starting with the relation

$$\begin{aligned} \|r_{k+1}\|_2^2 &= (r_k - \alpha_k A r_k, r_k - \alpha_k A r_k) &(5.19)\\ &= (r_k - \alpha_k A r_k, r_k) - \alpha_k (r_k - \alpha_k A r_k, A r_k). &(5.20)\end{aligned}$$

By construction, the new residual vector $r_k - \alpha_k A r_k$ must be orthogonal to the search direction Ar_k, and, as a result, the second term in the right-hand side of the above equation vanishes and we obtain

$$\begin{aligned} \|r_{k+1}\|_2^2 &= (r_k - \alpha_k A r_k, r_k)\\ &= (r_k, r_k) - \alpha_k (A r_k, r_k)\\ &= \|r_k\|_2^2 \left(1 - \frac{(Ar_k, r_k)}{(r_k, r_k)} \frac{(Ar_k, r_k)}{(Ar_k, Ar_k)}\right) \qquad (5.21)\\ &= \|r_k\|_2^2 \left(1 - \frac{(Ar_k, r_k)^2}{(r_k, r_k)^2} \frac{\|r_k\|_2^2}{\|Ar_k\|_2^2}\right). \end{aligned}$$

From Theorem 1.19, it can be stated that

$$\frac{(Ax, x)}{(x, x)} \ge \mu > 0, \tag{5.22}$$

where $\mu = \lambda_{min}(A + A^T)/2$. The desired result follows immediately by using the inequality $\|Ar_k\|_2 \le \|A\|_2 \, \|r_k\|_2$. ∎

There are alternative ways of obtaining inequalities that prove convergence. For example, starting from (5.21), (5.22) can be used again for the term $(Ar_k, r_k)/(r_k, r_k)$ and similarly, we can write

$$\frac{(Ax, x)}{(Ax, Ax)} = \frac{(Ax, A^{-1}(Ax))}{(Ax, Ax)} \ge \lambda_{min}\left(\frac{A^{-1} + A^{-T}}{2}\right) \; > \; 0,$$

since A^{-1} is also positive definite. This would yield the inequality

$$\|r_{k+1}\|_2^2 \le \left(1 - \mu(A)\mu(A^{-1})\right) \|r_k\|_2^2, \tag{5.23}$$

in which $\mu(B) = \lambda_{min}(B + B^T)/2$.

Another interesting observation is that if we define

$$\cos \angle_k = \frac{(Ar_k, r_k)}{\|Ar_k\|_2 \, \|r_k\|_2},$$

then (5.21) can be rewritten as

$$\begin{aligned} \|r_{k+1}\|_2^2 &= \|r_k\|_2^2 \left(1 - \frac{(Ar_k, r_k)}{(Ar_k, Ar_k)} \frac{(Ar_k, r_k)}{(r_k, r_k)}\right) \\ &= \|r_k\|_2^2 \left(1 - \cos^2 \angle_k\right) \\ &= \|r_k\|_2^2 \, \sin^2 \angle_k. \end{aligned}$$

At each step the reduction in the residual norm is equal to the *sine* of the acute angle between r and Ar. The convergence factor is therefore bounded by

$$\rho = \max_{x \,\in\, \mathbb{R}^n,\; x \neq 0} \quad \sin \angle(x, Ax),$$

in which $\angle(x, Ax)$ is the acute angle between x and Ax. The maximum angle $\angle(x, Ax)$ is guaranteed to be less than $\pi/2$ when A is positive definite as the above results show.

5.3.3 RESIDUAL NORM STEEPEST DESCENT

In the residual norm steepest descent algorithm, the assumption that A is positive definite is relaxed. In fact, the only requirement is that A is a (square) nonsingular matrix. At each step the algorithm uses $v = A^T r$ and $w = Av$, giving the following sequence of operations:

$$\begin{aligned} & r \leftarrow b - Ax, v = A^T r, \\ & \alpha \leftarrow \|v\|_2^2 / \|Av\|_2^2, \\ & x \leftarrow x + \alpha v. \end{aligned} \tag{5.24}$$

However, an algorithm based on the above sequence of operations would require three matrix-by-vector products, which is three times as many as the other algorithms seen in this section. The number of matrix-by-vector operations can be reduced to two per step by computing the residual differently. This variant is as follows.

ALGORITHM 5.4: Residual Norm Steepest Descent

```
Compute r := b − Ax
Until convergence, Do:
    v := A^T r
    Compute Av and α := ‖v‖₂² / ‖Av‖₂²
    x := x + αv
    r := r − αAv
EndDo
```

Here, each step minimizes $f(x) = \|b - Ax\|_2^2$ in the direction $-\nabla f$. As it turns out, this is equivalent to the steepest descent algorithm of Section 5.3.1 applied to the normal equations $A^T Ax = A^T b$. Since $A^T A$ is positive definite when A is nonsingular, then, according to Theorem 5.2, the method will converge whenever A is nonsingular.

5.4 ADDITIVE AND MULTIPLICATIVE PROCESSES

We begin by considering again the block relaxation techniques seen in the previous chapter. To define these techniques, a *set-decomposition* of $S = \{1, 2, \ldots, n\}$ is considered as the definition of p subsets $S_1, \ldots, S_p$ of S with

$$S_i \subseteq S, \qquad \bigcup_{i=1,\cdots,p} S_i = S.$$

Denote by n_i the size of S_i and define the subset S_i as

$$S_i = \{m_i(1), m_i(2), \ldots, m_i(n_i)\}.$$

Let V_i be the $n \times n_i$ matrix

$$V_i = [e_{m_i(1)}, e_{m_i(2)}, \ldots, e_{m_i(n_i)}],$$

where each e_j is the j-th column of the $n \times n$ identity matrix.

If the block Jacobi and block Gauss-Seidel algorithms, Algorithms 4.1 and 4.2, are examined carefully, it can be observed that each individual step in the main loop (lines 2 to 5) represents an orthogonal projection process over $K_i = \mathrm{span}\{V_i\}$. Indeed, the equation (4.17) is exactly (5.7) with $W = V = V_i$. This individual projection step modifies only the components corresponding to the subspace K_i. However, the general block Jacobi iteration combines these modifications, implicitly adding them together, to obtain the next iterate x_{k+1}. Borrowing from the terminology of domain decomposition techniques, this will be called an *additive projection procedure*. Generally, an additive projection procedure can be defined for any sequence of subspaces K_i, not just subspaces spanned by the columns of the identity matrix. The only requirement is that the subspaces K_i should be distinct, although they are allowed to overlap.

Let a sequence of p orthogonal systems V_i be given, with the condition that $\mathrm{span}\{V_i\}$

$\neq \operatorname{span}\{V_j\}$ for $i \neq j$, and define

$$A_i = V_i^T A V_i.$$

The additive projection procedure can be written as

$$\begin{aligned} y_i &= A_i^{-1} V_i^T (b - A x_k), \quad i = 1, \ldots, p\,, \\ x_{k+1} &= x_k + \sum_{i=1}^{p} V_i y_i, \end{aligned} \tag{5.25}$$

which leads to the following algorithm.

ALGORITHM 5.5: Additive Projection Procedure

1. *For* $k = 0, 1, \ldots,$ *until convergence, Do:*
2. *For* $i = 1, 2, \ldots, p$ *Do:*
3. *Solve* $A_i y_i = V_i^T (b - A x_k)$
4. *EndDo*
5. *Set* $x_{k+1} = x_k + \sum_{i=1}^{p} V_i y_i$
6. *EndDo*

Defining $r_k = b - A x_k$, the residual vector at step k, then clearly

$$\begin{aligned} r_{k+1} &= b - A x_{k+1} \\ &= b - A x_k - \sum_{i=1}^{p} A V_i \left(V_i^T A V_i\right)^{-1} V_i^T r_k \\ &= \left[I - \sum_{i=1}^{p} A V_i \left(V_i^T A V_i\right)^{-1} V_i^T \right] r_k. \end{aligned}$$

Observe that each of the p operators

$$P_i = A V_i \left(V_i^T A V_i\right)^{-1} V_i^T$$

represents the projector onto the subspace spanned by AV_i, and orthogonal to V_i. Often, the additive processes are used in conjunction with an acceleration parameter ω, thus (5.25) is replaced by

$$\begin{aligned} y_i &= A_i^{-1} V_i^T (b - A x_k), \quad i = 1, \ldots, p\,, \\ x_{k+1} &= x_k + \omega \sum_{i=1}^{p} V_i y_i. \end{aligned}$$

Even more generally, a different parameter ω_i can be used for each projection, i.e.,

$$\begin{aligned} y_i &= A_i^{-1} V_i^T (b - A x_k), \quad i = 1, \ldots, p, \\ x_{k+1} &= x_k + \sum_{i=1}^{p} \omega_i V_i y_i. \end{aligned}$$

The residual norm in this situation is given by

$$r_{k+1} = \left(I - \sum_{i=1}^{p} \omega_i P_i \right) r_k, \tag{5.26}$$

considering the single ω parameter as a particular case. Exercise 14 gives an example of the choice of ω_i which has the effect of producing a sequence with decreasing residual norms.

We now return to the generic case, where $\omega_i = 1, \; \forall i$. A least-squares option can be defined by taking for each of the subproblems $L_i = AK_i$. In this situation, P_i becomes an orthogonal projector onto AK_i, since

$$P_i = AV_i \left((AV_i)^T AV_i \right)^{-1} (AV_i)^T.$$

It is interesting to note that the residual vector obtained after one outer loop is related to the previous residual by

$$r_{k+1} = \left(I - \sum_{i=1}^{p} P_i \right) r_k,$$

where the P_i's are now orthogonal projectors. In particular, in the ideal situation when the AV_i's are orthogonal to each other, and the total rank of the P_i's is n, then the exact solution would be obtained in one outer step, since in this situation

$$I - \sum_{i=1}^{p} P_i = 0.$$

Thus, the maximum reduction in the residual norm is achieved when the V_i's are A-orthogonal to one another.

Similar to the Jacobi and Gauss-Seidel iterations, what distinguishes the additive and multiplicative iterations is that the latter updates the component to be corrected at step i immediately. Then this updated approximate solution is used to compute the residual vector needed to correct the next component. The Jacobi iteration uses the same previous approximation x_k to update all the components of the solution. Thus, the analogue of the block Gauss-Seidel iteration can be defined as follows.

ALGORITHM 5.6: Multiplicative Projection Procedure

1. *Until convergence, Do:*
2.   *For* $i = 1, 2, \ldots, p$ *Do:*
3.   *Solve* $A_i y = V_i^T (b - Ax)$
4.   *Set* $x := x + V_i y$
5.   *EndDo*
6. *EndDo*

EXERCISES

1. Consider the linear system $Ax = b$, where A is a Symmetric Positive Definite matrix.

a. Consider the sequence of one-dimensional projection processes with $\mathcal{K} = \mathcal{L} = \operatorname{span}\{e_i\}$, where the sequence of indices i is selected in any fashion. Let x_{new} be a new iterate after one projection step from x and let $r = b - Ax$, $d = A^{-1}b - x$, and $d_{new} = A^{-1}b - x_{new}$. Show that

$$(Ad_{new}, d_{new}) = (Ad, d) - (r, e_i)^2/a_{ii}.$$

Does this equality, as is, establish convergence of the algorithm?

b. Assume now that i is selected at each projection step to be the index of a component of largest absolute value in the current residual vector $r = b - Ax$. Show that

$$\|d_{new}\|_A \le \left(1 - \frac{1}{n\kappa^2(A)}\right)^{1/2} \|d\|_A,$$

in which $\kappa(A)$ is the spectral condition number of A. [Hint: Use the inequality $|e_i^T r| \ge n^{-1/2}\|r\|_2$.] Does this prove that the algorithm converges?

2. Consider the linear system $Ax = b$, where A is a Symmetric Positive Definite matrix. Consider a projection step with $\mathcal{K} = \mathcal{L} = \operatorname{span}\{v\}$ where v is some nonzero vector. Let x_{new} be the new iterate after one projection step from x and let $d = A^{-1}b - x$, and $d_{new} = A^{-1}b - x_{new}$.

a. Show that

$$(Ad_{new}, d_{new}) = (Ad, d) - (r, v)^2/(Av, v).$$

Does this equality establish convergence of the algorithm?

b. In Gastinel's method, the vector v is selected in such a way that $(v, r) = \|r\|_1$, e.g., by defining the components of v to be $v_i = \operatorname{sign}(e_i^T r)$, where $r = b - Ax$ is the current residual vector. Show that

$$\|d_{new}\|_A \le \left(1 - \frac{1}{n\kappa^2(A)}\right)^{1/2} \|d\|_A,$$

in which $\kappa(A)$ is the spectral condition number of A. Does this prove that the algorithm converges?

c. Compare the cost of one step of this method with that of cyclic Gauss-Seidel (see Example 5.1) and that of "optimal" Gauss-Seidel where at each step $\mathcal{K} = \mathcal{L} = \operatorname{span}\{e_i\}$ and i is a component of largest magnitude in the current residual vector.

3. In Section 5.3.3, it was shown that taking a one-dimensional projection technique with $\mathcal{K} = \operatorname{span}\,\{A^T r\}$ and $\mathcal{L} = \operatorname{span}\{AA^T r\}$ is mathematically equivalent to using the usual steepest descent algorithm applied to the normal equations $A^T Ax = A^T b$. Show that an *orthogonal* projection method for $A^T Ax = A^T b$ using a subspace $\mathcal{K}$ is mathematically equivalent to applying a projection method onto $\mathcal{K}$, orthogonally to $\mathcal{L} = A\mathcal{K}$ for solving the system $Ax = b$.

4. Consider the matrix

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

a. Find a rectangle or square in the complex plane which contains all the eigenvalues of A, without computing the eigenvalues.

b. Is the Minimal Residual iteration guaranteed to converge for a linear system with the matrix A?

5. Consider the linear system

$$\begin{pmatrix} D_1 & -F \\ -E & -D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

in which D_1 and D_2 are both nonsingular matrices of size m each.

a. Define an orthogonal projection method using the set of vectors $e_1, \ldots, e_m$, i.e., $\mathcal{L} = \mathcal{K} = \text{span}\{e_1, \ldots, e_m\}$. Write down the corresponding projection step (x_1 is modified into $\tilde{x}_1$). Similarly, write the projection step for the second half of the vectors, i.e., when $\mathcal{L} = \mathcal{K} = \text{span}\{e_{m+1}, \ldots, e_n\}$.

b. Consider an iteration procedure which consists of performing the two successive half-steps described above until convergence. Show that this iteration is equivalent to a (standard) Gauss-Seidel iteration applied to the original system.

c. Now consider a similar idea in which $\mathcal{K}$ is taken to be the same as before for each half-step and $\mathcal{L} = A\mathcal{K}$. Write down the iteration procedure based on this approach. Name another technique to which it is mathematically equivalent.

6. Consider the linear system $Ax = b$, where A is a Symmetric Positive Definite matrix. We define a projection method which uses a two-dimensional space at each step. At a given step, take $\mathcal{L} = \mathcal{K} = \text{span}\{r, Ar\}$, where $r = b - Ax$ is the current residual.

a. For a basis of $\mathcal{K}$ use the vector r and the vector p obtained by orthogonalizing Ar against r with respect to the A-inner product. Give the formula for computing p (no need to normalize the resulting vector).

b. Write the algorithm for performing the projection method described above.

c. Will the algorithm converge for any initial guess x_0? Justify the answer. [Hint: Exploit the convergence results for one-dimensional projection techniques.]

7. Consider projection methods which update at each step the current solution with linear combinations from two directions: the current residual r and Ar.

a. Consider an orthogonal projection method, i.e., at each step $\mathcal{L} = \mathcal{K} = \text{span}\{r, Ar\}$. Assuming that A is Symmetric Positive Definite, establish convergence of the algorithm.

b. Consider a least-squares projection method in which at each step $\mathcal{K} = \text{span}\{r, Ar\}$ and $\mathcal{L} = A\mathcal{K}$. Assuming that A is positive definite (not necessarily symmetric), establish convergence of the algorithm.

[Hint: The convergence results for any of the one-dimensional projection techniques can be exploited.]

8. The "least-squares" Gauss-Seidel relaxation method defines a relaxation step as $x_{new} = x + \delta\, e_i$ (same as Gauss-Seidel), but chooses δ to minimize the residual norm of x_{new}.

a. Write down the resulting algorithm.

b. Show that this iteration is mathematically equivalent to a Gauss-Seidel iteration applied to the normal equations $A^T Ax = A^T b$.

9. Derive three types of one-dimensional projection algorithms in the same manner as was done in Section 5.3, by replacing every occurrence of the residual vector r by a vector e_i, a column of the identity matrix.

10. Derive three types of one-dimensional projection algorithms in the same manner as was done in Section 5.3, by replacing every occurrence of the residual vector r by a vector Ae_i, a column of the matrix A. What would be an "optimal" choice for i at each projection step? Show that the method is globally convergent in this case.

11. A minimal residual iteration as defined in Section 5.3.2 can also be defined for an arbitrary search direction d, not necessarily related to r in any way. In this case, we still define $e = Ad$.

a. Write down the corresponding algorithm.

b. Under which condition are all iterates defined?

c. Under which condition on d does the new iterate make no progress, i.e., $\|r_{k+1}\|_2 = \|r_k\|_2$?

d. Write a general sufficient condition which must be satisfied by d at each step in order to guarantee convergence.

12. Consider the following real-valued functions of the vector variable x, where A and b are the coefficient matrix and right-hand system of a given linear system $Ax = b$ and $x_* = A^{-1}b$.

$$\begin{aligned} a(x) &= \|x_* - x\|_2^2, \\ f(x) &= \|b - Ax\|_2^2, \\ g(x) &= \|A^T b - A^T Ax\|_2^2, \\ h(x) &= 2(b, x) - (Ax, x). \end{aligned}$$

a. Calculate the gradients of all four functions above.

b. How is the gradient of g related to that of f?

c. How is the gradient of f related to that of h when A is symmetric?

d. How does the function h relate to the A-norm of the error $x_* - x$ when A is Symmetric Positive Definite?

13. The block Gauss-Seidel iteration can be expressed as a method of successive projections. The subspace $\mathcal{K}$ used for each projection is of the form

$$\mathcal{K} = \text{span}\{e_i, e_{i+1}, \ldots, e_{i+p}\}.$$

What is $\mathcal{L}$? Not too commonly used an alternative is to take $\mathcal{L} = A\mathcal{K}$, which amounts to solving a least-squares problem instead of a linear system. Develop algorithms for this case. What are the advantages and disadvantages of the two approaches (ignoring convergence rates)?

14. Let the scalars ω_i in the additive projection procedure satisfy the constraint

$$\sum_{i=1}^{p} \omega_i = 1. \tag{5.27}$$

It is not assumed that each ω_i is positive but only that $|\omega_i| \le 1$ for all i. The residual vector is given by the Formula (5.26) or, equivalently,

$$r_{k+1} = \sum_{i=1}^{p} \omega_i (I - P_i) r_k.$$

a. Show that in the least-squares case, we have $\|r_{k+1}\|_2 \le \|r_k\|_2$ for any choice of ω_i's which satisfy the constraint (5.27).

b. We wish to choose a set of ω_i's such that the 2-norm of the residual vector r_{k+1} is minimal. Determine this set of ω_i's, assuming that the vectors $(I - P_i)r_k$ are all linearly independent.

c. The "optimal" ω_i's provided in the previous question require the solution of a $p \times p$ Symmetric Positive Definite linear system. Let $z_i \equiv V_i y_i$ be the "search directions" provided by each of the individual projection steps. To avoid this difficulty, a simpler strategy is used which consists of performing p successive minimal residual iterations along these search directions, as is described below.

$r := r_k$
For $i = 1, \ldots, p$ *Do:*
  $\omega_i := (r, Az_i)/(Az_i, Az_i)$
  $x := x + \omega_i z_i$
  $r := r - \omega_i A z_i$
EndDo

Show that $\|r_{k+1}\|_2 \le \|r_k\|_2$. Give a sufficient condition to ensure global convergence.

15. Consider the iteration: $x_{k+1} = x_k + \alpha_k d_k$, where d_k is a vector called the *direction of search*, and α_k is a scalar. It is assumed throughout that d_k is a nonzero vector. Consider a method which determines x_{k+1} so that the residual $\|r_{k+1}\|_2$ is the smallest possible.

a. Determine α_k so that $\|r_{k+1}\|_2$ is minimal.

b. Show that the residual vector r_{k+1} obtained in this manner is orthogonal to Ar_k.

c. Show that the residual vectors satisfy the relation:

$$\|r_{k+1}\|_2 \le \|r_k\|_2 \ \sin \angle(r_k, Ad_k).$$

d. Assume that at each step k, we have $(r_k, Ad_k) \neq 0$. Will the method always converge?

e. Now assume that A is positive definite and select at each step $d_k \equiv r_k$. Prove that the method will converge for any initial guess x_0.

16. Consider the iteration: $x_{k+1} = x_k + \alpha_k d_k$, where d_k is a vector called the *direction of search*, and α_k is a scalar. It is assumed throughout that d_k is a vector which is selected in the form $d_k = A^T f_k$ where f_k is some nonzero vector. Let $x_* = A^{-1}b$ be the exact solution. Now consider a method which at each step k determines x_{k+1} so that the error norm $\|x_* - x_{k+1}\|_2$ is the smallest possible.

a. Determine α_k so that $\|x_* - x_{k+1}\|_2$ is minimal and show that the error vector $e_{k+1} = x_* - x_{k+1}$ is orthogonal to d_k. The expression of α_k should not contain unknown quantities (e.g., x_* or e_k).

b. Show that $\|e_{k+1}\|_2 \le \|e_k\|_2 \ \sin \angle(e_k, d_k)$.

c. Establish the convergence of the algorithm for any x_0, when $f_k \equiv r_k$ for all k.

NOTES AND REFERENCES. Initially, the term *projection methods* was used mainly to describe one-dimensional techniques such as those presented in Section 5.3. An excellent account of what has been done in the late 1950s and early 1960s can be found in Householder's book [122] as well as Gastinel [101]. For more general, including nonlinear, projection processes, a good reference is Kranoselskii and co-authors [138].

Projection techniques are present in different forms in many other areas of scientific computing and can be formulated in abstract Hilbert functional spaces. The terms *Galerkin* and *Petrov-Galerkin* techniques are used commonly in finite element methods to describe projection methods on finite element spaces. The principles are identical to those seen in this chapter. ■

CHAPTER

6

KRYLOV SUBSPACE METHODS PART I

The next two chapters explore a few methods which are considered currently to be among the most important iterative techniques available for solving large linear systems. These techniques are based on projection processes, both orthogonal and oblique, onto Krylov subspaces, which are subspaces spanned by vectors of the form $p(A)v$ where p is a polynomial. In short, these techniques approximate $A^{-1}b$ by $p(A)b$, where p is a "good" polynomial. This chapter covers methods derived from, or related to, the Arnoldi orthogonalization. The next chapter covers methods based on Lanczos biorthogonalization.

6.1 INTRODUCTION

Recall from the previous chapter that a general *projection method* for solving the linear system

$$Ax = b, \tag{6.1}$$

is a method which seeks an approximate solution x_m from an affine subspace $x_0 + \mathcal{K}_m$ of dimension m by imposing the Petrov-Galerkin condition

$$b - Ax_m \perp \mathcal{L}_m,$$

where $\mathcal{L}_m$ is another subspace of dimension m. Here, x_0 represents an arbitrary initial guess to the solution. A Krylov subspace method is a method for which the subspace $\mathcal{K}_m$ is the Krylov subspace

$$\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\},$$

where $r_0 = b - Ax_0$. When there is no ambiguity, $\mathcal{K}_m(A, r_0)$ will be denoted by $\mathcal{K}_m$. The different versions of Krylov subspace methods arise from different choices of the subspace $\mathcal{L}_m$ and from the ways in which the system is *preconditioned*, a topic that will be covered in detail in later chapters.

Viewed from the angle of approximation theory, it is clear that the approximations obtained from a Krylov subspace method are of the form

$$A^{-1}b \approx x_m = x_0 + q_{m-1}(A)r_0,$$

in which q_{m-1} is a certain polynomial of degree $m-1$. In the simplest case where $x_0 = 0$, then

$$A^{-1}b \approx q_{m-1}(A)b.$$

In other words, $A^{-1}b$ is approximated by $q_{m-1}(A)b$.

Although all the techniques provide the same type of *polynomial* approximations, the choice of $\mathcal{L}_m$, i.e., the constraints used to build these approximations, will have an important effect on the iterative technique. Two broad choices for $\mathcal{L}_m$ give rise to the best-known techniques. The first is simply $\mathcal{L}_m = \mathcal{K}_m$ and the minimum-residual variation $\mathcal{L}_m = A\mathcal{K}_m$. A few of the numerous methods in this category will be described in this chapter. The second class of methods is based on defining $\mathcal{L}_m$ to be a Krylov subspace method associated with A^T, namely, $\mathcal{L}_m = \mathcal{K}_m(A^T, r_0)$. Methods of this class will be covered in the next chapter. There are also block extensions of each of these methods termed *block Krylov subspace methods*, which will be discussed only briefly. Note that a projection method may have several different implementations, giving rise to different algorithms which are all mathematically equivalent.

6.2 KRYLOV SUBSPACES

In this section we consider projection methods on *Krylov subspaces*, i.e., subspaces of the form

$$\mathcal{K}_m(A, v) \equiv \text{span}\,\{v, Av, A^2v, \ldots, A^{m-1}v\} \tag{6.2}$$

which will be denoted simply by $\mathcal{K}_m$ if there is no ambiguity. The dimension of the subspace of approximants increases by one at each step of the approximation process. A few elementary properties of Krylov subspaces can be established, many of which need no proof. A first property is that $\mathcal{K}_m$ is the subspace of all vectors in $\mathbb{R}^n$ which can be written as $x = p(A)v$, where p is a polynomial of degree not exceeding $m-1$. Recall that the minimal polynomial of a vector v is the nonzero monic polynomial p of lowest degree such that $p(A)v = 0$. The degree of the minimal polynomial of v with respect to A is often called the *grade of v with respect to A*, or simply the grade of v if there is no ambiguity. A consequence of the Cayley-Hamilton theorem is that the grade of v does not exceed n. The following proposition is easy to prove.

PROPOSITION 6.1 *Let μ be the grade of v. Then $\mathcal{K}_\mu$ is invariant under A and $\mathcal{K}_m = \mathcal{K}_\mu$ for all $m \geq \mu$.*

It was mentioned above that the dimension of $\mathcal{K}_m$ is nondecreasing. In fact, the following proposition determines the dimension of $\mathcal{K}_m$ in general.

PROPOSITION 6.2 *The Krylov subspace $\mathcal{K}_m$ is of dimension m if and only if the grade μ of v with respect to A is not less than m, i.e.,*

$$\dim(\mathcal{K}_m) = m \quad \leftrightarrow \quad \text{grade}(v) \geq m.$$

Therefore,

$$\dim(\mathcal{K}_m) = \min\,\{m, \text{grade}(v)\}.$$

Proof. The vectors $v, Av, \ldots, A^{m-1}v$ form a basis of $\mathcal{K}_m$ if and only if for any set of m scalars $\alpha_i, i = 0, \ldots, m-1$, where at least one α_i is nonzero, the linear combination $\sum_{i=0}^{m-1} \alpha_i A^i v$ is nonzero. This is equivalent to the condition that the only polynomial of degree $\leq m-1$ for which $p(A)v = 0$ is the zero polynomial. The second part of the proposition is a consequence of the previous proposition. ■

PROPOSITION 6.3 *Let Q_m be any projector onto $\mathcal{K}_m$ and let A_m be the section of A to $\mathcal{K}_m$, that is, $A_m = Q_m A_{|\mathcal{K}_m}$. Then for any polynomial q of degree not exceeding $m-1$,*

$$q(A)v = q(A_m)v,$$

and for any polynomial of degree $\leq m$,

$$Q_m q(A)v = q(A_m)v.$$

Proof. First we prove that $q(A)v = q(A_m)v$ for any polynomial q of degree $\leq m-1$. It is sufficient to show the property for the monic polynomials $q_i(t) \equiv t^i,\ i = 0, \ldots, m-1$. The proof is by induction. The property is true for the polynomial $q_0(t) \equiv 1$. Assume that it is true for $q_i(t) \equiv t^i$:

$$q_i(A)v = q_i(A_m)v.$$

Multiplying the above equation by A on both sides yields

$$q_{i+1}(A)v = Aq_i(A_m)v.$$

If $i+1 \leq m-1$ the vector on the left-hand side belongs to $\mathcal{K}_m$, and therefore if the above equation is multiplied on both sides by Q_m, then

$$q_{i+1}(A)v = Q_m A q_i(A_m)v.$$

Looking at the right-hand side we observe that $q_i(A_m)v$ belongs to $\mathcal{K}_m$. Hence,

$$q_{i+1}(A)v = Q_m A_{|\mathcal{K}_m} q_i(A_m)v = q_{i+1}(A_m)v,$$

which proves that the property is true for $i+1$, provided $i+1 \leq m-1$. For the case $i+1 = m$, it only remains to show that $Q_m q_m(A)v = q_m(A_m)v$, which follows from

$q_{m-1}(A)v = q_{m-1}(A_m)v$ by simply multiplying both sides by $Q_m A$. ■

6.3 ARNOLDI'S METHOD

Arnoldi's method [9] is an orthogonal projection method onto $\mathcal{K}_m$ for general non-Hermitian matrices. The procedure was introduced in 1951 as a means of reducing a dense matrix into Hessenberg form. Arnoldi presented his method in this manner but hinted that the eigenvalues of the Hessenberg matrix obtained from a number of steps smaller than n could provide accurate approximations to some eigenvalues of the original matrix. It was later discovered that this strategy leads to an efficient technique for approximating eigenvalues of large sparse matrices. The method will first be described theoretically, i.e., assuming exact arithmetic, then implementation details will be addressed.

6.3.1 THE BASIC ALGORITHM

Arnoldi's procedure is an algorithm for building an orthogonal basis of the Krylov subspace $\mathcal{K}_m$. In exact arithmetic, one variant of the algorithm is as follows:

ALGORITHM 6.1: Arnoldi

1. *Choose a vector* v_1 *of norm 1*
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3.   *Compute* $h_{ij} = (Av_j, v_i)$ *for* $i = 1, 2, \ldots, j$
4.   *Compute* $w_j := Av_j - \sum_{i=1}^{j} h_{ij} v_i$
5. $h_{j+1,j} = \|w_j\|_2$
6.   *If* $h_{j+1,j} = 0$ *then Stop*
7. $v_{j+1} = w_j / h_{j+1,j}$
8. *EndDo*

At each step, the algorithm multiplies the previous Arnoldi vector v_j by A and then orthonormalizes the resulting vector w_j against all previous v_i's by a standard Gram-Schmidt procedure. It will stop if the vector w_j computed in line 4 vanishes. This case will be examined shortly. Now a few simple properties of the algorithm are proved.

PROPOSITION 6.4 *Assume that Algorithm 6.1 does not stop before the* m*-th step. Then the vectors* $v_1, v_2, \ldots, v_m$ *form an orthonormal basis of the Krylov subspace*

$$\mathcal{K}_m = \operatorname{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}.$$

Proof. The vectors $v_j, j = 1, 2, \ldots, m$, are orthonormal by construction. That they span $\mathcal{K}_m$ follows from the fact that each vector v_j is of the form $q_{j-1}(A)v_1$ where q_{j-1} is a polynomial of degree $j-1$. This can be shown by induction on j as follows. The result is clearly true for $j = 1$, since $v_1 = q_0(A)v_1$ with $q_0(t) \equiv 1$. Assume that the result is true for all integers $\leq j$ and consider v_{j+1}. We have

$$h_{j+1}v_{j+1} = Av_j - \sum_{i=1}^{j} h_{ij}v_i = Aq_{j-1}(A)v_1 - \sum_{i=1}^{j} h_{ij}q_{i-1}(A)v_1 \tag{6.3}$$

which shows that v_{j+1} can be expressed as $q_j(A)v_1$ where q_j is of degree j and completes the proof. ∎

PROPOSITION 6.5 *Denote by V_m, the $n \times m$ matrix with column vectors $v_1, \ldots, v_m$, by $\bar{H}_m$, the $(m+1) \times m$ Hessenberg matrix whose nonzero entries h_{ij} are defined by Algorithm 6.1, and by H_m the matrix obtained from $\bar{H}_m$ by deleting its last row. Then the following relations hold:*

$$AV_m = V_mH_m + w_me_m^T \tag{6.4}$$

$$= V_{m+1}\bar{H}_m, \tag{6.5}$$

$$V_m^TAV_m = H_m. \tag{6.6}$$

Proof. The relation (6.5) follows from the following equality which is readily derived from lines 4, 5, and 7 of Algorithm 6.1,

$$Av_j = \sum_{i=1}^{j+1} h_{ij}v_i, \quad j = 1, 2, \ldots, m. \tag{6.7}$$

Relation (6.4) is a matrix reformulation of (6.7). Relation (6.6) follows by multiplying both sides of (6.4) by V_m^T and making use of the orthonormality of $\{v_1, \ldots, v_m\}$. ∎

The result of the proposition is illustrated in Figure 6.1.

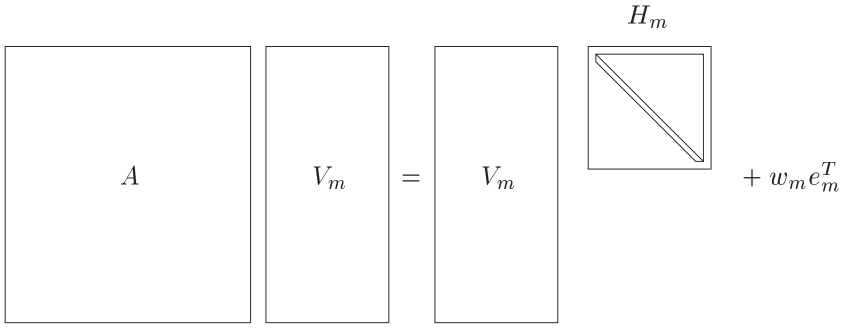


Figure 6.1 *The action of A on V_m gives V_mH_m plus a rank-one matrix.*

As was noted earlier, the algorithm may break down in case the norm of w_j vanishes at a certain step j. In this case, the vector v_{j+1} cannot be computed and the algorithm stops.

Still to be determined are the conditions under which this situation occurs.

PROPOSITION 6.6 *Arnoldi's algorithm breaks down at step j (i.e., $h_{j+1,j} = 0$ in line 5 of Algorithm 6.1), if and only if the minimal polynomial of v_1 is of degree j. Moreover, in this case the subspace $\mathcal{K}_j$ is invariant under A.*

Proof. If the degree of the minimal polynomial is j, then w_j must be equal to zero. Indeed, otherwise v_{j+1} can be defined and as a result $\mathcal{K}_{j+1}$ would be of dimension $j + 1$. Then Proposition 6.2 would imply that $\mu \geq j + 1$, which is a contradiction. To prove the converse, assume that $w_j = 0$. Then the degree μ of the minimal polynomial of v_1 is such that $\mu \leq j$. Moreover, it is impossible that $\mu < j$. Otherwise, by the first part of this proof, the vector w_μ would be zero and the algorithm would have stopped at the earlier step number μ. The rest of the result follows from Proposition 6.1. ■

A corollary of the proposition is that a projection method onto the subspace $\mathcal{K}_j$ will be exact when a breakdown occurs at step j. This result follows from Proposition 5.6 seen in Chapter 5. It is for this reason that such breakdowns are often called *lucky breakdowns*.

6.3.2 PRACTICAL IMPLEMENTATIONS

In the previous description of the Arnoldi process, exact arithmetic was assumed, mainly for simplicity. In practice, much can be gained by using the Modified Gram-Schmidt or the Householder algorithm instead of the standard Gram-Schmidt algorithm. With the Modified Gram-Schmidt alternative the algorithm takes the following form:

ALGORITHM 6.2: Arnoldi-Modified Gram-Schmidt

1. *Choose a vector* v_1 *of norm 1*
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3.   *Compute* $w_j := Av_j$
4.   *For* $i = 1, \ldots, j$ *Do:*
5. $h_{ij} = (w_j, v_i)$
6. $w_j := w_j - h_{ij} v_i$
7.   *EndDo*
8. $h_{j+1,j} = \|w_j\|_2$. *If* $h_{j+1,j} = 0$ *Stop*
9. $v_{j+1} = w_j / h_{j+1,j}$
10. *EndDo*

In exact arithmetic, this algorithm and Algorithm 6.1 are mathematically equivalent. In the presence of round-off the above formulation is much more reliable. However, there are cases where cancellations are so severe in the orthogonalization steps that even the Modified Gram-Schmidt option is inadequate. In this case, two further improvements can be utilized.

The first improvement resorts to double orthogonalization. Whenever the final vector w_j obtained at the end of the main loop in the above algorithm has been computed, a test is performed to compare its norm with the norm of the initial w_j (which is $\|Av_j\|_2$). If the reduction falls below a certain threshold, indicating severe cancellation might have occurred, a second orthogonalization is made. It is known from a result by Kahan that additional orthogonalizations are superfluous (see, for example, Parlett [160]).

The second improvement is to use a different technique altogether. From the numerical point of view, one of the most reliable orthogonalization techniques is the Householder algorithm. Recall from Chapter 1 that the Householder orthogonalization uses reflection matrices of the form $P_k = I - 2w_k w_k^T$ to transform a matrix X into upper triangular form. In the Arnoldi algorithm, the column vectors of the matrix X to be orthonormalized are not available ahead of time. Instead, the next vector is obtained as Av_j, where v_j is the current basis vector. In the Householder algorithm an orthogonal column v_i is obtained as $P_1 P_2 \dots P_i e_i$ where $P_1, \dots, P_i$ are the previous Householder matrices. This vector is then multiplied by A and the previous Householder transforms are applied to it. Then, the next Householder transform is determined from the resulting vector. This procedure is described in the following algorithm, which was originally proposed by Walker [221].

ALGORITHM 6.3: Householder Arnoldi

1. *Select a nonzero vector* v; *Set* $z_1 = v$
2. *For* $j = 1, \dots, m, m+1$ *Do:*
3. *Compute the Householder unit vector* w_j *such that*
4. $(w_j)_i = 0, i = 1, \dots, j-1$ *and*
5. $(P_j z_j)_i = 0, i = j+1, \dots, n$, *where* $P_j = I - 2w_j w_j^T$
6. $h_{j-1} = P_j z_j$
7. $v_j = P_1 P_2 \dots P_j e_j$
8. *If* $j \le m$ *compute* $z_{j+1} := P_j P_{j-1} \dots P_1 A v_j$
9. *EndDo*

For details regarding the determination of the Householder vector w_j in the third to fifth lines and on its use in the sixth to eight lines, see Chapter 1. Recall that the matrices P_j need not be formed explicitly. To obtain h_{j-1} from z_j in line 6, zero out all the components from position $j+1$ through n of the n-vector z_j and change its j-th component, leaving all others unchanged. Thus, the $n \times m$ matrix $[h_0, h_1, \dots, h_m]$ will have the same structure as the matrix X_m of equation (1.22) in Chapter 1. By comparison with the Householder algorithm seen in Chapter 1, we can infer that the above process computes the QR factorization of the matrix $v, Av_1, Av_2, Av_3, \dots, Av_m$. Define

$$Q_j = P_j P_{j-1} \dots P_1. \tag{6.8}$$

The definition of z_{j+1} in line 8 of the algorithm yields the relation,

$$Q_j A v_j = z_{j+1}.$$

After the next Householder transformation P_{j+1} is applied in line 6, h_j satisfies the relation,

$$h_j = P_{j+1} z_{j+1} = P_{j+1} Q_j A v_j = Q_{j+1} A v_j. \tag{6.9}$$

Now observe that since the components $j+2,\ldots,n$ of h_j are zero, then $P_i h_j = h_j$ for any $i \geq j+2$. Hence,

$$h_j = P_m P_{m-1} \ldots P_{j+2} h_j = Q_m A v_j, \quad j = 1, \ldots, m.$$

This leads to the factorization,

$$Q_m[v, Av_1, Av_2, \ldots, Av_m] = [h_0, h_1, \ldots, h_m] \tag{6.10}$$

where the matrix $[h_0, \ldots, h_m]$ is $n \times (m+1)$ and is upper triangular and Q_m is unitary.

It is important to relate the vectors v_i and h_i defined in this algorithm with vectors of the standard Arnoldi process. Let $\bar{H}_m$ be the $(m+1) \times m$ matrix obtained from the first $m+1$ rows of the $n \times m$ matrix $[h_1, \ldots, h_m]$. Since Q_{j+1} is unitary we have $Q_{j+1}^{-1} = Q_{j+1}^T$ and hence, from the relation (6.9)

$$Av_j = Q_{j+1}^T \sum_{i=1}^{j+1} h_{ij} e_i = \sum_{i=1}^{j+1} h_{ij} Q_{j+1}^T e_i$$

where each e_i is the i-th column of the $n \times n$ identity matrix. Since $P_k e_i = e_i$ for $i < k$, it is not difficult to see that

$$Q_{j+1}^T e_i = P_1 \ldots P_{j+1} e_i = v_i, \text{ for } i \leq j+1. \tag{6.11}$$

This yields the relation $Av_j = \sum_{i=1}^{j+1} h_{ij} v_i$, for $j = 1, \ldots, m$, which can be written in matrix form as

$$AV_m = V_{m+1} \bar{H}_m.$$

This is identical with the relation (6.5) obtained with the Gram-Schmidt or Modified Gram-Schmidt implementation. The v_i's form an orthonormal basis of the Krylov subspace $\mathcal{K}_m$ and are identical with the v_i's defined by the Arnoldi process, apart from a possible sign difference.

Although the Householder algorithm is numerically more viable than the Gram-Schmidt or Modified Gram-Schmidt versions, it is also more expensive. The cost of each of the outer loops, corresponding to the j control variable, is dominated by lines 7 and 8. These apply the reflection matrices P_i for $i = 1, \ldots, j$ to a vector, perform the matrix-vector product Av_j, and then apply the matrices P_i for $i = j, j-1, \ldots, 1$ to a vector. The application of each P_i to a vector is performed as

$$(I - 2w_i w_i^T)v = v - \sigma w_i \quad \text{with} \quad \sigma = 2w_i^T v.$$

This is essentially the result of a dot-product of length $n - i + 1$ followed by a vector update of the same length, requiring a total of about $4(n-i+1)$ operations for each application of P_i. Neglecting the last step, the number of operations due to the Householder transformations alone approximately totals

$$\sum_{j=1}^{m} \sum_{i=1}^{j} 8(n-i+1) = 8 \sum_{j=1}^{m} \left(jn - \frac{j(j-1)}{2} \right) \approx 4m^2 n - \frac{4}{3} m^3.$$

The table below shows the costs of different orthogonalization procedures. GS stands for Gram-Schmidt, MGS for Modified Gram-Schmidt, MGSR for Modified Gram-Schmidt with reorthogonalization, and HO for Householder.

	GS	MGS	MGSR	HO
Flops	$2m^2n$	$2m^2n$	$4m^2n$	$4m^2n - \frac{4}{3}m^3$
Storage	$(m+1)n$	$(m+1)n$	$(m+1)n$	$(m+1)n - \frac{1}{2}m^2$

The number of operations shown for MGSR corresponds to the worst case scenario when a second orthogonalization is performed each time. In practice, the number of operations is usually closer to that of the standard MGS. Regarding storage, the vectors $v_i, i = 1, \ldots, m$ need not be saved. In the algorithms for solving linear systems, these vectors are needed at the end of the process. This issue will be covered with the Householder implementations of these algorithms. For now, assume that only the w_i's are saved. The small gain in memory usage in the Householder version can be explained by the diminishing lengths of the vectors required at each step of the Householder transformation. However, this difference is negligible relative to the whole storage requirement of the algorithm, because $m \ll n$, typically.

The Householder orthogonalization may be a reasonable choice when developing general purpose, reliable software packages where robustness is a critical criterion. This is especially true for solving eigenvalue problems since the cost of orthogonalization is then amortized over several eigenvalue/eigenvector calculations. When solving linear systems, the Modified Gram-Schmidt orthogonalization, with a reorthogonalization strategy based on a measure of the level of cancellation, is more than adequate in most cases.

6.4 ARNOLDI'S METHOD FOR LINEAR SYSTEMS (FOM)

Given an initial guess x_0 to the original linear system $Ax = b$, we now consider an orthogonal *projection method* as defined in the previous chapter, which takes $\mathcal{L} = \mathcal{K} = \mathcal{K}_m(A, r_0)$, with

$$\mathcal{K}_m(A, r_0) = span\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}, \tag{6.12}$$

in which $r_0 = b - Ax_0$. This method seeks an approximate solution x_m from the affine subspace $x_0 + \mathcal{K}_m$ of dimension m by imposing the Galerkin condition

$$b - Ax_m \perp \mathcal{K}_m. \tag{6.13}$$

If $v_1 = r_0/\|r_0\|_2$ in Arnoldi's method, and set $\beta = \|r_0\|_2$, then

$$V_m^T A V_m = H_m$$

by (6.6) and

$$V_m^T r_0 = V_m^T(\beta v_1) = \beta e_1.$$

As a result, the approximate solution using the above m-dimensional subspaces is given by

$$x_m = x_0 + V_m y_m, \tag{6.14}$$

$$y_m = H_m^{-1}(\beta e_1). \tag{6.15}$$

A method based on this approach and called the Full Orthogonalization Method (FOM) is described next. Modified Gram-Schmidt is used in the Arnoldi step.

ALGORITHM 6.4: Full Orthogonalization Method (FOM)

1. *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2. *Define the* $m \times m$ *matrix* $H_m = \{h_{ij}\}_{i,j=1,\ldots,m}$; *Set* $H_m = 0$
3. *For* $j = 1, 2, \ldots, m$ *Do:*
4. *Compute* $w_j := Av_j$
5. *For* $i = 1, \ldots, j$ *Do:*
6. $h_{ij} = (w_j, v_i)$
7. $w_j := w_j - h_{ij}v_i$
8. *EndDo*
9. *Compute* $h_{j+1,j} = \|w_j\|_2$. *If* $h_{j+1,j} = 0$ *set* $m := j$ *and Goto 12*
10. *Compute* $v_{j+1} = w_j/h_{j+1,j}$.
11. *EndDo*
12. *Compute* $y_m = H_m^{-1}(\beta e_1)$ *and* $x_m = x_0 + V_m y_m$

The above algorithm depends on a parameter m which is the dimension of the Krylov subspace. In practice it is desirable to select m in a dynamic fashion. This would be possible if the residual norm of the solution x_m is available inexpensively (without having to compute x_m itself). Then the algorithm can be stopped at the appropriate step using this information. The following proposition gives a result in this direction.

PROPOSITION 6.7 *The residual vector of the approximate solution* x_m *computed by the FOM Algorithm is such that*

$$b - Ax_m = -h_{m+1,m} e_m^T y_m v_{m+1}$$

and, therefore,

$$\|b - Ax_m\|_2 = h_{m+1,m} |e_m^T y_m|. \tag{6.16}$$

Proof. We have the relations,

$$\begin{aligned} b - Ax_m &= b - A(x_0 + V_m y_m) \\ &= r_0 - AV_m y_m \\ &= \beta v_1 - V_m H_m y_m - h_{m+1,m} e_m^T y_m v_{m+1}. \end{aligned}$$

By the definition of y_m, $H_m y_m = \beta e_1$, and so $\beta v_1 - V_m H_m y_m = 0$ from which the result follows immediately. ∎

A rough estimate of the cost of each step of the algorithm is determined as follows. If $Nz(A)$ is the number of nonzero elements of A, then m steps of the Arnoldi procedure will require m matrix-vector products at the cost of $2m \times Nz(A)$. Each of the Gram-Schmidt steps costs approximately $4 \times j \times n$ operations, which brings the total over the m steps to approximately $2m^2n$. Thus, on the average, a step of FOM costs approximately

$$2Nz(A) + 2mn.$$

Regarding storage, m vectors of length n are required to save the basis V_m. Additional vectors must be used to keep the current solution and right-hand side, and a scratch vector for the matrix-vector product. In addition, the Hessenberg matrix H_m must be saved. The total is therefore roughly

$$(m+3)n + \frac{m^2}{2}.$$

In most situations m is small relative to n, so this cost is dominated by the first term.

6.4.1 VARIATION 1: RESTARTED FOM

Consider now the algorithm from a practical viewpoint. As m increases, the computational cost increases at least as $O(m^2)n$ because of the Gram-Schmidt orthogonalization. The memory cost increases as $O(mn)$. For large n this limits the largest value of m that can be used. There are two remedies. The first is to restart the algorithm periodically and the second is to "truncate" the orthogonalization in the Arnoldi algorithm. In this section we consider the first of these two options, which is described below.

ALGORITHM 6.5: Restarted FOM (FOM(m))

1. *Compute* $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, *and* $v_1 = r_0/\beta$.
2. *Generate the Arnoldi basis and the matrix* H_m *using the Arnoldi algorithm*
3. *starting with* v_1.
4. *Compute* $y_m = H_m^{-1}\beta e_1$ *and* $x_m = x_0 + V_m y_m$. *If satisfied then Stop.*
5. *Set* $x_0 := x_m$ *and go to 1.*

There are many possible variations to this basic scheme. One that is generally more economical in practice is based on the observation that sometimes a small m is sufficient for convergence and sometimes the largest possible m is necessary. Hence, the idea of averaging over different values of m. Start the algorithm with $m = 1$ and increment m by one in line 5 until a certain m_{max} is reached, after which m is reset to one, or kept the same. These variations will not be considered here.

Example 6.1 Table 6.1 shows the results of applying the FOM algorithm with no preconditioning to three of the test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	109	4442	0.36E-03	0.67E-04
F3D	66	11664	0.87E-03	0.35E-03
ORS	300	13558	0.26E+00	0.71E-04

Table 6.1 *A test run of FOM with no preconditioning.*

The column labeled *Iters* shows the total actual number of matrix-vector multiplications (matvecs) required to converge. The stopping criterion used is that the 2-norm of the residual be reduced by a factor of 10^7 relative to the 2-norm of the initial residual. A maximum of 300 matvecs are allowed. *Kflops* is the total number of floating point operations performed, in thousands. *Residual* and *Error* represent the two-norm of the residual and error vectors, respectively. In this test, m was taken to be 10. Note that the method did not succeed in solving the third problem.

6.4.2 VARIATION 2: IOM AND DIOM

A second alternative to FOM is to truncate the Arnoldi recurrence. Specifically, an integer k is selected and the following "incomplete" orthogonalization is performed.

ALGORITHM 6.6: Incomplete Orthogonalization Process

1. *For* $j = 1, 2, \ldots, m$ *Do:*
2.   *Compute* $w := Av_j$
3.   *For* $i = \max\{1, j-k+1\}, \ldots, j$ *Do:*
4. $h_{i,j} = (w, v_i)$
5. $w := w - h_{ij} v_i$
6.   *EndDo*
7.   *Compute* $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w/h_{j+1,j}$
8. *EndDo*

The number of directions k against which to orthogonalize may be dictated by memory limitations. The Incomplete Orthogonalization Method (IOM) consists of performing the above incomplete orthogonalization procedure and computing an approximate solution using the same formulas (6.14) and (6.15).

ALGORITHM 6.7: IOM Algorithm

Run a modification of Algorithm 6.4 in which the Arnoldi process in lines 3 to 11 is replaced by the Incomplete Orthogonalization process and every other computation remains unchanged.

It is now necessary to keep only the k previous v_i vectors. The others are not needed in the above process and may be discarded. However, the difficulty remains that when the solution is computed by formula (6.14), all the vectors v_i for $i = 1, 2, \ldots, m$ are required. One option is to recompute them at the end, but essentially this doubles the cost of the algorithm. Fortunately, a formula can be developed whereby the current approximate solution x_m can be updated from the previous approximation x_{m-1} and a small number of vectors that are also updated at each step. This *progressive* formulation of the solution leads to an algorithm termed *Direct IOM* (DIOM) which we now derive.

The Hessenberg matrix H_m obtained from the incomplete orthogonalization process has a band structure with a bandwidth of $k+1$. For example, when $k=3$ and $m=5$, it is of the form

$$H_m = \begin{pmatrix} h_{11} & h_{12} & h_{13} & & \\ h_{21} & h_{22} & h_{23} & h_{24} & \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \end{pmatrix}. \tag{6.17}$$

The *Direct* version of IOM is derived from exploiting the special structure of the LU factorization, $H_m = L_m U_m$, of the matrix H_m. Assuming no pivoting is used, the matrix L_m is unit lower bidiagonal and U_m is banded upper triangular, with k diagonals. Thus, the above matrix has a factorization of the form

$$H_m = \begin{pmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ & l_{32} & 1 & & \\ & & l_{43} & 1 & \\ & & & l_{54} & 1 \end{pmatrix} \times \begin{pmatrix} u_{11} & u_{12} & u_{13} & & \\ & u_{22} & u_{23} & u_{24} & \\ & & u_{33} & u_{34} & u_{35} \\ & & & u_{44} & u_{45} \\ & & & & u_{55} \end{pmatrix}.$$

The approximate solution is then given by

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1).$$

Defining

$$P_m \equiv V_m U_m^{-1}$$

and

$$z_m = L_m^{-1}(\beta e_1),$$

the approximate solution is given by

$$x_m = x_0 + P_m z_m. \tag{6.18}$$

Because of the structure of U_m, P_m can be updated easily. Indeed, equating the last columns of the matrix relation $P_m U_m = V_m$ yields

$$\sum_{i=m-k+1}^{m} u_{im} p_i = v_m,$$

which allows the vector p_m to be computed from the previous p_i's and v_m, with the help of the relation,

$$p_m = \frac{1}{u_{mm}} \left[v_m - \sum_{i=m-k+1}^{m-1} u_{im} p_i \right].$$

In addition, because of the structure of L_m, we have the relation

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix}$$

in which

$$\zeta_m = -l_{m,m-1}\zeta_{m-1}.$$

From (6.18),

$$x_m = x_0 + [P_{m-1}, p_m]\begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} = x_0 + P_{m-1}z_{m-1} + \zeta_m p_m.$$

Noting that $x_0 + P_{m-1}z_{m-1} = x_{m-1}$, it follows that the approximation x_m can be updated at each step by the relation,

$$x_m = x_{m-1} + \zeta_m p_m$$

where p_m is defined above. This gives the following algorithm, called the Direct Incomplete Orthogonalization Method (DIOM).

ALGORITHM 6.8: DIOM

1. *Choose* x_0 *and compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, $v_1 := r_0/\beta$.
2. *For* $m = 1, 2, \ldots$, *until convergence Do:*
3. *Compute* h_{im}, $i = \max\{1, m-k+1\}, \ldots, m$ *and* v_{m+1} *as in*
4. *lines 2-7 of Algorithm (6.6).*
5. *Update the LU factorization of* H_m, *i.e., obtain the last column*
6. *of* U_m *using the previous* k *pivots. If* $u_{mm} = 0$ *Stop.*
7. $\zeta_m = \{$ *if* $m = 1$ *then* β, *else* $-l_{m,m-1}\,\zeta_{m-1}\}$
8. $p_m = u_{mm}^{-1}\left(v_m - \sum_{i=m-k+1}^{m-1} u_{im}p_i\right)$ *(for* $i \leq 0$ *set* $u_{im}p_i \equiv 0$*)*
9. $x_m = x_{m-1} + \zeta_m p_m$
10. *EndDo*

Note that the above algorithm is based implicitly on Gaussian elimination without pivoting for the solution of the Hessenberg system $H_m y_m = \beta e_1$. This may cause a premature termination in line 6. Fortunately, there is an implementation based on Gaussian elimination with partial pivoting. The details of this variant can be found in [174]. DIOM can also be derived by imposing the properties that are satisfied by the residual vector and the conjugate directions, i.e., the p_i's.

Observe that (6.4) is still valid and as a consequence, Proposition 6.7, which is based on it, still holds. That is because the orthogonality properties were not used to derive the two relations therein. Since the residual vector is a scalar multiple of v_{m+1} and since the v_i's are no longer orthogonal, IOM and DIOM are not orthogonal projection techniques. They can, however, be viewed as oblique projection techniques onto $\mathcal{K}_m$ and orthogonal to an artificially constructed subspace.

PROPOSITION 6.8 *IOM and DIOM are mathematically equivalent to a projection process onto* $\mathcal{K}_m$ *and orthogonally to*

$$\mathcal{L}_m = \operatorname{span}\{z_1, z_2, \ldots, z_m\}$$

where

$$z_i = v_i - (v_i, v_{m+1})v_{m+1}, \quad i = 1, \ldots, m.$$

Proof. The proof is an immediate consequence of the fact that r_m is a multiple of v_{m+1} and by construction, v_{m+1} is orthogonal to all z_i's defined in the proposition. ■

The following simple properties can be shown:

- The residual vectors r_i, $i = 1, \ldots, m$, are "locally" orthogonal,
$$(r_j, r_i) = 0, \quad \text{for} \quad |i - j| \leq k, \;\; i \neq j.$$
- The p_j's are locally A-orthogonal to the Arnoldi vectors, i.e.,
$$(Ap_j, v_i) = 0 \quad \text{for} \quad j - k + 1 < i < j.$$
- For the case $k = \infty$ (full orthogonalization) the p_j's are semi-conjugate, i.e.,
$$(Ap_j, p_i) = 0 \quad \text{for} \quad i < j.$$

6.5 GMRES

The Generalized Minimum Residual Method (GMRES) is a projection method based on taking $\mathcal{K} = \mathcal{K}_m$ and $\mathcal{L} = A\mathcal{K}_m$, in which $\mathcal{K}_m$ is the m-th Krylov subspace with $v_1 = r_0/\|r_0\|_2$. As seen in Chapter 5, such a technique minimizes the residual norm over all vectors in $x_0 + \mathcal{K}_m$. The implementation of an algorithm based on this approach is similar to that of the FOM algorithm. We first describe the basic idea and then discuss a few practical variations.

6.5.1 THE BASIC GMRES ALGORITHM

There are two ways to derive the algorithm. The first way exploits the optimality property and the relation (6.5). Any vector x in $x_0 + \mathcal{K}_m$ can be written as

$$x = x_0 + V_m y, \tag{6.19}$$

where y is an m-vector. Defining

$$J(y) = \|b - Ax\|_2 = \|b - A\left(x_0 + V_m y\right)\|_2, \tag{6.20}$$

the relation (6.5) results in

$$\begin{aligned} b - Ax &= b - A\left(x_0 + V_m y\right) \\ &= r_0 - AV_m y \\ &= \beta v_1 - V_{m+1}\bar{H}_m y \\ &= V_{m+1}\left(\beta e_1 - \bar{H}_m y\right). \end{aligned} \tag{6.21}$$

Since the column-vectors of V_{m+1} are orthonormal, then

$$J(y) \equiv \|b - A\left(x_0 + V_m y\right)\|_2 = \|\beta e_1 - \bar{H}_m y\|_2. \tag{6.22}$$

The GMRES approximation is the unique vector of $x_0 + \mathcal{K}_m$ which minimizes (6.20). By (6.19) and (6.22), this approximation can be obtained quite simply as $x_m = x_0 + V_m y_m$ where y_m minimizes the function $J(y) = \|\beta e_1 - \bar{H}_m y\|_2$, i.e.,

$$x_m = x_0 + V_m y_m, \quad \text{where} \tag{6.23}$$

$$y_m = \operatorname{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2. \tag{6.24}$$

The minimizer y_m is inexpensive to compute since it requires the solution of an $(m+1) \times m$ least-squares problem where m is typically small. This gives the following algorithm.

ALGORITHM 6.9: GMRES

1. *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2. *Define the* $(m+1) \times m$ *matrix* $\bar{H}_m = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$. *Set* $\bar{H}_m = 0$.
3. *For* $j = 1, 2, \ldots, m$ *Do:*
4.   *Compute* $w_j := Av_j$
5.   *For* $i = 1, \ldots, j$ *Do:*
6. $h_{ij} := (w_j, v_i)$
7. $w_j := w_j - h_{ij} v_i$
8.   *EndDo*
9. $h_{j+1,j} = \|w_j\|_2$. *If* $h_{j+1,j} = 0$ *set* $m := j$ *and go to 12*
10. $v_{j+1} = w_j / h_{j+1,j}$
11. *EndDo*
12. *Compute* y_m *the minimizer of* $\|\beta e_1 - \bar{H}_m y\|_2$ *and* $x_m = x_0 + V_m y_m$.

The second way to derive the GMRES algorithm is to use the equations (5.7) with $W_m = AV_m$. This is the subject of Exercise 4.

6.5.2 THE HOUSEHOLDER VERSION

The previous algorithm utilizes the Modified Gram-Schmidt orthogonalization in the Arnoldi process. Section 6.3.2 described a Householder variant of the Arnoldi process which is numerically more robust than Gram-Schmidt. Here, we focus on a modification of GMRES which retrofits the Householder orthogonalization. Section 6.3.2 explained how to get the v_j and the columns of $\bar{H}_{m+1}$ at each step, from the Householder-Arnoldi algorithm. Since V_m and $\bar{H}_m$ are the only items needed to extract the approximate solution at the end of the GMRES process, the modification seems rather straightforward. However, this is only true if the v_i's are stored. In this case, line 12 would remain the same and the modification to the algorithm would be in lines 3-11 which are to be replaced by the Householder variant of the Arnoldi process. It was mentioned in Section 6.3.2 that it is preferable not to store the v_i's because this would double the storage requirement. In this case, a formula must be found to generate the approximate solution in line 12, using only the w_i's, i.e., the P_i's. Let

$$y_m = (\eta_1, \eta_2, \cdots, \eta_m)^T,$$

so that the solution is of the form $x_m = x_0 + \eta_1 v_1 + \cdots + \eta_m v_m$. Recall that in the Householder variant of the Arnoldi process, each v_j is defined by

$$v_j = P_1 P_2 \dots P_j e_j.$$

Using a Horner-like scheme, we obtain

$$\begin{aligned} x_m &= x_0 + \eta_1 P_1 e_1 + \eta_2 P_1 P_2 e_2 + \dots + \eta_m P_1 P_2 \dots P_m e_m \\ &= x_0 + P_1 \left(\eta_1 e_1 + P_2 \left(\eta_2 e_2 + \dots + P_{m-1} \left(\eta_{m-1} e_{m-1} + P_m \eta_m e_m\right)\right)\right). \end{aligned}$$

Therefore, when Householder orthogonalization is used, then line 12 of the GMRES algorithm should be replaced by a step of the form

$$z := 0 \tag{6.25}$$

$$z := P_j \left(\eta_j e_j + z\right), j = m, m-1, \dots, 1 \tag{6.26}$$

$$x_m = x_0 + z. \tag{6.27}$$

The above step requires roughly as many operations as computing the last Arnoldi vector v_m. Therefore, its cost is negligible relative to the cost of the Arnoldi loop.

ALGORITHM 6.10: GMRES with Householder orthogonalization

1. *Compute* $r_0 = b - Ax_0$, $z := r_0$.
2. *For* $j = 1, \dots, m, m+1$ *Do:*
3. *Compute the Householder unit vector* w_j *such that*
4. $(w_j)_i = 0, i = 1, \dots, j-1$ *and*
5. $(P_j z)_i = 0, i = j+1, \dots, n$ *where* $P_j = I - 2 w_j w_j^T$;
6. $h_{j-1} := P_j z$; *If* $j = 1$ *then let* $\beta := e_1^T h_0$.
7. $v := P_1 P_2 \dots P_j e_j$.
8. *If* $j \le m$ *compute* $z := P_j P_{j-1} \dots P_1 A v$,
9. *EndDo*
10. *Define* $\bar{H}_m$ = *the* $(m+1) \times m$ *upper part of the matrix* $[h_1, \dots, h_m]$.
11. *Compute* $y_m = \text{Argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$. *Let* $y_m = (\eta_1, \eta_2, \dots, \eta_m)^T$.
12. *z := 0*
13. *For* $j = m, m-1, \dots, 1$ *Do:*
14. $z := P_j \left(\eta_j e_j + z\right)$,
15. *EndDo*
16. *Compute* $x_m = x_0 + z$

Note that now only the set of w_j vectors needs to be saved. The scalar β defined in line 6 is equal to $\pm\|r_0\|_2$. This is because $P_1 z = \beta e_1$ where β is defined by the equations (1.21) seen in Chapter 1, which define the first Householder transformation. As was observed earlier the Householder factorization actually obtains the QR factorization (6.10) with $v = r_0$. We can also formulate GMRES directly from this factorization. Indeed, if $x = x_0 + V_m y_m$, then according to this factorization, the corresponding residual norm is equal to

$$\|h_0 - \eta_1 h_1 - \eta_2 h_2 - \dots - \eta_m h_m\|_2$$

whose minimizer is the same as the one defined by the algorithm.

The details of implementation of the solution of the least-squares problem as well as the estimate of the residual norm are identical with those of the Gram-Schmidt versions and are discussed next.

6.5.3 PRACTICAL IMPLEMENTATION ISSUES

A clear difficulty with Algorithm 6.9 is that it does not provide the approximate solution x_m explicitly at each step. As a result, it is not easy to determine when to stop. One remedy is to compute the approximation solution x_m at regular intervals and check for convergence by a test on the residual, for example. However, there is a more elegant solution which is related to the way in which the least-squares problem (6.24) is solved.

In order to solve the least-squares problem $\min \|\beta e_1 - \bar{H}_m y\|$, it is natural to transform the Hessenberg matrix into upper triangular form by using plane rotations. Define the rotation matrices

$$\Omega_i = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c_i & s_i & & & \\ & & & -s_i & c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} \begin{array}{l} \\ \\ \\ \leftarrow \text{row } i \\ \leftarrow \text{row } i+1 \\ \\ \\ \\ \end{array} \tag{6.28}$$

with $c_i^2 + s_i^2 = 1$. If m steps of the GMRES iteration are performed then these matrices have dimension $(m+1) \times (m+1)$.

Multiply the Hessenberg matrix $\bar{H}_m$ and the corresponding right-hand side $\bar{g}_0 \equiv \beta e_1$ by a sequence of such matrices from the left. The coefficients s_i, c_i are selected to eliminate $h_{i+1,i}$ at each time. Thus, if $m = 5$ we would have

$$\bar{H}_5 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix}, \quad \bar{g}_0 = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Then premultiply $\bar{H}_5$ by

$$\Omega_1 = \begin{pmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}$$

with

$$s_1 = \frac{h_{21}}{\sqrt{h_{11}^2 + h_{21}^2}}, \quad c_1 = \frac{h_{11}}{\sqrt{h_{11}^2 + h_{21}^2}}$$

to obtain the matrix and right-hand side

$$\bar{H}_5^{(1)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ & h_{22}^{(1)} & h_{23}^{(1)} & h_{24}^{(1)} & h_{25}^{(1)} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix}, \quad \bar{g}_1 = \begin{pmatrix} c_1\beta \\ -s_1\beta \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{6.29}$$

We can now premultiply the above matrix and right-hand side again by a rotation matrix Ω_2 to eliminate h_{32}. This is achieved by taking

$$s_2 = \frac{h_{32}}{\sqrt{(h_{22}^{(1)})^2 + h_{32}^2}}, \quad c_2 = \frac{h_{22}^{(1)}}{\sqrt{(h_{22}^{(1)})^2 + h_{32}^2}}.$$

This elimination process is continued until the m-th rotation is applied, which transforms the problem into one involving the matrix and right-hand side,

$$\bar{H}_5^{(5)} = \begin{pmatrix} h_{11}^{(5)} & h_{12}^{(5)} & h_{13}^{(5)} & h_{14}^{(5)} & h_{15}^{(5)} \\ & h_{22}^{(5)} & h_{23}^{(5)} & h_{24}^{(5)} & h_{25}^{(5)} \\ & & h_{33}^{(5)} & h_{34}^{(5)} & h_{35}^{(5)} \\ & & & h_{44}^{(5)} & h_{45}^{(5)} \\ & & & & h_{55}^{(5)} \\ & & & & 0 \end{pmatrix}, \quad \bar{g}_5 = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ . \\ . \\ \gamma_6 \end{pmatrix}. \tag{6.30}$$

Generally, the scalars c_i and s_i of the i^{th} rotation Ω_i are defined as

$$s_i = \frac{h_{i+1,i}}{\sqrt{(h_{ii}^{(i-1)})^2 + h_{i+1,i}^2}}, \quad c_i = \frac{h_{ii}^{(i-1)}}{\sqrt{(h_{ii}^{(i-1)})^2 + h_{i+1,i}^2}}. \tag{6.31}$$

Define Q_m the product of matrices Ω_i,

$$Q_m = \Omega_m \Omega_{m-1} \dots \Omega_1 \tag{6.32}$$

and

$$\bar{R}_m = \bar{H}_m^{(m)} = Q_m \bar{H}_m, \tag{6.33}$$

$$\bar{g}_m = Q_m(\beta e_1) = (\gamma_1, \dots, \gamma_{m+1})^T. \tag{6.34}$$

Since Q_m is unitary,

$$\min \|\beta e_1 - \bar{H}_m y\|_2 = \min \|\bar{g}_m - \bar{R}_m y\|_2.$$

The solution to the above least-squares problem is obtained by simply solving the triangular system resulting from deleting the last row of the matrix $\bar{R}_m$ and right-hand side $\bar{g}_m$ in (6.30). In addition, it is clear that for the solution y_*, the "residual" $\|\beta e_1 - \bar{H}_m y_*\|$ is nothing but the last element of the right-hand side, i.e., the term γ_6 in the above illustration.

PROPOSITION 6.9 *Let $\Omega_i, i = 1, \dots, m$ be the rotation matrices used to transform $\bar{H}_m$ into an upper triangular form and $\overline{R}_m, \bar{g}_m = (\gamma_1, \dots, \gamma_{m+1})^T$ the resulting matrix and right-hand side, as defined by (6.33), (6.34). Denote by R_m the $m \times m$ upper triangular* *matrix obtained from $\bar{R}_m$ by deleting its last row and by g_m the m-dimensional vector obtained from $\bar{g}_m$ by deleting its last component. Then,*

1. *The rank of AV_m is equal to the rank of R_m. In particular, if $r_{mm} = 0$ then A must be singular.*
2. *The vector y_m which minimizes $\|\beta e_1 - \bar{H}_m y\|_2$ is given by*
$$y_m = R_m^{-1} g_m.$$
3. *The residual vector at step m satisfies*
$$b - Ax_m = V_{m+1}\left(\beta e_1 - \bar{H}_m y_m\right) = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1}) \tag{6.35}$$
and, as a result,
$$\|b - Ax_m\|_2 = |\gamma_{m+1}|. \tag{6.36}$$

Proof. To prove first part (1), use (6.5), to obtain the relation

$$\begin{aligned} AV_m &= V_{m+1}\bar{H}_m \\ &= V_{m+1} Q_m^T Q_m \bar{H}_m \\ &= V_{m+1} Q_m^T \bar{R}_m. \end{aligned}$$

Since $V_{m+1}Q_m^T$ is unitary, the rank of AV_m is that of $\bar{R}_m$, which equals the rank of R_m since these two matrices differ only by a zero row (the last row of $\bar{R}_m$). If $r_{mm} = 0$ then R_m is of rank $\leq m-1$ and as a result AV_m is also of rank $\leq m-1$. Since V_m is of full rank, this means that A must be singular.

The second part (2), was essentially proved before the proposition. For any vector y,

$$\begin{aligned} \|\beta e_1 - \bar{H}_m y\|_2^2 &= \|Q_m(\beta e_1 - \bar{H}_m y)\|_2^2 \\ &= \|\bar{g}_m - \bar{R}_m y\|_2^2 \\ &= |\gamma_{m+1}|^2 + \|g_m - R_m y\|_2^2 \end{aligned} \tag{6.37}$$

The minimum of the left-hand side is reached when the second term in the right-hand side of (6.37) is zero. Since R_m is nonsingular, this is achieved when $y = R_m^{-1} g_m$.

To prove the third part (3), we start with the definitions used for GMRES and the relation (6.21). For any $x = x_0 + V_m y$,

$$\begin{aligned} b - Ax &= V_{m+1}\left(\beta e_1 - \bar{H}_m y\right) \\ &= V_{m+1} Q_m^T \, Q_m \left(\beta e_1 - \bar{H}_m y\right) \\ &= V_{m+1} Q_m^T \left(\bar{g}_m - \bar{R}_m y\right). \end{aligned}$$

As was seen in the proof of the second part above, the 2-norm of $\bar{g}_m - \bar{R}_m y$ is minimized when y annihilates all components of the right-hand side $\bar{g}_m$ except the last one, which is equal to γ_{m+1}. As a result,

$$b - Ax_m = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1})$$

which is (6.35). The result (6.36) follows from the orthonormality of the column-vectors of $V_{m+1}Q_m^T$. ■

So far we have only described a process for computing the least-squares solution y_m

of (6.24). Note that this approach with plane rotations can also be used to solve the linear system (6.15) for the FOM method. The only difference is that the last rotation Ω_m must be omitted. In particular, a single program can be written to implement both algorithms using a switch for selecting the FOM or GMRES options.

It is possible to implement the above process in a progressive manner, i.e., at each step of the GMRES algorithm. This approach will allow one to obtain the residual norm at every step, with virtually no additional arithmetic operations. To illustrate this, start with (6.30), i.e., assume that the first m rotations have already been applied. Now the residual norm is available for x_5 and the stopping criterion can be applied. Assume that the test dictates that further steps be taken. One more step of the Arnoldi algorithm must be executed to get Av_6 and the 6-th column of $\bar{H}_6$. This column is appended to $\bar{R}_5$ which has been augmented by a zero row to match the dimension. Then the previous rotations $\Omega_1, \Omega_2, \ldots, \Omega_5$ are applied to this last column. After this is done the following matrix and right-hand side are obtained:

$$H_6^{(5)} = \begin{pmatrix} h_{11}^{(5)} & h_{12}^{(5)} & h_{13}^{(5)} & h_{14}^{(5)} & h_{15}^{(5)} & h_{16}^{(5)} \\ & h_{22}^{(5)} & h_{23}^{(5)} & h_{24}^{(5)} & h_{25}^{(5)} & h_{26}^{(5)} \\ & & h_{33}^{(5)} & h_{34}^{(5)} & h_{35}^{(5)} & h_{36}^{(5)} \\ & & & h_{44}^{(5)} & h_{45}^{(5)} & h_{46}^{(5)} \\ & & & & h_{55}^{(5)} & h_{56}^{(5)} \\ & & & & 0 & h_{66}^{(5)} \\ & & & & 0 & h_{76} \end{pmatrix}, \quad g_6^{(5)} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ . \\ . \\ \gamma_6 \\ 0 \end{pmatrix}. \tag{6.38}$$

The algorithm now continues in the same way as before. We need to premultiply the matrix by a rotation matrix Ω_6 (now of size 7×7) with

$$s_6 = \frac{h_{76}}{\sqrt{(h_{66}^{(5)})^2 + h_{76}^2}}, \quad c_6 = \frac{h_{66}^{(5)}}{\sqrt{(h_{66}^{(5)})^2 + h_{76}^2}}$$

to get the matrix and right-hand side,

$$\bar{R}_6 = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} & r_{16} \\ & r_{22} & r_{23} & r_{24} & r_{25} & r_{26} \\ & & r_{33} & r_{34} & r_{35} & r_{36} \\ & & & r_{44} & r_{45} & r_{46} \\ & & & & r_{55} & r_{56} \\ & & & & & r_{66} \\ & & & & & 0 \end{pmatrix}, \quad \bar{g}_6 = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ . \\ . \\ c_6\gamma_6 \\ -s_6\gamma_6 \end{pmatrix}. \tag{6.39}$$

If the residual norm as given by $|\gamma_{m+1}|$ is small enough, the process must be stopped. The last rows of $\bar{R}_m$ and $\bar{g}_m$ are deleted and the resulting upper triangular system is solved to obtain y_m. Then the approximate solution $x_m = x_0 + V_m y_m$ is computed.

Note from (6.39) that the following useful relation for γ_{j+1} results

$$\gamma_{j+1} = -s_j \gamma_j. \tag{6.40}$$

In particular, if $s_j = 0$ then the residual norm must be equal to zero which means that the solution is exact at step j.

6.5.4 BREAKDOWN OF GMRES

If Algorithm 6.9 is examined carefully, we observe that the only possibilities of breakdown in GMRES are in the Arnoldi loop, when $\hat{v}_{j+1} = 0$, i.e., when $h_{j+1,j} = 0$ at a given step j. In this situation, the algorithm stops because the next Arnoldi vector cannot be generated. However, in this situation, the residual vector is zero, i.e., the algorithm will deliver the exact solution at this step. In fact, the converse is also true: If the algorithm stops at step j with $b - Ax_j = 0$, then $h_{j+1,j} = 0$.

PROPOSITION 6.10 *Let A be a nonsingular matrix. Then, the GMRES algorithm breaks down at step j, i.e., $h_{j+1,j} = 0$, if and only if the approximate solution x_j is exact.*

Proof. To show the necessary condition, observe that if $h_{j+1,j} = 0$, then $s_j = 0$. Indeed, since A is nonsingular, then $r_{jj} = h_{jj}^{(j-1)}$ is nonzero by the first part of Proposition 6.9 and (6.31) implies $s_j = 0$. Then, the relations (6.36) and (6.40) imply that $r_j = 0$.

To show the sufficient condition, we use (6.40) again. Since the solution is exact at step j and not at step $j - 1$, then $s_j = 0$. From the formula (6.31), this implies that $h_{j+1,j} = 0$. ■

6.5.5 RELATIONS BETWEEN FOM AND GMRES

If the last row of the least-squares system in (6.38) is deleted, instead of the one in (6.39), i.e., before the last rotation Ω_6 is applied, the same approximate solution as FOM would result. As a practical consequence a single subroutine can be written to handle both cases. This observation can also be helpful in understanding the relationships between the two algorithms.

We begin by establishing an interesting relation between the FOM and GMRES iterates, which will be exploited in the next chapter. A general lemma is first shown regarding the solutions of the triangular systems

$$R_m y_m = g_m$$

obtained from applying successive rotations to the Hessenberg matrices $\bar{H}_m$. As was stated before, the only difference between the y_m vectors obtained in GMRES and Arnoldi is that the last rotation Ω_m is omitted in FOM. In other words, the R_m matrix for the two methods differs only in its (m, m) entry while the right-hand sides differ only in their last components.

LEMMA 6.1 *Let $\tilde{R}_m$ be the $m \times m$ upper part of the matrix $Q_{m-1}\bar{H}_m$ and, as before, let R_m be the $m \times m$ upper part of the matrix $Q_m\bar{H}_m$. Similarly, let $\tilde{g}_m$ be the vector of the first m components of $Q_{m-1}(\beta e_1)$ and let g_m be the vector of the first m components of $Q_m(\beta e_1)$. Define*

$$\tilde{y}_m = \tilde{R}_m^{-1}\tilde{g}_m, \quad y_m = R_m^{-1}g_m$$

the y vectors obtained for an m-dimensional FOM and GMRES methods, respectively.

Then

$$y_m - \binom{y_{m-1}}{0} = c_m^2 \left(\tilde{y}_m - \binom{y_{m-1}}{0} \right) \tag{6.41}$$

in which c_m is the cosine used in the m-th rotation Ω_m, as defined by (6.31).

Proof. The following relation holds:

$$R_m = \begin{pmatrix} R_{m-1} & z_m \\ 0 & \xi_m \end{pmatrix}, \quad \tilde{R}_m = \begin{pmatrix} R_{m-1} & z_m \\ 0 & \tilde{\xi}_m \end{pmatrix}.$$

Similarly, for the right-hand sides,

$$g_m = \begin{pmatrix} g_{m-1} \\ \gamma_m \end{pmatrix}, \quad \tilde{g}_m = \begin{pmatrix} g_{m-1} \\ \tilde{\gamma}_m \end{pmatrix}$$

with

$$\gamma_m = c_m \tilde{\gamma}_m. \tag{6.42}$$

Denoting by λ the scalar $\sqrt{\tilde{\xi}_m^2 + h_{m+1,m}^2}$, and using the definitions of s_m and c_m, we obtain

$$\xi_m = c_m \tilde{\xi}_m + s_m h_{m+1,m} = \frac{\tilde{\xi}_m^2}{\lambda} + \frac{h_{m+1,m}^2}{\lambda} = \lambda = \frac{\tilde{\xi}_m}{c_m}. \tag{6.43}$$

Now,

$$y_m = R_m^{-1} g_m = \begin{pmatrix} R_{m-1}^{-1} & -\frac{1}{\xi_m} R_{m-1}^{-1} z_m \\ 0 & \frac{1}{\xi_m} \end{pmatrix} \begin{pmatrix} g_{m-1} \\ \gamma_m \end{pmatrix} \tag{6.44}$$

which, upon observing that $R_{m-1}^{-1} g_{m-1} = y_{m-1}$, yields,

$$y_m - \binom{y_{m-1}}{0} = \frac{\gamma_m}{\xi_m} \begin{pmatrix} -R_{m-1}^{-1} z_m \\ 1 \end{pmatrix}. \tag{6.45}$$

Replacing y_m, ξ_m, γ_m by $\tilde{y}_m, \tilde{\xi}_m, \tilde{\gamma}_m$, respectively, in (6.44), a relation similar to (6.45) would result except that γ_m/ξ_m is replaced by $\tilde{\gamma}_m/\tilde{\xi}_m$ which, by (6.42) and (6.43), satisfies the relation

$$\frac{\gamma_m}{\xi_m} = c_m^2 \frac{\tilde{\gamma}_m}{\tilde{\xi}_m}.$$

The result follows immediately. ∎

If the FOM and GMRES iterates are denoted by the superscripts F and G, respectively, then the relation (6.41) implies that

$$x_m^G - x_{m-1}^G = c_m^2 \left(x_m^F - x_{m-1}^G \right),$$

or,

$$x_m^G = s_m^2 x_{m-1}^G + c_m^2 x_m^F. \tag{6.46}$$

This leads to the following relation for the residual vectors obtained by the two methods,

$$r_m^G = s_m^2 r_{m-1}^G + c_m^2 r_m^F \tag{6.47}$$

which indicates that, in general, the two residual vectors will evolve hand in hand. In particular, if $c_m = 0$, then GMRES will not progress at step m, a phenomenon known as stagnation. However, in this situation, according to the definitions (6.31) of the rotations, $h_{mm}^{(m-1)} = 0$ which implies that H_m is singular and, therefore, x_m^F is not defined. In fact, the reverse of this is also true, a result due to Brown [43], which is stated without proof in the following proposition.

PROPOSITION 6.11 *If at any given step m, the GMRES iterates make no progress, i.e., if $x_m^G = x_{m-1}^G$ then H_m is singular and x_m^F is not defined. Conversely, if H_m is singular at step m, i.e., if FOM breaks down at step m, and A is nonsingular, then $x_m^G = x_{m-1}^G$.*

Note also that the use of the above lemma is not restricted to the GMRES-FOM pair. Some of the iterative methods defined in this chapter and the next involve a least-squares problem of the form (6.24). In such cases, the iterates of the least-squares method and those of the orthogonal residual (Galerkin) method will be related by the same equation.

Another important observation from (6.40) is that if ρ_i is the residual norm $\|b - Ax_i\|_2$ obtained at step i, then

$$\rho_m^G = |s_m| \rho_{m-1}^G.$$

The superscripts G and F are used again to distinguish between GMRES and FOM quantities. A consequence of this is that,

$$\rho_m^G = |s_1 s_2 \ \dots \ s_m| \beta. \tag{6.48}$$

Now consider the FOM iterates, assuming that x_m is defined, i.e., that H_m is nonsingular. An equation similar to (6.48) for FOM can be derived. Using the same notation as in the proof of the lemma, and recalling that

$$\rho_m^F = h_{m+1,m} |e_m^T H_m^{-1} (\beta e_1)|,$$

note that

$$e_m^T H_m^{-1} (\beta e_1) = \frac{\tilde{\gamma}_m}{\tilde{\xi}_m}.$$

Clearly,

$$|\tilde{\gamma}_m| = |s_{m-1} \gamma_{m-1}| = \cdots = |s_1 s_2 \dots s_{m-1} \beta|$$

and therefore,

$$\rho_m^F = \frac{h_{m+1,m}}{|\tilde{\xi}_m|} |s_1 s_2 \dots s_{m-1} \beta|.$$

Using (6.31), observe that $h_{m+1,m}/|\tilde{\xi}_m|$ is the tangent of the angle defining the m-th rotation, and therefore,

$$\rho_m^F = \frac{|s_m| \sqrt{\tilde{\xi}_m^2 + h_{m+1,m}^2}}{|\tilde{\xi}_m|} |s_1 s_2 \dots s_{m-1} \beta|$$

which, by a comparison with (6.48), yields a revealing relation between the residuals of the FOM and GMRES algorithms, namely,

$$\rho_m^F = \frac{1}{c_m}\rho_m^G \; = \rho_m^G \; \sqrt{1 + \frac{h_{m+1,m}^2}{\tilde{\xi}_m^2}}.$$

Another way to prove the above expression is to exploit the relation (6.47); see Exercise 12. These results are summarized in the following proposition (Brown [43]).

PROPOSITION 6.12 *Assume that m steps of the Arnoldi process have been taken and that H_m is nonsingular. Let $\xi \equiv (Q_{m-1}\bar{H}_m)_{mm}$ and $h \equiv h_{m+1,m}$. Then the residual norms produced by the FOM and the GMRES algorithms are related by the equality*

$$\rho_m^F = \frac{1}{c_m}\rho_m^G = \rho_m^G \; \sqrt{1 + \frac{h^2}{\xi^2}}. \tag{6.49}$$

6.5.6 VARIATION 1: RESTARTING

Similar to the FOM algorithm of the previous section, the GMRES algorithm becomes impractical when m is large because of the growth of memory and computational requirements as m increases. These requirements are identical with those of FOM. As with FOM, there are two remedies. One is based on restarting and the other on truncating the Arnoldi orthogonalization. The straightforward restarting option is described here.

ALGORITHM 6.11: Restarted GMRES

1. *Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, and $v_1 = r_0/\beta$*
2. *Generate the Arnoldi basis and the matrix $\bar{H}_m$ using the Arnoldi algorithm*
3. *starting with v_1*
4. *Compute y_m which minimizes $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m = x_0 + V_m y_m$*
5. *If satisfied then Stop, else set $x_0 := x_m$ and GoTo 1*

Note that the implementation tricks discussed in the previous section can be applied, providing the residual norm at each sub-step j without computing the approximation x_j. This enables the program to exit as soon as this norm is small enough.

A well known difficulty with the restarted GMRES algorithm is that it can *stagnate* when the matrix is not positive definite. The full GMRES algorithm is guaranteed to converge in at most n steps, but this would be impractical if there were many steps required for convergence. Obviously, a preconditioner for the linear system can be used to reduce the number of steps, or a better preconditioner if one is already in use. This issue will be covered later along with preconditioning techniques.

Example 6.2 Table 6.2 shows the results of applying the GMRES algorithm with no preconditioning to three of the test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	95	3841	0.32E-02	0.11E-03
F3D	67	11862	0.37E-03	0.28E-03
ORS	205	9221	0.33E+00	0.68E-04

Table 6.2 *A test run of GMRES with no preconditioning.*

See Example 6.1 for the meaning of the column headers in the table. In this test, the dimension of the Krylov subspace is $m = 10$. Observe that the problem ORS, which could not be solved by FOM(10), is now solved in 205 steps.

6.5.7 VARIATION 2: TRUNCATED GMRES VERSIONS

It is possible to derive an Incomplete version of the GMRES algorithm. This algorithm is called Quasi-GMRES (QGMRES) for the sake of notational uniformity with other algorithms developed in the literature (some of which will be seen in the next chapter). A direct version called DQGMRES using exactly the same arguments as in Section 6.4.2 for DIOM can also be derived. We begin by defining the QGMRES algorithm, in simple terms, by replacing the Arnoldi Algorithm with Algorithm 6.6, the Incomplete Orthogonalization procedure.

ALGORITHM 6.12: Quasi-GMRES

Run a modification of Algorithm 6.9 in which the Arnoldi process in lines 3 to 11 is replaced by the Incomplete Orthogonalization process and all other computations remain unchanged.

Similar to IOM, only the k previous v_i vectors must be kept at any given step. However, this version of GMRES will potentially save computations but not storage. This is because computing the solution by formula (6.23) requires the vectors v_i for $i = 1, \ldots, m$ to be accessed. Fortunately, the approximate solution can be updated in a progressive manner, as in DIOM.

The implementation of this progressive version is quite similar to DIOM. First, note that if $\bar{H}_m$ is banded, as for example, when $m = 5, k = 2$,

$$\bar{H}_5 = \begin{pmatrix} h_{11} & h_{12} & & & \\ h_{21} & h_{22} & h_{23} & & \\ & h_{32} & h_{33} & h_{34} & \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix}, \quad g = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{6.50}$$

then the premultiplications by the rotation matrices Ω_i as described in the previous section will only introduce an additional diagonal. For the above case, the resulting least-squares system is $\bar{R}_5 y = \bar{g}_5$ with:

$$\bar{R}_5 = \begin{pmatrix} r_{11} & r_{12} & r_{13} & & \\ & r_{22} & r_{23} & r_{24} & \\ & & r_{33} & r_{34} & r_{35} \\ & & & r_{44} & r_{45} \\ & & & & r_{55} \\ & & & & 0 \end{pmatrix}, \qquad \bar{g}_5 = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ . \\ . \\ \gamma_6 \end{pmatrix}. \tag{6.51}$$

The approximate solution is given by

$$x_m = x_0 + V_m R_m^{-1} g_m$$

where R_m and g_m are obtained by removing the last row of $\bar{R}_m$ and $\bar{g}_m$, respectively. Defining P_m as in DIOM,

$$P_m \equiv V_m R_m^{-1}$$

then,

$$x_m = x_0 + P_m g_m.$$

Also note that similarly to DIOM,

$$g_m = \begin{bmatrix} g_{m-1} \\ \gamma_m \end{bmatrix}$$

in which

$$\gamma_m = c_m \gamma_m^{(m-1)},$$

where $\gamma_m^{(m-1)}$ is the last component of the vector $\bar{g}_{m-1}$, i.e., the right-hand side before the m-th rotation is applied. Thus, x_m can be updated at each step, via the relation

$$x_m = x_{m-1} + \gamma_m p_m.$$

ALGORITHM 6.13: DQGMRES

1. *Compute* $r_0 = b - Ax_0$, $\gamma_1 := \|r_0\|_2$, *and* $v_1 := r_0/\gamma_1$
2. *For* $m = 1, 2, \ldots,$ *until convergence Do:*
3. *Compute* h_{im}, $i = \max\{1, m-k+1\}, \ldots, m$ *and* v_{m+1}
4. *as in lines 2 to 6 of Algorithm 6.6*
5. *Update the QR factorization of* $\bar{H}_m$, *i.e.,*
6. *Apply* Ω_i, $i = m-k, \ldots, m-1$ *to the* m*-th column of* $\bar{H}_m$
7. *Compute the rotation coefficients* c_m, s_m *by (6.31)*
8. *Apply* Ω_m *to* $\bar{H}_m$ *and* $\bar{g}_m$, *i.e., Compute:*
9. $\gamma_{m+1} := -s_m \gamma_m$
10. $\gamma_m := c_m \gamma_m$
11. $h_{mm} := c_m h_{mm} + s_m h_{m+1,m} \quad \left(= \sqrt{h_{m+1,m}^2 + h_{mm}^2}\right)$
12. $p_m = \left(v_m - \sum_{i=m-k}^{m-1} h_{im} p_i\right) / h_{mm}$
13. $x_m = x_{m-1} + \gamma_m p_m$
14. *If* $|\gamma_{m+1}|$ *is small enough then Stop*
15. *EndDo*

The above algorithm does not minimize the norm of the residual vector over $x_0 + \mathcal{K}_m$. Rather, it attempts to perform an approximate minimization. The formula (6.35) is still valid since orthogonality is not used to derive it. Therefore,

$$b - Ax_m = V_{m+1} Q_m^T (\gamma_{m+1} e_{m+1}). \tag{6.52}$$

If the v_i's were orthogonal to each other, then this is equivalent to GMRES and the residual norm is minimized over all vectors of the form $x_0 + V_m y$. Since only an incomplete orthogonalization is used then the v_i's are only locally orthogonal and, as a result, only an approximate minimization may be obtained. In addition, (6.36) is no longer valid. This equality had been derived from the above equation by exploiting the orthogonality of the v_i's. It turns out that in practice, $|\gamma_{m+1}|$ remains a reasonably good estimate of the actual residual norm because the v_i's are nearly orthogonal. The following inequality provides an actual upper bound of the residual norm in terms of computable quantities:

$$\|b - Ax_m\| \leq \sqrt{m-k+1}\,|\gamma_{m+1}|. \tag{6.53}$$

Here, k is to be replaced by m when $m \leq k$. The proof of this inequality is a consequence of (6.52). If the unit vector $q \equiv Q_m^T e_{m+1}$ has components $\eta_1, \eta_2, \ldots, \eta_{m+1}$, then

$$\begin{aligned}
\|b - Ax_m\|_2 &= |\gamma_{m+1}|\ \|V_{m+1} q\|_2 \\
&\leq |\gamma_{m+1}| \left(\left\| \sum_{i=1}^{k+1} \eta_i v_i \right\|_2 + \left\| \sum_{i=k+2}^{m+1} \eta_i v_i \right\|_2 \right) \\
&\leq |\gamma_{m+1}| \left(\left[\sum_{i=1}^{k+1} \eta_i^2 \right]^{1/2} + \sum_{i=k+2}^{m+1} |\eta_i|\ \|v_i\|_2 \right) \\
&\leq |\gamma_{m+1}| \left(\left[\sum_{i=1}^{k+1} \eta_i^2 \right]^{1/2} + \sqrt{m-k} \left[\sum_{i=k+2}^{m+1} \eta_i^2 \right]^{1/2} \right)
\end{aligned}$$

Here, the orthogonality of the first $k+1$ vectors v_i was used and the last term comes from using the Cauchy-Schwartz inequality. The desired inequality follows from using the Cauchy-Schwartz inequality again in the form

$$1 \,.\, a + \sqrt{m-k} \,.\, b \leq \sqrt{m-k+1}\,\sqrt{a^2 + b^2}$$

and from the fact that the vector q is of norm unity. Thus, using $|\gamma_{m+1}|$ as a residual estimate, we would make an error of a factor of $\sqrt{m-k+1}$ at most. In general, this is an overestimate and $|\gamma_{m+1}|$ tends to give an adequate estimate for the residual norm.

It is also interesting to observe that with a little bit more arithmetic, it is possible to actually compute the exact residual vector and norm. This is based on the observation that, according to (6.52), the residual vector is γ_{m+1} times the vector z_{m+1} which is the last column of the matrix

$$Z_{m+1} \equiv V_{m+1} Q_m^T. \tag{6.54}$$

It is an easy exercise to see that this last column can be updated from v_{m+1} and z_m. Indeed,

$$Z_{m+1} = [V_m, v_{m+1}] Q_{m-1}^T \Omega_m$$

$$= [V_m Q_{m-1}^T, v_{m+1}]\Omega_m$$
$$= [Z_m, v_{m+1}]\Omega_m$$

where all the matrices related to the rotation are of size $(m+1) \times (m+1)$. The result is that

$$z_{m+1} = -s_m z_m + c_m v_{m+1}. \tag{6.55}$$

The z_i's can be updated at the cost of one extra vector in memory and $4n$ operations at each step. The norm of z_{m+1} can be computed at the cost of $2n$ operations and the exact residual norm for the current approximate solution can then be obtained by multiplying this norm by $|\gamma_{m+1}|$.

Because this is a little expensive, it may be preferred to just "correct" the estimate provided by γ_{m+1} by exploiting the above recurrence relation,

$$\|z_{m+1}\|_2 \leq |s_m| \|z_m\|_2 + |c_m|.$$

If $\zeta_m \equiv \|z_m\|_2$, then the following recurrence relation holds,

$$\zeta_{m+1} \leq |s_m| \zeta_m + |c_m|. \tag{6.56}$$

The above relation is inexpensive to update, yet provides an upper bound that is sharper than (6.53); see Exercise 20.

An interesting consequence of (6.55) is a relation between two successive residual vectors:

$$\begin{aligned} r_m &= \gamma_{m+1} z_{m+1} \\ &= \gamma_{m+1}[-s_m z_m + c_m v_{m+1}] \\ &= s_m^2 r_{m-1} + c_m \gamma_{m+1} v_{m+1}. \end{aligned} \tag{6.57}$$

This exploits the fact that $\gamma_{m+1} = -s_m \gamma_m$ and $r_j = \gamma_{j+1} z_{j+1}$.

Example 6.3 Table 6.3 shows the results of applying the DQGMRES algorithm with no preconditioning to three of the test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	98	7216	0.36E-02	0.13E-03
F3D	75	22798	0.64E-03	0.32E-03
ORS	300	24138	0.13E+02	0.25E-02

Table 6.3 *A test run of DQGMRES with no preconditioning.*

See Example 6.1 for the meaning of the column headers in the table. In this test the number k of directions in the recurrence is $k = 10$.

It is possible to relate the quasi-minimal residual norm to the actual minimal residual norm provided by GMRES. The following result was proved by Nachtigal (1991) [152] for the QMR algorithm to be seen in the next chapter.

THEOREM 6.1 *Assume that V_{m+1}, the Arnoldi basis associated with DQGMRES, is of full rank. Let r_m^Q and r_m^G be the residual norms obtained after m steps of the DQGMRES and GMRES algorithms, respectively. Then*

$$\|r_m^Q\|_2 \leq \kappa_2(V_{m+1})\|r_m^G\|_2. \tag{6.58}$$

Proof. Consider the subset of $\mathcal{K}_{m+1}$ defined by

$$\mathcal{R} = \{r : r = V_{m+1}t;\ t = \beta e_1 - \bar{H}_m y;\ y\ \in \mathbb{C}^m\}.$$

Denote by y_m the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ over y and $t_m = \beta e_1 - \bar{H}_m y_m$, $r_m = V_{m+1}t_m \equiv r_m^Q$. By assumption, V_{m+1} is of full rank and there is an $(m+1)\times(m+1)$ nonsingular matrix S such that $W_{m+1} = V_{m+1}S$ is unitary. Then, for any member of $\mathcal{R}$,

$$r = W_{m+1}S^{-1}t, \quad t = SW_{m+1}^H r$$

and, in particular,

$$\|r_m\|_2 \leq \|S^{-1}\|_2\|t_m\|_2. \tag{6.59}$$

Now $\|t_m\|_2$ is the minimum of the 2-norm of $\beta e_1 - \bar{H}_m y$ over all y's and therefore,

$$\begin{aligned}\|t_m\|_2 = \|SW_{m+1}^H r_m\| &\leq \|SW_{m+1}^H r\|_2 \quad \forall r\ \in \mathcal{R}\\ &\leq \|S\|_2\|r\|_2 \quad \forall r\ \in \mathcal{R}\\ &\leq \|S\|_2\|r^G\|_2.\end{aligned} \tag{6.60}$$

The result follows from (6.59), (6.60), and the fact that $\kappa_2(V_{m+1}) = \kappa_2(S)$. ∎

6.6 THE SYMMETRIC LANCZOS ALGORITHM

The symmetric Lanczos algorithm can be viewed as a simplification of Arnoldi's method for the particular case when the matrix is symmetric. When A is symmetric, then the Hessenberg matrix H_m becomes symmetric tridiagonal. This leads to a three-term recurrence in the Arnoldi process and short-term recurrences for solution algorithms such as FOM and GMRES. On the theoretical side, there is also much more to be said on the resulting approximation in the symmetric case.

6.6.1 THE ALGORITHM

To introduce the Lanczos algorithm we begin by making the observation stated in the following theorem.

THEOREM 6.2 *Assume that Arnoldi's method is applied to a real symmetric matrix A. Then the coefficients h_{ij} generated by the algorithm are such that*

$$h_{ij} = 0, \quad \text{for} \quad 1 \leq i < j-1, \tag{6.61}$$

$$h_{j,j+1} = h_{j+1,j}, \; j = 1, 2, \ldots, m. \tag{6.62}$$

In other words, the matrix H_m obtained from the Arnoldi process is tridiagonal and symmetric.

Proof. The proof is an immediate consequence of the fact that $H_m = V_m^T A V_m$ is a symmetric matrix which is also a Hessenberg matrix by construction. Therefore, H_m must be a symmetric tridiagonal matrix. ∎

The standard notation used to describe the Lanczos algorithm is obtained by setting

$$\alpha_j \equiv h_{jj}, \qquad \beta_j \equiv h_{j-1,j},$$

and if T_m denotes the resulting H_m matrix, it is of the form,

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & . & . & . & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}. \tag{6.63}$$

This leads to the following form of the Modified Gram-Schmidt variant of Arnoldi's method, namely, Algorithm 6.2.

ALGORITHM 6.14: The Lanczos Algorithm

1. *Choose an initial vector* v_1 *of norm unity. Set* $\beta_1 \equiv 0, v_0 \equiv 0$
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. $\quad w_j := Av_j - \beta_j v_{j-1}$
4. $\quad \alpha_j := (w_j, v_j)$
5. $\quad w_j := w_j - \alpha_j v_j$
6. $\quad \beta_{j+1} := \|w_j\|_2$. *If* $\beta_{j+1} = 0$ *then Stop*
7. $\quad v_{j+1} := w_j / \beta_{j+1}$
8. *EndDo*

It is rather surprising that the above simple algorithm guarantees, at least in exact arithmetic, that the vectors $v_i, i = 1, 2, \ldots,$ are orthogonal. In reality, exact orthogonality of these vectors is only observed at the beginning of the process. At some point the v_i's start losing their global orthogonality rapidly. There has been much research devoted to finding ways to either recover the orthogonality, or to at least diminish its effects by *partial* or *selective* orthogonalization; see Parlett [160].

The major practical differences with Arnoldi's method are that the matrix H_m is tridiagonal and, more importantly, that only three vectors must be saved, unless some form of reorthogonalization is employed.

6.6.2 RELATION WITH ORTHOGONAL POLYNOMIALS

In exact arithmetic, the core of Algorithm 6.14 is a relation of the form

$$\beta_{j+1}v_{j+1} = Av_j - \alpha_j v_j - \beta_{j-1}v_{j-1}.$$

This three-term recurrence relation is reminiscent of the standard three-term recurrence relation of orthogonal polynomials. In fact, there is indeed a strong relationship between the Lanczos algorithm and orthogonal polynomials. To begin, recall that if the grade of v_1 is $\geq m$, then the subspace $\mathcal{K}_m$ is of dimension m and consists of all vectors of the form $q(A)v_1$, where q is a polynomial with $\text{degree}(q) \leq m-1$. In this case there is even an isomorphism between $\mathcal{K}_m$ and $\mathbb{P}_{m-1}$, the space of polynomials of degree $\leq m-1$, which is defined by

$$q \in \mathbb{P}_{m-1} \rightarrow x = q(A)v_1 \in \mathcal{K}_m.$$

Moreover, we can consider that the subspace $\mathbb{P}_{m-1}$ is provided with the inner product

$$< p, q >_{v_1} = (p(A)v_1, q(A)v_1). \tag{6.64}$$

This is indeed a nondegenerate bilinear form under the assumption that m does not exceed μ, the grade of v_1. Now observe that the vectors v_i are of the form

$$v_i = q_{i-1}(A)v_1$$

and the orthogonality of the v_i's translates into the orthogonality of the polynomials with respect to the inner product (6.64). It is known that real orthogonal polynomials satisfy a three-term recurrence. Moreover, the Lanczos procedure is nothing but the Stieltjes algorithm; (see, for example, Gautschi [102]) for computing a sequence of orthogonal polynomials with respect to the inner product (6.64). It is known [180] that the characteristic polynomial of the tridiagonal matrix produced by the Lanczos algorithm minimizes the norm $\|.\|_{v_1}$ over the monic polynomials. The recurrence relation between the characteristic polynomials of tridiagonal matrices also shows that the Lanczos recurrence computes the sequence of vectors $p_{T_m}(A)v_1$, where p_{T_m} is the characteristic polynomial of T_m.

6.7 THE CONJUGATE GRADIENT ALGORITHM

The Conjugate Gradient algorithm is one of the best known iterative techniques for solving sparse Symmetric Positive Definite linear systems. Described in one sentence, the method is a realization of an orthogonal projection technique onto the Krylov subspace $\mathcal{K}_m(r_0, A)$ where r_0 is the initial residual. It is therefore mathematically equivalent to FOM. However, because A is symmetric, some simplifications resulting from the three-term Lanczos recurrence will lead to more elegant algorithms.

6.7.1 DERIVATION AND THEORY

We first derive the analogue of FOM, or Arnoldi's method, for the case when A is symmetric. Given an initial guess x_0 to the linear system $Ax = b$ and the Lanczos vectors $v_i, i = 1, \ldots, m$ together with the tridiagonal matrix T_m, the approximate solution obtained from an orthogonal projection method onto $\mathcal{K}_m$, is given by

$$x_m = x_0 + V_m y_m, \quad y_m = T_m^{-1}(\beta e_1). \tag{6.65}$$

ALGORITHM 6.15: Lanczos Method for Linear Systems

1. *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. $\quad w_j = Av_j - \beta_j v_{j-1}$ *(If* $j = 1$ *set* $\beta_1 v_0 \equiv 0$*)*
4. $\quad \alpha_j = (w_j, v_j)$
5. $\quad w_j := w_j - \alpha_j v_j$
6. $\quad \beta_{j+1} = \|w_j\|_2$. *If* $\beta_{j+1} = 0$ *set* $m := j$ *and go to 9*
7. $\quad v_{j+1} = w_j / \beta_{j+1}$
8. *EndDo*
9. *Set* $T_m = \mathrm{tridiag}\,(\beta_i, \alpha_i, \beta_{i+1})$, *and* $V_m = [v_1, \ldots, v_m]$.
10. *Compute* $y_m = T_m^{-1}(\beta e_1)$ *and* $x_m = x_0 + V_m y_m$

Many of the results obtained from Arnoldi's method for linear systems are still valid. For example, the residual vector of the approximate solution x_m is such that

$$b - Ax_m = -\beta_{m+1} e_m^T y_m v_{m+1}. \tag{6.66}$$

The Conjugate Gradient algorithm can be derived from the Lanczos algorithm in the same way DIOM was derived from IOM. In fact, the Conjugate Gradient algorithm can be viewed as a variation of DIOM(2) for the case when A is symmetric. We will follow the same steps as with DIOM, except that the notation will be simplified whenever possible.

First write the LU factorization of T_m as $T_m = L_m U_m$. The matrix L_m is unit lower bidiagonal and U_m is upper bidiagonal. Thus, the factorization of T_m is of the form

$$T_m = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \lambda_4 & 1 & \\ & & & \lambda_5 & 1 \end{pmatrix} \times \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \beta_4 & \\ & & & \eta_4 & \beta_5 \\ & & & & \eta_5 \end{pmatrix}.$$

The approximate solution is then given by,

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1).$$

Letting

$$P_m \equiv V_m U_m^{-1}$$

and

$$z_m = L_m^{-1} \beta e_1,$$

then,

$$x_m = x_0 + P_m z_m.$$

As for DIOM, p_m, the last column of P_m, can be computed from the previous p_i's and v_m by the simple update

$$p_m = \eta_m^{-1}[v_m - \beta_m p_{m-1}].$$

Note that β_m is a scalar computed from the Lanczos algorithm, while η_m results from the m-th Gaussian elimination step on the tridiagonal matrix, i.e.,

$$\lambda_m = \frac{\beta_m}{\eta_{m-1}}, \tag{6.67}$$

$$\eta_m = \alpha_m - \lambda_m \beta_m. \tag{6.68}$$

In addition, following again what has been shown for DIOM,

$$z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix},$$

in which $\zeta_m = -\lambda_m \zeta_{m-1}$. As a result, x_m can be updated at each step as

$$x_m = x_{m-1} + \zeta_m p_m$$

where p_m is defined above.

This gives the following algorithm, which we call the direct version of the Lanczos algorithm for linear systems.

ALGORITHM 6.16: D-Lanczos

1. *Compute* $r_0 = b - Ax_0$, $\zeta_1 := \beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2. *Set* $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3. *For* $m = 1, 2, \ldots$, *until convergence Do:*
4. *Compute* $w := Av_m - \beta_m v_{m-1}$ *and* $\alpha_m = (w, v_m)$
5. *If* $m > 1$ *then compute* $\lambda_m = \frac{\beta_m}{\eta_{m-1}}$ *and* $\zeta_m = -\lambda_m \zeta_{m-1}$
6. $\eta_m = \alpha_m - \lambda_m \beta_m$
7. $p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1})$
8. $x_m = x_{m-1} + \zeta_m p_m$
9. *If* x_m *has converged then Stop*
10. $w := w - \alpha_m v_m$
11. $\beta_{m+1} = \|w\|_2$, $v_{m+1} = w/\beta_{m+1}$
12. *EndDo*

This algorithm computes the solution of the tridiagonal system $T_m y_m = \beta e_1$ progressively by using Gaussian elimination without pivoting. However, as was explained for DIOM, partial pivoting can also be implemented at the cost of having to keep an extra vector. In fact, Gaussian elimination with partial pivoting is sufficient to ensure stability for tridiagonal systems. The more complex LQ factorization has also been exploited in this context and gave rise to an algorithm known as SYMMLQ [159].

The two algorithms 6.15 and 6.16 are mathematically equivalent, that is, they deliver the same approximate solution if they are both executable. However, since Gaussian elimination without pivoting is being used implicitly to solve the tridiagonal system $T_m y = \beta e_1$, the direct version may be more prone to breakdowns.

Observe that the residual vector for this algorithm is in the direction of v_{m+1} due to equation (6.66). Therefore, the residual vectors are orthogonal to each other as in FOM. Likewise, the vectors p_i are A-orthogonal, or *conjugate*. These results are established in the next proposition.

PROPOSITION 6.13 *Let $r_m = b - Ax_m$, $m = 0, 1, \ldots$, be the residual vectors produced by the Lanczos and the D-Lanczos algorithms (6.15 and 6.16) and p_m, $m = 0, 1, \ldots,$ the auxiliary vectors produced by Algorithm 6.16. Then,*

1. *Each residual vector r_m is such that $r_m = \sigma_m v_{m+1}$ where σ_m is a certain scalar. As a result, the residual vectors are orthogonal to each other.*
2. *The auxiliary vectors p_i form an A-conjugate set, i.e., $(Ap_i, p_j) = 0$, for $i \neq j$.*

Proof. The first part of the proposition is an immediate consequence of the relation (6.66). For the second part, it must be proved that $P_m^T A P_m$ is a diagonal matrix, where $P_m = V_m U_m^{-1}$. This follows from

$$\begin{aligned} P_m^T A P_m &= U_m^{-T} V_m^T A V_m U_m^{-1} \\ &= U_m^{-T} T_m U_m^{-1} \\ &= U_m^{-T} L_m. \end{aligned}$$

Now observe that $U_m^{-T} L_m$ is a lower triangular which is also symmetric since it is equal to the symmetric matrix $P_m^T A P_m$. Therefore, it must be a diagonal matrix. ■

A consequence of the above proposition is that a version of the algorithm can be derived by imposing the orthogonality and conjugacy conditions. This gives the Conjugate Gradient algorithm which we now derive. The vector x_{j+1} can be expressed as

$$x_{j+1} = x_j + \alpha_j p_j. \tag{6.69}$$

Therefore, the residual vectors must satisfy the recurrence

$$r_{j+1} = r_j - \alpha_j A p_j. \tag{6.70}$$

If the r_j's are to be orthogonal, then it is necessary that $(r_j - \alpha_j A p_j, r_j) = 0$ and as a result

$$\alpha_j = \frac{(r_j, r_j)}{(Ap_j, r_j)}. \tag{6.71}$$

Also, it is known that the next search direction p_{j+1} is a linear combination of r_{j+1} and p_j, and after rescaling the p vectors appropriately, it follows that

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{6.72}$$

Thus, a first consequence of the above relation is that

$$(Ap_j, r_j) = (Ap_j, p_j - \beta_{j-1} p_{j-1}) = (Ap_j, p_j)$$

because Ap_j is orthogonal to p_{j-1}. Then, (6.71) becomes $\alpha_j = (r_j, r_j)/(Ap_j, p_j)$. In addition, writing that p_{j+1} as defined by (6.72) is orthogonal to Ap_j yields

$$\beta_j = -\frac{(r_{j+1}, Ap_j)}{(p_j, Ap_j)}.$$

Note that from (6.70)

$$Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j) \tag{6.73}$$

and therefore,

$$\beta_j = \frac{1}{\alpha_j}\frac{(r_{j+1}, (r_{j+1} - r_j))}{(Ap_j, p_j)} = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}.$$

Putting these relations together gives the following algorithm.

ALGORITHM 6.17: Conjugate Gradient

1. *Compute* $r_0 := b - Ax_0$, $p_0 := r_0$.
2. *For* $j = 0, 1, \ldots$, *until convergence Do:*
3. $\quad\alpha_j := (r_j, r_j)/(Ap_j, p_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad r_{j+1} := r_j - \alpha_j Ap_j$
6. $\quad\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
7. $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
8. *EndDo*

It is important to note that the scalars α_j, β_j in this algorithm are different from those of the Lanczos algorithm. The vectors p_j are multiples of the p_j's of Algorithm 6.16.

In terms of storage, in addition to the matrix A, four vectors (x, p, Ap, and r) must be saved in Algorithm 6.17, versus five vectors (v_m, v_{m-1}, w, p, and x) for Algorithm 6.16.

6.7.2 ALTERNATIVE FORMULATIONS

Algorithm 6.17 is the best known formulation of the Conjugate Gradient algorithm. There are, however, several alternative formulations. Here, only one such formulation is shown, which can be derived once more from the Lanczos algorithm.

The residual polynomial $r_m(t)$ associated with the m-th CG iterate must satisfy a three-term recurrence, implied by the three-term recurrence of the Lanczos vectors. Indeed, these vectors are just the scaled versions of the residual vectors. Therefore, we must seek a three-term recurrence of the form

$$r_{m+1}(t) = \rho_m(r_m(t) - \gamma_m t r_m(t)) + \delta_m r_{m-1}(t).$$

In addition, the consistency condition $r_m(0) = 1$ must be maintained for each m, leading to the recurrence,

$$r_{m+1}(t) = \rho_m(r_m(t) - \gamma_m t r_m(t)) + (1 - \rho_m) r_{m-1}(t). \tag{6.74}$$

Observe that if $r_m(0) = 1$ and $r_{m-1}(0) = 1$, then $r_{m+1}(0) = 1$, as desired. Translating the above relation into the sequence of residual vectors yields

$$r_{m+1} = \rho_m(r_m - \gamma_m A r_m) + (1 - \rho_m) r_{m-1}. \tag{6.75}$$

Recall that the vectors r_i's are multiples of the Lanczos vectors v_i's. As a result, γ_m should be the inverse of the scalar α_m of the Lanczos algorithm. In terms of the r-vectors this means

$$\gamma_m = \frac{(r_m, r_m)}{(A r_m, r_m)}.$$

Equating the inner products of both sides of (6.75) with r_{m-1}, and using the orthogonality of the r-vectors, gives the following expression for ρ_m, after some algebraic calculations,

$$\rho_m = \left[1 - \frac{\gamma_m}{\gamma_{m-1}} \frac{(r_m, r_m)}{(r_{m-1}, r_{m-1})} \frac{1}{\rho_{m-1}}\right]^{-1}. \tag{6.76}$$

The recurrence relation for the approximate solution vectors can be extracted from the recurrence relation for the residual vectors. This is found by starting from (6.74) and using the relation $r_m(t) = 1 - t s_{m-1}(t)$ between the solution polynomial $s_{m-1}(t)$ and the residual polynomial $r_m(t)$. Thus,

$$\begin{aligned} s_m(t) &= \frac{1 - r_{m+1}(t)}{t} \\ &= \rho_m \left(\frac{1 - r_m(t)}{t} - \gamma_m r_m(t) \right) + (1 - \rho_m) \frac{1 - r_{m-1}(t)}{t} \\ &= \rho_m \left(s_{m-1}(t) - \gamma_m r_m(t) \right) + (1 - \rho_m) s_{m-2}(t). \end{aligned}$$

This gives the recurrence,

$$x_{m+1} = \rho_m(x_m - \gamma_m r_m) + (1 - \rho_m) x_{m-1}. \tag{6.77}$$

All that is left for the recurrence to be determined completely is to define the first two iterates. The initial iterate x_0 is given. The first vector should be of the form

$$x_1 = x_0 - \gamma_0 r_0,$$

to ensure that r_1 is orthogonal to r_0. This means that the two-term recurrence can be started with $\rho_0 = 1$, and by setting $x_{-1} \equiv 0$. Putting these relations and definitions together gives the following algorithm.

ALGORITHM 6.18: CG – Three-Term Recurrence Variant

1. *Compute* $r_0 = b - A x_0$. *Set* $x_{-1} \equiv 0$ *and* $\rho_0 = 1$.
2. *For* $j = 0, 1, \ldots,$ *until convergence Do:*
3. *Compute* $A r_j$ *and* $\gamma_j = \frac{(r_j, r_j)}{(A r_j, r_j)}$
4. *If* $(j > 0)$ *compute* $\rho_j = \left[1 - \frac{\gamma_j}{\gamma_{j-1}} \frac{(r_j, r_j)}{(r_{j-1}, r_{j-1})} \frac{1}{\rho_{j-1}}\right]^{-1}$
5. $x_{j+1} = \rho_j (x_j - \gamma_j r_j) + (1 - \rho_j) x_{j-1}$
6. *Compute* $r_{j+1} = \rho_j (r_j - \gamma_j A r_j) + (1 - \rho_j) r_{j-1}$
7. *EndDo*

The residual r_{j+1} could also be computed directly as $r_{j+1} = b - Ax_{j+1}$ in line 6 of the algorithm, but this would require an additional matrix-vector product.

6.7.3 EIGENVALUE ESTIMATES FROM THE CG COEFFICIENTS

Sometimes, it is useful to be able to obtain the tridiagonal matrix T_m related to the underlying Lanczos iteration from the coefficients of the Conjugate Gradient algorithm 6.17. This tridiagonal matrix can provide valuable eigenvalue information on the matrix A. For example, the largest and smallest eigenvalues of the tridiagonal matrix can approximate the smallest and largest eigenvalues of A. This could be used to compute an estimate of the condition number of A which in turn can help provide estimates of the error norm from the residual norm. Since the Greek letters α_i and β_i have been used in both algorithms, notations must be changed. Denote by

$$T_m = \text{tridiag}\,[\eta_j, \delta_j, \eta_{j+1}],$$

the tridiagonal matrix (6.63) associated with the m-th step of the Lanczos algorithm. We must seek expressions of the coefficients η_j, δ_j in terms of the coefficients α_j, β_j, obtained from the CG algorithm. The key information regarding the correspondence between the two pairs of coefficients resides in the correspondence between the vectors generated by the two algorithms. From (6.66) it is known that

$$r_j = \text{scalar} \times v_{j+1}. \tag{6.78}$$

As a result,

$$\delta_{j+1} = \frac{(Av_{j+1}, v_{j+1})}{(v_{j+1}, v_{j+1})} = \frac{(Ar_j, r_j)}{(r_j, r_j)}.$$

The denominator (r_j, r_j) is readily available from the coefficients of the CG algorithm, but the numerator (Ar_j, r_j) is not. The relation (6.72) can be exploited to obtain

$$r_j = p_j - \beta_{j-1} p_{j-1} \tag{6.79}$$

which is then substituted in (Ar_j, r_j) to get

$$(Ar_j, r_j) = (A(p_j - \beta_{j-1}p_{j-1}), p_j - \beta_{j-1}p_{j-1}).$$

Note that the terms $\beta_{j-1}p_{j-1}$ are defined to be zero when $j = 0$. Because the p vectors are A-orthogonal,

$$(Ar_j, r_j) = (Ap_j, p_j) + \beta_{j-1}^2 (Ap_{j-1}, p_{j-1}),$$

from which we finally obtain for $j > 0$,

$$\delta_{j+1} = \frac{(Ap_j, p_j)}{(r_j, r_j)} + \beta_{j-1}^2 \frac{(Ap_{j-1}, p_{j-1})}{(r_j, r_j)} = \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}}. \tag{6.80}$$

The above expression is only valid for $j > 0$. For $j = 0$, the second term in the right-hand side should be omitted as was observed above. Therefore, the diagonal elements of T_m are given by

$$\delta_{j+1} = \begin{cases} \frac{1}{\alpha_j} & \text{for} \quad j = 0, \\ \frac{1}{\alpha_j} + \frac{\beta_{j-1}}{\alpha_{j-1}} & \text{for} \quad j > 0. \end{cases} \tag{6.81}$$

Now an expression for the co-diagonal elements η_{j+1} is needed. From the definitions in the Lanczos algorithm,

$$\eta_{j+1} = (Av_j, v_{j+1}) = \frac{|(Ar_{j-1}, r_j)|}{\|r_{j-1}\|_2 \|r_j\|_2}.$$

Using (6.79) again and the relation (6.73) as well as orthogonality properties of the CG algorithm, the following sequence of equalities results:

$$\begin{aligned} (Ar_{j-1}, r_j) &= (A(p_{j-1} - \beta_{j-2} p_{j-2}), r_j) \\ &= (Ap_{j-1}, r_j) - \beta_{j-2}(Ap_{j-2}, r_j) \\ &= \frac{-1}{\alpha_{j-1}} (r_j - r_{j-1}, r_j) + \frac{\beta_{j-2}}{\alpha_{j-2}} (r_{j-1} - r_{j-2}, r_j) \\ &= \frac{-1}{\alpha_{j-1}} (r_j, r_j). \end{aligned}$$

Therefore,

$$\eta_{j+1} = \frac{1}{\alpha_{j-1}} \frac{(r_j, r_j)}{\|r_{j-1}\|_2 \|r_j\|_2} = \frac{1}{\alpha_{j-1}} \frac{\|r_j\|_2}{\|r_{j-1}\|_2} = \frac{\sqrt{\beta_{j-1}}}{\alpha_{j-1}}.$$

This finally gives the general form of the m-dimensional Lanczos tridiagonal matrix in terms of the CG coefficients,

$$T_m = \begin{pmatrix} \frac{1}{\alpha_0} & \frac{\sqrt{\beta_0}}{\alpha_0} & & & \\ \frac{\sqrt{\beta_0}}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & \frac{\sqrt{\beta_1}}{\alpha_1} & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} \\ & & & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}} + \frac{\beta_{m-2}}{\alpha_{m-2}} \end{pmatrix}. \tag{6.82}$$

6.8 THE CONJUGATE RESIDUAL METHOD

In the previous section we derived the Conjugate Gradient algorithm as a special case of FOM for Symmetric Positive Definite matrices. Similarly, a new algorithm can be derived from GMRES for the particular case where A is Hermitian. In this case, the residual vectors should be A-orthogonal, i.e., conjugate. In addition, the vectors Ap_i's $i = 0, 1, \ldots,$ are orthogonal. When looking for an algorithm with the same structure as CG, but satisfying these conditions, we find the Conjugate Residual algorithm. Notice that the residual vectors are now conjugate to each other, hence, the name of the algorithm.

ALGORITHM 6.19: Conjugate Residual Algorithm

1. *Compute* $r_0 := b - Ax_0$, $p_0 := r_0$
2. *For* $j = 0, 1, \ldots,$ *until convergence Do:*
3. $\quad \alpha_j := (r_j, Ar_j)/(Ap_j, Ap_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad r_{j+1} := r_j - \alpha_j Ap_j$
6. $\quad \beta_j := (r_{j+1}, Ar_{j+1})/(r_j, Ar_j)$
7. $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
8. $\quad$ *Compute* $Ap_{j+1} = Ar_{j+1} + \beta_j Ap_j$
9. *EndDo*

The last line in the above algorithm computes Ap_{j+1} from Ar_{j+1} without an additional matrix-vector product. Five vectors of storage are needed in addition to the matrix A: x, p, Ap, r, Ar. The algorithm requires one more vector update, i.e., $2n$ more operations than the Conjugate Gradient method and one more vector of storage. Since the two methods exhibit typically a similar convergence behavior, the Conjugate Gradient method is often preferred over the Conjugate Residual algorithm.

6.9 GCR, ORTHOMIN, AND ORTHODIR

All algorithms developed in this chapter are strongly related to, as well as defined by, the choice of a basis of the Krylov subspace. The GMRES algorithm uses an orthogonal basis. In the Conjugate Gradient algorithm, the p's are A-orthogonal, i.e., conjugate. In the Conjugate Residual method just described, the Ap_i's are orthogonal, i.e., the p_i's are A^TA-orthogonal. A number of algorithms can be developed using a basis of this form in the nonsymmetric case as well. The main result that is exploited in all these algorithms is the following lemma.

LEMMA 6.2 *Let* $p_0, p_1, \ldots, p_{m-1}$ *be a basis of the Krylov subspace* $\mathcal{K}_m(A, r_0)$ *which is* A^TA*-orthogonal, i.e., such that*

$$(Ap_i, Ap_j) = 0, \quad \text{for } i \neq j.$$

Then the approximate solution x_m *which has the smallest residual norm in the affine space* $x_0 + \mathcal{K}_m(A, r_0)$ *is given by*

$$x_m = x_0 + \sum_{i=0}^{m-1} \frac{(r_0, Ap_i)}{(Ap_i, Ap_i)} p_i. \tag{6.83}$$

In addition, x_m *can be computed from* x_{m-1} *by*

$$x_m = x_{m-1} + \frac{(r_{m-1}, Ap_{m-1})}{(Ap_{m-1}, Ap_{m-1})} p_{m-1}. \tag{6.84}$$

Proof. The approximate solution and the associated residual vector can be written in the form

$$x_m = x_0 + \sum_{i=0}^{m-1} \alpha_i p_i, \quad r_m = r_0 - \sum_{i=0}^{m-1} \alpha_i A p_i. \tag{6.85}$$

According to the optimality result of Proposition 5.3, in order for $\|r_m\|_2$ to be minimum, the orthogonality relations

$$(r_m, Ap_i) = 0, \quad i = 0, \ldots, m-1$$

must be enforced. Using (6.85) and the orthogonality of the Ap_i's gives immediately,

$$\alpha_i = (r_0, Ap_i)/(Ap_i, Ap_i).$$

This proves the first part of the lemma. Assume now that x_{m-1} is known and that x_m must be determined. According to formula (6.83), $x_m = x_{m-1} + \alpha_{m-1} p_{m-1}$ with α_{m-1} defined above. Note that from the second part of (6.85),

$$r_{m-1} = r_0 - \sum_{j=0}^{m-2} \alpha_j A p_j$$

so that

$$(r_{m-1}, Ap_{m-1}) = (r_0, Ap_{m-1}) - \sum_{j=0}^{m-2} \alpha_j (Ap_j, Ap_{m-1}) = (r_0, Ap_{m-1})$$

exploiting, once more, the orthogonality of the vectors Ap_j, $j = 0, \ldots, m-1$. Thus,

$$\alpha_{m-1} = \frac{(r_{m-1}, Ap_{m-1})}{(Ap_{m-1}, Ap_{m-1})},$$

which proves the expression (6.84). ■

This lemma opens up many different ways to obtain algorithms that are mathematically equivalent to the full GMRES. The simplest option computes the next basis vector p_{m+1} as a linear combination of the current residual r_m and all previous p_i's. The approximate solution is updated by using (6.84). This is called the Generalized Conjugate Residual (GCR) algorithm.

ALGORITHM 6.20: GCR

1. *Compute* $r_0 = b - Ax_0$. *Set* $p_0 = r_0$.
2. *For* $j = 0, 1, 2, \ldots,$ *until convergence Do:*
3. $\quad \alpha_j = \frac{(r_j, Ap_j)}{(Ap_j, Ap_j)}$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j A p_j$
6. $\quad$ *Compute* $\beta_{ij} = -\frac{(Ar_{j+1}, Ap_i)}{(Ap_i, Ap_i)}$, *for* $i = 0, 1, \ldots, j$
7. $\quad p_{j+1} = r_{j+1} + \sum_{i=0}^{j} \beta_{ij} p_i$
8. *EndDo*

To compute the scalars β_{ij} in the above algorithm, the vector Ar_j and the previous Ap_i's are required. In order to limit the number of matrix-vector products per step to one, we can proceed as follows. Follow line 5 by a computation of Ar_{j+1} and then compute Ap_{j+1} after line 7 from the relation

$$Ap_{j+1} = Ar_{j+1} + \sum_{i=0}^{j} \beta_{ij} Ap_i .$$

Both the set of p_i's and that of the Ap_i's need to be saved. This doubles the storage requirement compared with GMRES. The number of arithmetic operations per step is also roughly 50% higher than GMRES.

The above version of GCR suffers from the same practical limitations as GMRES and FOM. A restarted version called GCR(m) can be trivially defined. Also, a truncation of the orthogonalization of the Ap_i's, similar to IOM, leads to an algorithm known as ORTHOMIN(k). Specifically, lines 6 and 7 of Algorithm 6.20 are replaced by

6a. Compute $\beta_{ij} = -\frac{(Ar_{j+1}, Ap_i)}{(Ap_i, Ap_i)}$, for $i = j-k+1, \ldots, j$

7a. $p_{j+1} = r_{j+1} + \sum_{i=j-k+1}^{j} \beta_{ij} p_i$.

Another class of algorithms is defined by computing the next basis vector p_{j+1} as

$$p_{j+1} = Ap_j + \sum_{i=0}^{j} \beta_{ij} p_i \tag{6.86}$$

in which, as before, the β_{ij}'s are selected to make the Ap_i's orthogonal, i.e.,

$$\beta_{ij} = -\frac{(A^2 p_j, Ap_i)}{(Ap_i, Ap_i)} .$$

The resulting algorithm is called ORTHODIR [127]. Restarted and truncated versions of ORTHODIR can also be defined.

6.10 THE FABER-MANTEUFFEL THEOREM

As was seen in Section 6.6 when A is symmetric, the Arnoldi algorithm simplifies into the Lanczos procedure, which is defined through a three-term recurrence. As a consequence, FOM is mathematically equivalent to the Conjugate Gradient algorithm in this case. Similarly, the full GMRES algorithm gives rise to the Conjugate Residual algorithm. It is clear that the CG-type algorithms, i.e., algorithms defined through short-term recurrences, are more desirable than those algorithms which require storing entire sequences of vectors as in the GMRES process. These algorithms require less memory and operations per step.

Therefore, the question is: *Is it possible to define algorithms which are based on optimal Krylov subspace projection and which give rise to sequences involving short-term recurrences?* An optimal Krylov subspace projection means a technique which minimizes a certain norm of the error, or residual, on the Krylov subspace. Such methods can be defined from the Arnoldi process. If the Arnoldi process simplifies into an s-term recurrence, i.e., if $h_{ij} = 0$ for $i < j - s + 1$, then the conjugate directions p_i in DIOM are also defined from an s-term recurrence. Similarly, the full GMRES would also simplify into a DQGMRES algorithm involving a short recurrence. Therefore, for all purposes, it is sufficient to analyze what happens to the Arnoldi process (or FOM). We start by generalizing the CG result in a simple way, by considering the DIOM algorithm.

PROPOSITION 6.14 *Let A be a matrix such that*

$$A^T v \in \mathcal{K}_s(A, v)$$

for any vector v. Then, DIOM(s) is mathematically equivalent to the FOM algorithm.

Proof. The assumption is equivalent to the statement that, for any v, there is a polynomial q_v of degree $\leq s - 1$, such that $A^T v = q_v(A)v$. In the Arnoldi process, the scalars h_{ij} are defined by $h_{ij} = (Av_j, v_i)$ and therefore

$$h_{ij} = (Av_j, v_i) = (v_j, A^T v_i) = (v_j, q_{v_j}(A)v_i). \tag{6.87}$$

Since q_{v_j} is a polynomial of degree $\leq s - 1$, the vector $q_{v_j}(A)v_i$ is a linear combination of the vectors $v_i, v_{i+1}, \ldots, v_{i+s-1}$. As a result, if $i < j - s + 1$, then $h_{ij} = 0$. Therefore, DIOM(k) will give the same approximate solution as FOM. ∎

In particular, if

$$A^T = q(A)$$

where q is a polynomial of degree $\leq s - 1$, then the result holds. However, the above relation implies that each eigenvector of A is also an eigenvector of A^T. According to Theorem 1.2, this can be true only if A is a normal matrix. As it turns out, the reverse is also true. That is, when A is normal, then there is a polynomial of degree $\leq n - 1$ such that $A^H = q(A)$. Proving this is easy because when $A = Q\Lambda Q^H$ where Q is unitary and Λ diagonal, then $q(A) = Qq(\Lambda)Q^H$. By choosing the polynomial q so that

$$q(\lambda_j) = \bar{\lambda}_j, \; j = 1, \ldots, n$$

we obtain $q(A) = Q\bar{\Lambda}Q^H = A^H$ which is the desired result.

Let $\nu(A)$ be the smallest degree of all polynomials q such that $A^H = q(A)$. Then the following lemma due to Faber and Manteuffel [85] states an interesting relation between s and $\nu(A)$.

LEMMA 6.3 *A nonsingular matrix A is such that*

$$A^H v \in \mathcal{K}_s(A, v)$$

for every vector v if and only if A is normal and $\nu(A) \leq s - 1$.

Proof. The sufficient condition is trivially true. To prove the necessary condition, assume that, for any vector v, $A^H v = q_v(A)v$ where q_v is a polynomial of degree $\leq s - 1$. Then it is easily seen that any eigenvector of A is also an eigenvector of A^H. Therefore, from Theorem 1.2, A is normal. Let μ be the degree of the minimal polynomial for A. Then, since A has μ distinct eigenvalues, there is a polynomial q of degree $\mu - 1$ such that

$q(\lambda_i) = \bar{\lambda}_i$ for $i = 1, \ldots, \mu$. According to the above argument, for this q, it holds $A^H = q(A)$ and therefore $\nu(A) \leq \mu - 1$. Now it must be shown that $\mu \leq s$. Let w be a (nonzero) vector whose grade is μ. By assumption, $A^H w \in \mathcal{K}_s(A, w)$. On the other hand, we also have $A^H w = q(A)w$. Since the vectors $w, Aw, \ldots, A^{\mu-1}w$ are linearly independent, $\mu - 1$ must not exceed $s - 1$. Otherwise, two different expressions for $A^H w$ with respect to the basis $w, Aw, \ldots, A^{\mu-1}w$ would result and this would imply that $A^H w = 0$. Since A is nonsingular, then $w = 0$, which is a contradiction. ■

Proposition 6.14 gives a sufficient condition for DIOM(s) to be equivalent to FOM. According to Lemma 6.3, this condition is equivalent to A being normal and $\nu(A) \leq s - 1$. Now consider the reverse result. Faber and Manteuffel define CG(s) to be the class of all matrices such that *for every* v_1, *it is true that* $(Av_j, v_i) = 0$ *for all* i, j *such that* $i + s \leq j \leq \mu(v_1) - 1$. The inner product can be different from the canonical Euclidean dot product. With this definition it is possible to show the following theorem [85] which is stated without proof.

THEOREM 6.3 $A \in CG(s)$, *if and only if* $\mu(A) \leq s$ *or* A *is normal and* $\nu(A) \leq s-1$.

It is interesting to consider the particular case where $\nu(A) \leq 1$, which is the case of the Conjugate Gradient method. In fact, it is easy to show that in this case A either has a minimal degree ≤ 1, or is Hermitian, or is of the form

$$A = e^{i\theta} (\rho I + B)$$

where θ and ρ are real and B is skew-Hermitian, i.e., $B^H = -B$. Thus, the cases in which DIOM simplifies into an (optimal) algorithm defined from a three-term recurrence are already known. The first is the Conjugate Gradient method. The second is a version of the CG algorithm for skew-Hermitian matrices which can be derived from the Lanczos algorithm in the same way as CG. This algorithm will be seen in Chapter 9.

6.11 CONVERGENCE ANALYSIS

The convergence behavior of the different algorithms seen in this chapter can be analyzed by exploiting optimality properties whenever such properties exist. This is the case for the Conjugate Gradient and the GMRES algorithms. On the other hand, the non-optimal algorithms such as FOM, IOM, and QGMRES will be harder to analyze.

One of the main tools used in the analysis of these methods is Chebyshev polynomials. These polynomials are useful both in theory, when studying convergence, and in practice, as a means of accelerating single-vector iterations or projection processes. In the following, real and complex Chebyshev polynomials are discussed separately.

6.11.1 REAL CHEBYSHEV POLYNOMIALS

The Chebyshev polynomial of the first kind of degree k is defined by

$$C_k(t) = \cos[k\, \cos^{-1}(t)] \quad \text{for} \quad -1 \le t \le 1. \tag{6.88}$$

That this is a polynomial with respect to t can be shown easily by induction from the trigonometric relation

$$\cos[(k+1)\theta] + \cos[(k-1)\theta] = 2\cos\theta \cos k\theta,$$

and the fact that $C_1(t) = t, C_0(t) = 1$. Incidentally, this also shows the important three-term recurrence relation

$$C_{k+1}(t) = 2\,t\,C_k(t) - C_{k-1}(t).$$

The definition (6.88) can be extended to cases where $|t| > 1$ with the help of the following formula:

$$C_k(t) = \cosh\,[k \cosh^{-1}(t)], \quad |t| \ge 1. \tag{6.89}$$

This is readily seen by passing to complex variables and using the definition $\cos\theta = (e^{i\theta} + e^{-i\theta})/2$. As a result of (6.89) the following expression can be derived:

$$C_k(t) = \frac{1}{2}\left[\left(t + \sqrt{t^2-1}\right)^k + \left(t + \sqrt{t^2-1}\right)^{-k}\right], \tag{6.90}$$

which is valid for $|t| \ge 1$ but can also be extended to the case of $|t| < 1$. The following approximation, valid for large values of k, will be sometimes used:

$$C_k(t) \gtrapprox \frac{1}{2}\left(t + \sqrt{t^2-1}\right)^k \quad \text{for} \quad |t| \ge 1. \tag{6.91}$$

In what follows we denote by $\mathbb{P}_k$ the set of all polynomials of degree k. An important result from approximation theory is the following theorem.

THEOREM 6.4 *Let $[\alpha, \beta]$ be a non-empty interval in $\mathbb{R}$ and let γ be any real scalar outside the interval $[\alpha, \beta]$. Then the minimum*

$$\min_{p\in\mathbb{P}_k, p(\gamma)=1} \; \max_{t\in[\alpha,\beta]} |p(t)|$$

is reached by the polynomial

$$\hat{C}_k(t) \equiv \frac{C_k\left(1 + 2\frac{t-\beta}{\beta-\alpha}\right)}{C_k\left(1 + 2\frac{\gamma-\beta}{\beta-\alpha}\right)}. \tag{6.92}$$

For a proof, see Cheney [52]. The maximum of C_k for t in $[-1, 1]$ is 1 and a corollary of the above result is

$$\min_{p\in\mathbb{P}_k,\ p(\gamma)=1} \; \max_{t\in[\alpha,\beta]} |p(t)| = \frac{1}{|C_k(1 + 2\frac{\gamma-\beta}{\beta-\alpha})|} = \frac{1}{|C_k(2\frac{\gamma-\mu}{\beta-\alpha})|}$$

in which $\mu \equiv (\alpha + \beta)/2$ is the middle of the interval. The absolute values in the denominator are needed only when γ is to the left of the interval, i.e., when $\gamma \le \alpha$. For this case, it may be more convenient to express the best polynomial as

$$\hat{C}_k(t) \equiv \frac{C_k\left(1 + 2\frac{\alpha - t}{\beta - \alpha}\right)}{C_k\left(1 + 2\frac{\alpha - \gamma}{\beta - \alpha}\right)}.$$

which is obtained by exchanging the roles of α and β in (6.92).

6.11.2 COMPLEX CHEBYSHEV POLYNOMIALS

The standard definition of real Chebyshev polynomials given by equation (6.88) extends without difficulty to complex variables. First, as was seen before, when t is real and $|t| > 1$, the alternative definition, $C_k(t) = \cosh[k \cosh^{-1}(t)]$, can be used. These definitions can be unified by switching to complex variables and writing

$$C_k(z) = \cosh(k\zeta), \quad \text{where} \quad \cosh(\zeta) = z\ .$$

Defining the variable $w = e^{\zeta}$, the above formula is equivalent to

$$C_k(z) = \frac{1}{2}[w^k + w^{-k}] \quad \text{where} \quad z = \frac{1}{2}[w + w^{-1}]. \tag{6.93}$$

The above definition for Chebyshev polynomials will be used in $\mathbb{C}$. Note that the equation $\frac{1}{2}(w + w^{-1}) = z$ has two solutions w which are inverse of each other. As a result, the value of $C_k(z)$ does not depend on which of these solutions is chosen. It can be verified directly that the C_k's defined by the above equations are indeed polynomials in the z variable and that they satisfy the three-term recurrence

$$\begin{aligned} &C_{k+1}(z) = 2\,zC_k(z) - C_{k-1}(z), \\ &C_0(z) \equiv 1, \qquad C_1(z) \equiv z. \end{aligned} \tag{6.94}$$

As is now explained, Chebyshev polynomials are intimately related to ellipses in the complex plane. Let C_ρ be the circle of radius ρ centered at the origin. Then the so-called Joukowski mapping

$$J(w) = \frac{1}{2}[w + w^{-1}]$$

transforms C_ρ into an ellipse of center the origin, foci $-1, 1$, major semi-axis $\frac{1}{2}[\rho + \rho^{-1}]$ and minor semi-axis $\frac{1}{2}|\rho - \rho^{-1}|$. This is illustrated in Figure 6.2.

There are two circles which have the same image by the mapping $J(w)$, one with the radius ρ and the other with the radius ρ^{-1}. So it is sufficient to consider only those circles with radius $\rho \ge 1$. Note that the case $\rho = 1$ is a degenerate case in which the ellipse $E(0, 1, -1)$ reduces to the interval $[-1, 1]$ traveled through twice.

An important question is whether or not a generalization of the min-max result of Theorem 6.4 holds for the complex case. Here, the maximum of $|p(z)|$ is taken over the ellipse boundary and γ is some point not enclosed by the ellipse. The answer to the question is no; Chebyshev polynomials are only optimal in some cases. However, Chebyshev polynomials are asymptotically optimal, which is all that is needed in practice.

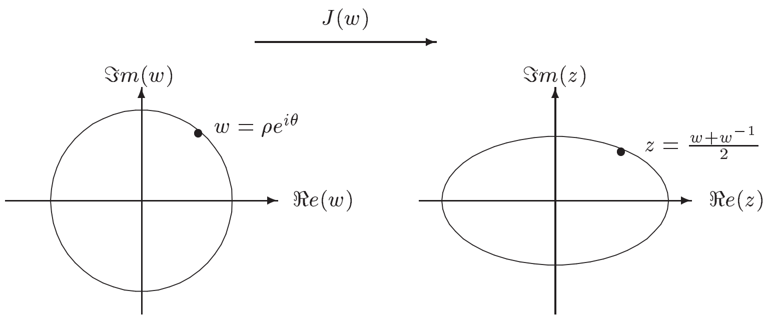


Figure 6.2 *The Joukowski mapping transforms a circle into an ellipse in the complex plane.*

To prove the asymptotic optimality, we begin with a lemma due to Zarantonello, which deals with the particular case where the ellipse reduces to a circle. This particular case is important in itself.

LEMMA 6.4 Zarantonello Let $C(0, \rho)$ be a circle of center the origin and radius ρ and let γ be a point of $\mathbb{C}$ not enclosed by $C(0, \rho)$. Then

$$\min_{p \in \mathbb{P}_k, \ p(\gamma)=1} \quad \max_{z \, \in \, C(0,\rho)} |p(z)| \ = \ \left(\frac{\rho}{|\gamma|}\right)^k, \tag{6.95}$$

the minimum being achieved for the polynomial $(z/\gamma)^k$.

Proof. See reference [168] for a proof. ∎

Note that by changing variables, shifting, and rescaling the polynomial, then for any circle centered at c and for any scalar γ such that $|\gamma| > \rho$, the following min-max result holds:

$$\min_{p \in \mathbb{P}_k \ \ p(\gamma)=1} \quad \max_{z \, \in \, C(c,\rho)} |p(z)| \ = \ \left(\frac{\rho}{|\gamma - c|}\right)^k.$$

Now consider the case of an ellipse centered at the origin, with foci $1, -1$ and semi-major axis a, which can be considered as mapped by J from the circle $C(0, \rho)$, with the convention that $\rho \geq 1$. Denote by E_ρ such an ellipse.

THEOREM 6.5 *Consider the ellipse E_ρ mapped from $C(0, \rho)$ by the mapping J and let γ be any point in the complex plane not enclosed by it. Then*

$$\frac{\rho^k}{|w_\gamma|^k} \leq \min_{p \in \mathbb{P}_k \, p(\gamma)=1} \quad \max_{z \, \in \, E_\rho} |p(z)| \leq \frac{\rho^k + \rho^{-k}}{|w_\gamma^k + w_\gamma^{-k}|} \tag{6.96}$$

in which w_γ is the dominant root of the equation $J(w) = \gamma$.

Proof. We start by showing the second inequality. Any polynomial p of degree k satisfying the constraint $p(\gamma) = 1$ can be written as

$$p(z) = \frac{\sum_{j=0}^{k} \xi_j z^j}{\sum_{j=0}^{k} \xi_j \gamma^j}.$$

A point z on the ellipse is transformed by J from a certain w in $C(0, \rho)$. Similarly, let w_γ be one of the two inverse transforms of γ by the mapping, namely, the one with largest modulus. Then, p can be rewritten as

$$p(z) = \frac{\sum_{j=0}^{k} \xi_j (w^j + w^{-j})}{\sum_{j=0}^{k} \xi_j (w_\gamma^j + w_\gamma^{-j})}. \tag{6.97}$$

Consider the particular polynomial obtained by setting $\xi_k = 1$ and $\xi_j = 0$ for $j \neq k$,

$$p^*(z) = \frac{w^k + w^{-k}}{w_\gamma^k + w_\gamma^{-k}}$$

which is a scaled Chebyshev polynomial of the first kind of degree k in the variable z. It is apparent that the maximum modulus of this polynomial is reached in particular when $w = \rho e^{i\theta}$ is real, i.e., when $w = \rho$. Thus,

$$\max_{z \in E_\rho} |p^*(z)| = \frac{\rho^k + \rho^{-k}}{|w_\gamma^k + w_\gamma^{-k}|}$$

which proves the second inequality.

To prove the left inequality, we rewrite (6.97) as

$$p(z) = \left(\frac{w^{-k}}{w_\gamma^{-k}}\right) \frac{\sum_{j=0}^{k} \xi_j (w^{k+j} + w^{k-j})}{\sum_{j=0}^{k} \xi_j (w_\gamma^{k+j} + w_\gamma^{k-j})}$$

and take the modulus of $p(z)$,

$$|p(z)| = \frac{\rho^{-k}}{|w_\gamma|^{-k}} \left| \frac{\sum_{j=0}^{k} \xi_j (w^{k+j} + w^{k-j})}{\sum_{j=0}^{k} \xi_j (w_\gamma^{k+j} + w_\gamma^{k-j})} \right|.$$

The polynomial in w of degree $2k$ inside the large modulus bars in the right-hand side is such that its value at w_γ is one. By Lemma 6.4, the modulus of this polynomial over the circle $C(0, \rho)$ is not less than $(\rho/|w_\gamma|)^{2k}$, i.e., for any polynomial, satisfying the constraint $p(\gamma) = 1$,

$$\max_{z \in E_\rho} |p(z)| \geq \frac{\rho^{-k}}{|w_\gamma|^{-k}} \frac{\rho^{2k}}{|w_\gamma|^{2k}} = \frac{\rho^k}{|w_\gamma|^k}.$$

This proves that the minimum over all such polynomials of the maximum modulus on the ellipse E_ρ is $\geq (\rho/|w_\gamma|)^k$. ∎

The difference between the left and right bounds in (6.96) tends to zero as k increases to infinity. Thus, the important point made by the theorem is that for large k, the Chebyshev polynomial

$$p^*(z) = \frac{w^k + w^{-k}}{w_\gamma^k + w_\gamma^{-k}}, \quad \text{where} \quad z = \frac{w + w^{-1}}{2}$$

is close to the optimal polynomial. More specifically, Chebyshev polynomials are *asymptotically* optimal.

For a more general ellipse $E(c, d, a)$ centered at c, and with focal distance d and semi-major axis a, a simple change of variables shows that the near-best polynomial is given by

$$\hat{C}_k(z) = \frac{C_k\left(\frac{c-z}{d}\right)}{C_k\left(\frac{c-\gamma}{d}\right)}. \tag{6.98}$$

In addition, by examining the expression $(w^k + w^{-k})/2$ for $w = \rho e^{i\theta}$ it is easily seen that the maximum modulus of $\hat{C}_k(z)$, i.e., the infinity norm of this polynomial over the ellipse, is reached at the point $c + a$ located on the real axis. From this we get,

$$\max_{z \in E(c,d,a)} |\hat{C}_k(z)| = \frac{C_k\left(\frac{a}{d}\right)}{|C_k\left(\frac{c-\gamma}{d}\right)|}$$

Here, we point out that d and a both can be purely imaginary [for an example, see part (B) of Figure 6.3]. In this case a/d is real and the numerator in the above expression is always real. Using the definition for C_k we obtain the following useful expression and approximation:

$$\frac{C_k\left(\frac{a}{d}\right)}{C_k\left(\frac{c-\gamma}{d}\right)} = \frac{\left(\frac{a}{d} + \sqrt{\left(\frac{a}{d}\right)^2 - 1}\right)^k + \left(\frac{a}{d} + \sqrt{\left(\frac{a}{d}\right)^2 - 1}\right)^{-k}}{\left(\frac{c-\gamma}{d} + \sqrt{\left(\frac{c-\gamma}{d}\right)^2 - 1}\right)^k + \left(\frac{c-\gamma}{d} + \sqrt{\left(\frac{c-\gamma}{d}\right)^2 - 1}\right)^{-k}} \tag{6.99}$$

$$\approx \left(\frac{a + \sqrt{a^2 - d^2}}{c - \gamma + \sqrt{(c-\gamma)^2 - d^2}}\right)^k \tag{6.100}$$

Finally, we note that an alternative and more detailed result has been proven by Fischer and Freund in [89].

6.11.3 CONVERGENCE OF THE CG ALGORITHM

As usual, $\|x\|_A$ denotes the norm defined by

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

The following lemma characterizes the approximation obtained from the Conjugate Gradient algorithm.

LEMMA 6.5 *Let x_m be the approximate solution obtained from the m-th step of the CG algorithm, and let $d_m = x_* - x_m$ where x_* is the exact solution. Then, x_m is of the form*

$$x_m = x_0 + q_m(A)r_0$$

where q_m is a polynomial of degree $m - 1$ such that

$$\|(I - Aq_m(A))d_0\|_A = \min_{q \in \mathbb{P}_{m-1}} \|(I - Aq(A))d_0\|_A.$$

Proof. This is a consequence of the fact that x_m minimizes the A-norm of the error in the affine subspace $x_0 + \mathcal{K}_m$, a result of Proposition 5.2, and the fact that $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m-1$. ∎

From this, the following theorem can be proved.

THEOREM 6.6 *Let x_m be the approximate solution obtained at the m-th step of the Conjugate Gradient algorithm, and x_* the exact solution and define*

$$\eta = \frac{\lambda_{min}}{\lambda_{max} - \lambda_{min}}. \tag{6.101}$$

Then,

$$\|x_* - x_m\|_A \leq \frac{\|x_* - x_0\|_A}{C_m(1+2\eta)}, \tag{6.102}$$

in which C_m is the Chebyshev polynomial of degree m of the first kind.

Proof. From the previous lemma, it is known that $\|x_* - x_m\|_A$ minimizes A-norm of the error over polynomials $r(t)$ which take the value one at 0, i.e.,

$$\|x_* - x_m\|_A = \min_{r \in \mathbb{P}_m,\ r(0)=1} \|r(A)d_0\|_A.$$

If $\lambda_i, i = 1, \ldots, n$ are the eigenvalues of A, and $\xi_i, i = 1, \ldots, n$ the components of the initial error d_0 in the eigenbasis, then

$$\begin{aligned}\|r(A)d_0\|_A^2 = \sum_{i=1}^{n} \lambda_i r(\lambda_i)^2(\xi_i)^2 &\leq \max_i (r(\lambda_i))^2 \|d_0\|_A^2 \\ &\leq \max_{\lambda \in [\lambda_{min}, \lambda_{max}]} (r(\lambda))^2 \|d_0\|_A^2.\end{aligned}$$

Therefore,

$$\|x_* - x_m\|_A \leq \min_{r \in \mathbb{P}_m,\ r(0)=1} \ \max_{\lambda \in [\lambda_{min}, \lambda_{max}]} |r(\lambda)| \|d_0\|_A.$$

The result follows immediately by using the well known result of Theorem 6.4 from approximation theory. This gives the polynomial r which minimizes the right-hand side. ∎

A slightly different formulation of inequality (6.102) can be derived. Using the relation,

$$\begin{aligned}C_m(t) &= \frac{1}{2}\left[\left(t + \sqrt{t^2-1}\right)^m + \left(t + \sqrt{t^2-1}\right)^{-m}\right] \\ &\geq \frac{1}{2}\left(t + \sqrt{t^2-1}\right)^m\end{aligned}$$

then

$$\begin{aligned}C_m(1+2\eta) &\geq \frac{1}{2}\left(1 + 2\eta + \sqrt{(1+2\eta)^2 - 1}\right)^m \\ &\geq \frac{1}{2}\left(1 + 2\eta + 2\sqrt{\eta(\eta+1)}\right)^m.\end{aligned}$$

Now notice that

$$1 + 2\eta + 2\sqrt{\eta(\eta+1)} = \left(\sqrt{\eta} + \sqrt{\eta+1}\right)^2 \tag{6.103}$$

$$= \frac{\left(\sqrt{\lambda_{min}} + \sqrt{\lambda_{max}}\right)^2}{\lambda_{max} - \lambda_{min}} \tag{6.104}$$

$$= \frac{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}} \tag{6.105}$$

$$= \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \tag{6.106}$$

in which κ is the spectral condition number $\kappa = \lambda_{max}/\lambda_{min}$.

Substituting this in (6.102) yields,

$$\|x_* - x_m\|_A \leq 2\left[\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right]^m \|x_* - x_0\|_A. \tag{6.107}$$

This bound is similar to that of the steepest descent algorithm except that the condition number of A is now replaced by its square root.

6.11.4 CONVERGENCE OF GMRES

We begin by stating a *global* convergence result. Recall that a matrix A is called positive definite if its symmetric part $(A+A^T)/2$ is Symmetric Positive Definite. This is equivalent to the property that $(Ax, x) > 0$ for all nonzero real vectors x.

THEOREM 6.7 *If A is a positive definite matrix, then GMRES(m) converges for any $m \geq 1$.*

Proof. This is true because the subspace $\mathcal{K}_m$ contains the initial residual vector at each restart. Since the algorithm minimizes the residual norm in the subspace $\mathcal{K}_m$, at each outer iteration, the residual norm will be reduced by as much as the result of one step of the Minimal Residual method seen in the previous chapter. Therefore, the inequality (5.18) is satisfied by residual vectors produced after each outer iteration and the method converges. ■

Next we wish to establish a result similar to the one for the Conjugate Gradient method, which would provide an upper bound on the convergence rate of the GMRES iterates. We begin with a lemma similar to Lemma 6.5.

LEMMA 6.6 *Let x_m be the approximate solution obtained from the m-th step of the GMRES algorithm, and let $r_m = b - Ax_m$. Then, x_m is of the form*

$$x_m = x_0 + q_m(A)r_0$$

and

$$\|r_m\|_2 = \|(I - Aq_m(A))r_0\|_2 = \min_{q \,\in\, \mathbb{P}_{m-1}} \|(I - Aq(A))r_0\|_2.$$

Proof. This is true because x_m minimizes the 2-norm of the residual in the affine subspace $x_0 + \mathcal{K}_m$, a result of Proposition 5.3, and the fact that $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m-1$. ∎

Unfortunately, it not possible to prove a simple result such as Theorem 6.6 unless A is normal.

PROPOSITION 6.15 *Assume that A is a diagonalizable matrix and let $A = X\Lambda X^{-1}$ where $\Lambda = \text{diag}\ \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ is the diagonal matrix of eigenvalues. Define,*

$$\epsilon^{(m)} = \min_{p \in \mathbb{P}_m, p(0)=1} \max_{i=1,\ldots,n} |p(\lambda_i)|.$$

Then, the residual norm achieved by the m-th step of GMRES satisfies the inequality

$$\|r_m\|_2 \leq \kappa_2(X)\epsilon^{(m)}\|r_0\|_2.$$

where $\kappa_2(X) \equiv \|X\|_2\|X^{-1}\|_2$.

Proof. Let p be any polynomial of degree $\leq m$ which satisfies the constraint $p(0) = 1$, and x the vector in $\mathcal{K}_m$ to which it is associated via $b - Ax = p(A)r_0$. Then,

$$\|b - Ax\|_2 = \|Xp(\Lambda)X^{-1}r_0\|_2 \leq \|X\|_2\|X^{-1}\|_2\|r_0\|_2\|p(\Lambda)\|_2$$

Since Λ is diagonal, observe that

$$\|p(\Lambda)\|_2 = \max_{i=1,\ldots,n} |p(\lambda_i)|.$$

Since x_m minimizes the residual norm over $x_0 + \mathcal{K}_m$, then for any consistent polynomial p,

$$\|b - Ax_m\| \leq \|b - Ax\|_2 \leq \|X\|_2\|X^{-1}\|_2\|r_0\|_2 \max_{i=1,\ldots,n} |p(\lambda_i)|.$$

Now the polynomial p which minimizes the right-hand side in the above inequality can be used. This yields the desired result,

$$\|b - Ax_m\| \leq \|b - Ax\|_2 \leq \|X\|_2\|X^{-1}\|_2\|r_0\|_2\epsilon^{(m)}.$$

∎

The results of Section 6.11.2 on near-optimal Chebyshev polynomials in the complex plane can now be used to obtain an upper bound for $\epsilon^{(m)}$. Assume that the spectrum of A in contained in an ellipse $E(c, d, a)$ with center c, focal distance d, and major semi axis a. In addition it is required that the origin lie outside this ellipse. The two possible cases are shown in Figure 6.3. Case (B) corresponds to the situation when d is purely imaginary, i.e., the major semi-axis is aligned with the imaginary axis.

COROLLARY 6.1 *Let A be a diagonalizable matrix, i.e, let $A = X\Lambda X^{-1}$ where $\Lambda = \text{diag}\ \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ is the diagonal matrix of eigenvalues. Assume that all the eigenvalues of A are located in the ellipse $E(c, d, a)$ which excludes the origin. Then, the* *residual norm achieved at the m-th step of GMRES satisfies the inequality,*

$$\|r_m\|_2 \le \kappa_2(X)\, \frac{C_m\left(\frac{a}{d}\right)}{\left|C_m\left(\frac{c}{d}\right)\right|}\, \|r_0\|_2.$$

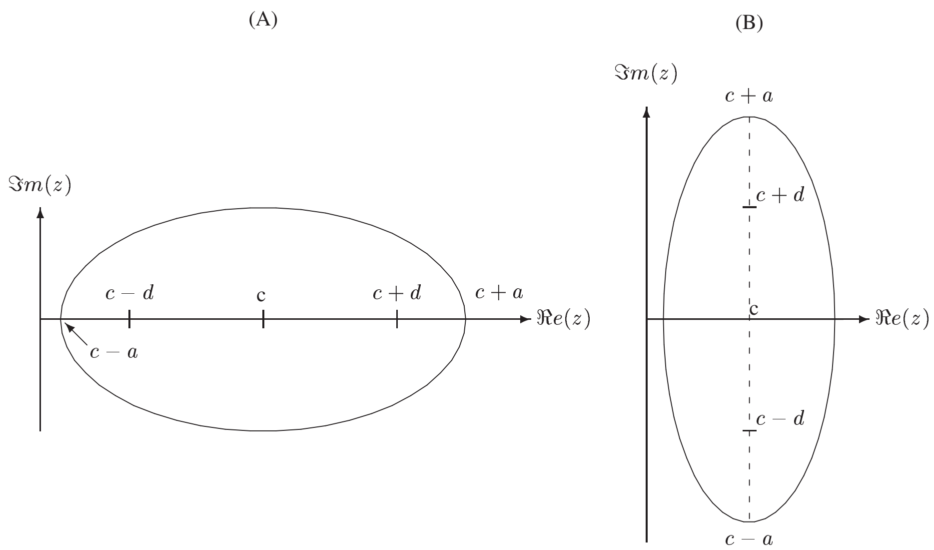


Figure 6.3 *Ellipses containing the spectrum of A. Case (A): real d; case (B): purely imaginary d.*

Proof. All that is needed is an upper bound for the scalar $\epsilon^{(m)}$ under the assumptions. By definition,

$$\begin{aligned}\epsilon^{(m)} &= \min_{p\in\mathbb{P}_m, p(0)=1} \ \max_{i=1,\dots,n} |p(\lambda_i)| \\ &\le \min_{p\in\mathbb{P}_m, p(0)=1} \ \max_{\lambda\,\in\, E(c,d,a)} |p(\lambda)|.\end{aligned}$$

The second inequality is due to the fact that the maximum modulus of a complex analytical function is reached on the boundary of the domain. We can now use as a trial polynomial $\hat{C}_m$ defined by (6.98), with $\gamma = 0$:

$$\begin{aligned}\epsilon^{(m)} &\le \min_{p\in\mathbb{P}_m, p(0)=1} \ \max_{\lambda\,\in\, E(c,d,a)} |p(\lambda)| \\ &\le \max_{\lambda\,\in\, E(c,d,a)} |\hat{C}_m(\lambda)| = \frac{C_m\left(\frac{a}{d}\right)}{\left|C_m\left(\frac{c}{d}\right)\right|}.\end{aligned}$$

This completes the proof. ■

An explicit expression for the coefficient $C_m\left(\frac{a}{d}\right) \;/\; C_m\left(\frac{c}{d}\right)$ and an approximation are readily obtained from (6.99-6.100) by taking $\gamma = 0$:

$$\frac{C_m\left(\frac{a}{d}\right)}{C_m\left(\frac{c}{d}\right)} = \frac{\left(\frac{a}{d} + \sqrt{\left(\frac{a}{d}\right)^2 - 1}\right)^k + \left(\frac{a}{d} + \sqrt{\left(\frac{a}{d}\right)^2 - 1}\right)^{-k}}{\left(\frac{c}{d} + \sqrt{\left(\frac{c}{d}\right)^2 - 1}\right)^k + \left(\frac{c}{d} + \sqrt{\left(\frac{c}{d}\right)^2 - 1}\right)^{-k}}$$
$$\approx \left(\frac{a + \sqrt{a^2 - d^2}}{c + \sqrt{c^2 - d^2}}\right)^k .$$

Since the condition number $\kappa_2(X)$ of the matrix of eigenvectors X is typically not known and can be very large, results of the nature of the corollary are of limited practical interest. They can be useful only when it is known that the matrix is nearly normal, in which case, $\kappa_2(X) \approx 1$.

6.12 BLOCK KRYLOV METHODS

In many circumstances, it is desirable to work with a block of vectors instead of a single vector. For example, out-of-core finite-element codes are more efficient when they are programmed to exploit the presence of a block of the matrix A in fast memory, as much as possible. This can be achieved by using block generalizations of Krylov subspace methods, for which A always operates on a group of vectors instead of a single vector. We begin by describing a block version of the Arnoldi algorithm.

ALGORITHM 6.21: Block Arnoldi

1. *Choose a unitary matrix* V_1 *of dimension* $n \times p$.
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. *Compute* $H_{ij} = V_i^T A V_j \quad i = 1, 2, \ldots, j$
4. *Compute* $W_j = AV_j - \sum_{i=1}^{j} V_i H_{ij}$
5. *Compute the Q-R factorization of* W_j: $W_j = V_{j+1} H_{j+1,j}$
6. *EndDo*

The above algorithm is a straightforward block analogue of Algorithm 6.1. By construction, the blocks generated by the algorithm are orthogonal blocks that are also orthogonal to each other. In the following we denote by I_k the $k \times k$ identity matrix and use the following notation:

$$U_m = [V_1, V_2, \ldots, V_m],$$
$$H_m = (H_{ij})_{1 \le i,j \le m}, \quad H_{ij} \equiv 0, \quad \text{for} \quad i > j + 1,$$
$$E_m = \text{matrix of the last } p \text{ columns of } I_n.$$

Then, the following analogue of the relation (6.4) is easily proved:

$$AU_m = U_m H_m + V_{m+1} H_{m+1,m} E_m^T. \tag{6.108}$$

Here, the matrix H_m is no longer Hessenberg, but band-Hessenberg, meaning that it has p subdiagonals instead of only one. Note that the dimension of the subspace in which the solution is sought is not m but $m.p$.

A second version of the algorithm uses a modified block Gram-Schmidt procedure instead of the simple Gram-Schmidt procedure used above. This leads to a block generalization of Algorithm 6.2, the Modified Gram-Schmidt version of Arnoldi's method.

ALGORITHM 6.22: Block Arnoldi with Block MGS

1. *Choose a unitary matrix* V_1 *of size* $n \times p$
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. *Compute* $W_j := AV_j$
4. *For* $i = 1, 2, \ldots, j$ *do:*
5. $H_{ij} := V_i^T W_j$
6. $W_j := W_j - V_i H_{ij}$
7. *EndDo*
8. *Compute the Q-R decomposition* $W_j = V_{j+1} H_{j+1,j}$
9. *EndDo*

Again, in practice the above algorithm is more viable than its predecessor. Finally, a third version, developed by A. Ruhe [170] for the symmetric case (block Lanczos), yields a variant that is quite similar to the original Arnoldi algorithm. Assume that the initial block of p orthonormal vectors, $v_1, \ldots, v_p$ is available. The first step of the algorithm is to multiply v_1 by A and orthonormalize the resulting vector w against $v_1, \ldots, v_p$. The resulting vector is defined to be v_{p+1}. In the second step it is v_2 that is multiplied by A and orthonormalized against all available v_i's. Thus, the algorithm works similarly to Algorithm 6.2 except for a delay in the vector that is multiplied by A at each step.

ALGORITHM 6.23: Block Arnoldi–Ruhe's variant

1. *Choose* p *initial orthonormal vectors* $\{v_i\}_{i=1,\ldots,p}$.
2. *For* $j = p, p+1, \ldots, m$ *Do:*
3. *Set* $k := j - p + 1$;
4. *Compute* $w := Av_k$;
5. *For* $i = 1, 2, \ldots, j$ *Do:*
6. $h_{i,k} := (w, v_i)$
7. $w := w - h_{i,k} v_i$
8. *EndDo*
9. *Compute* $h_{j+1,k} := \|w\|_2$ *and* $v_{j+1} := w/h_{j+1,k}$.
10. *EndDo*

Observe that the particular case $p = 1$ coincides with the usual Arnoldi process. Also, the dimension m of the subspace of approximants, is no longer restricted to being a multiple of the block-size p as in the previous algorithms. The mathematical equivalence of Algorithms 6.22 and 6.23 when m is a multiple of p is straightforward to show. The advantage of the above formulation is its simplicity. A slight disadvantage is that it gives up some potential parallelism. In the original version, the columns of the matrix AV_j can be computed in parallel whereas in the new algorithm, they are computed in sequence. This can be remedied, however, by performing p matrix-by-vector products every p steps.

At the end of the loop consisting of lines 5 through 8 of Algorithm 6.23, the vector w satisfies the relation

$$w = Av_k - \sum_{i=1}^{j} h_{ik} v_i,$$

where k and j are related by $k = j - p + 1$. Line 9 gives $w = h_{j+1,k} v_{j+1}$ which results in

$$Av_k = \sum_{i=1}^{k+p} h_{ik} v_i.$$

As a consequence, the analogue of the relation (6.5) for Algorithm 6.23 is

$$AV_m = V_{m+p} \bar{H}_m. \tag{6.109}$$

As before, for any j the matrix V_j represents the $n \times j$ matrix with columns $v_1, \ldots v_j$. The matrix $\bar{H}_m$ is now of size $(m + p) \times m$.

Now the block generalizations of FOM and GMRES can be defined in a straightforward way. These block algorithms can solve linear systems with multiple right-hand sides,

$$Ax^{(i)} \;=\; b^{(i)}, \quad i = 1, \cdots, p, \tag{6.110}$$

or, in matrix form

$$AX = B, \tag{6.111}$$

where the columns of the $n \times p$ matrices B and X are the $b^{(i)}$'s and $x^{(i)}$'s, respectively. Given an initial block of initial guesses $x_0^{(i)}$ for $i = 1, \ldots, p$, we define R_0 the block of initial residuals

$$R_0 \equiv [r_0^{(1)}, r_0^{(2)}, \ldots, r_0^{(p)}],$$

where each column is $r_0^{(i)} = b^{(i)} - Ax_0^{(i)}$. It is preferable to use the unified notation derived from Algorithm 6.23. In this notation, m is not restricted to being a multiple of the block-size p and the same notation is used for the v_i's as in the scalar Arnoldi Algorithm. Thus, the first step of the block-FOM or block-GMRES algorithm is to compute the QR factorization of the block of initial residuals:

$$R_0 = [v_1, v_2, \ldots, v_p]\, R.$$

Here, the matrix $[v_1, \ldots, v_p]$ is unitary and R is $p \times p$ upper triangular. This factorization provides the first p vectors of the block-Arnoldi basis.

Each of the approximate solutions has the form

$$x^{(i)} = x_0^{(i)} + V_m y^{(i)}, \tag{6.112}$$

and, grouping these approximations $x^{(i)}$ in a block X and the $y^{(i)}$ in a block Y, we can write

$$X = X_0 + V_m Y. \tag{6.113}$$

It is now possible to imitate what was done for the standard FOM and GMRES algorithms. The only missing link is the vector βe_1 in (6.21) which now becomes a matrix. Let E_1 be the $(m + p) \times p$ matrix whose upper $p \times p$ principal block is an identity matrix. Then, the relation (6.109) results in

$$\begin{aligned} B - AX &= B - A\left(X_0 + V_m Y\right) \\ &= R_0 - AV_m Y \\ &= [v_1, \ldots, v_p]R - V_{m+p}\bar{H}_m Y \\ &= V_{m+p}\left(E_1 R - \bar{H}_m Y\right). \end{aligned} \tag{6.114}$$

The vector

$$\bar{g}^{(i)} \equiv E_1 R e_i$$

is a vector of length $m + p$ whose components are zero except those from 1 to i which are extracted from the i-th column of the upper triangular matrix R. The matrix $\bar{H}_m$ is an $(m + p) \times m$ matrix. The block-FOM approximation would consist of deleting the last p rows of $\bar{g}^{(i)}$ and $\bar{H}_m$ and solving the resulting system,

$$H_m y^{(i)} = g^{(i)}.$$

The approximate solution $x^{(i)}$ is then computed by (6.112).

The block-GMRES approximation $x^{(i)}$ is the unique vector of the form $x_0^{(i)} + V_m y^{(i)}$ which minimizes the 2-norm of the individual columns of the block-residual (6.114). Since the column-vectors of V_{m+p} are orthonormal, then from (6.114) we get,

$$\|b^{(i)} - Ax^{(i)}\|_2 = \|\bar{g}^{(i)} - \bar{H}_m y^{(i)}\|_2. \tag{6.115}$$

To minimize the residual norm, the function on the right hand-side must be minimized over $y^{(i)}$. The resulting least-squares problem is similar to the one encountered for GMRES. The only differences are in the right-hand side and the fact that the matrix is no longer Hessenberg, but band-Hessenberg. Rotations can be used in a way similar to the scalar case. However, p rotations are now needed at each new step instead of only one. Thus, if $m = 6$ and $p = 2$, the matrix $\bar{H}_6$ and block right-hand side would be as follows:

$$\bar{H}_8 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} & h_{26} \\ h_{31} & h_{32} & h_{33} & h_{34} & h_{35} & h_{36} \\ & h_{42} & h_{43} & h_{44} & h_{45} & h_{46} \\ & & h_{53} & h_{54} & h_{55} & h_{56} \\ & & & h_{64} & h_{65} & h_{66} \\ & & & & h_{75} & h_{76} \\ & & & & & h_{86} \end{pmatrix} \qquad \bar{G} = \begin{pmatrix} g_{11} & g_{12} \\ & g_{22} \\ & \\ & \\ & \\ & \\ & \\ & \end{pmatrix}.$$

For each new column generated in the block-Arnoldi process, p rotations are required to eliminate the elements $h_{k,j}$, for $k = j + p$ down to $k = j + 1$. This backward order is important. In the above example, a rotation is applied to eliminate $h_{3,1}$ and then a second rotation is used to eliminate the resulting $h_{2,1}$, and similarly for the second, third step, etc.

This complicates programming slightly since two-dimensional arrays must now be used to save the rotations instead of one-dimensional arrays in the scalar case. After the first column of $\bar{H}_m$ is processed, the block of right-hand sides will have a diagonal added under the diagonal of the upper triangular matrix. Specifically, the above two matrices will have the structure,

$$\bar{H}_8 = \begin{pmatrix} \star & \star & \star & \star & \star & \star \\ & \star & \star & \star & \star & \star \\ & \star & \star & \star & \star & \star \\ & \star & \star & \star & \star & \star \\ & & \star & \star & \star & \star \\ & & & \star & \star & \star \\ & & & & \star & \star \\ & & & & & \star \end{pmatrix} \qquad \bar{G} = \begin{pmatrix} \star & \star \\ \star & \star \\ & \star \\ & \\ & \\ & \\ & \\ & \end{pmatrix},$$

where a $\star$ represents a nonzero element. After all columns are processed, the following least-squares system is obtained.

$$\bar{H}_8 = \left(\begin{array}{cccccc} \star & \star & \star & \star & \star & \star \\ & \star & \star & \star & \star & \star \\ & & \star & \star & \star & \star \\ & & & \star & \star & \star \\ & & & & \star & \star \\ & & & & & \star \\ \hline & & & & & \\ & & & & & \end{array}\right) \qquad \bar{G} = \left(\begin{array}{cc} \star & \star \\ \star & \star \\ \star & \star \\ \star & \star \\ \star & \star \\ \star & \star \\ \hline \star & \star \\ & \star \end{array}\right).$$

To obtain the least-squares solutions for each right-hand side, ignore anything below the horizontal lines in the above matrices and solve the resulting triangular systems. The residual norm of the i-th system for the original problem is the 2-norm of the vector consisting of the components $m+1$, through $m+i$ in the i-th column of the above block of right-hand sides.

Generally speaking, the block methods are of great practical value in applications involving linear systems with multiple right-hand sides. However, they are not as well studied from the theoretical point of view. Perhaps, one of the reasons is the lack of a convincing analogue for the relationship with orthogonal polynomials, established in subsection 6.6.2 for the single-vector Lanczos algorithm. The block version of the Lanczos algorithm has not been covered but the generalization is straightforward.

EXERCISES

1. In the Householder implementation of the Arnoldi algorithm, show the following points of detail:

a. Q_{j+1} is unitary and its inverse is Q_{j+1}^T.

b. $Q_{j+1}^T = P_1 P_2 \ldots P_{j+1}$.

c. $Q_{j+1}^T e_i = v_i$ for $i < j$.

d. $Q_{j+1} A V_m = V_{m+1}[e_1, e_2, \ldots, e_{j+1}]\bar{H}_m$, where e_i is the i-th column of the $n \times n$ identity matrix.

e. The v_i's are orthonormal.

f. The vectors $v_1, \ldots, v_j$ are equal to the Arnoldi vectors produced by the Gram-Schmidt version, except possibly for a scaling factor.

2. Rewrite the Householder implementation of the Arnoldi algorithm with more detail. In particular, define precisely the Householder vector w_j used at step j (lines 3-5).

3. Consider the Householder implementation of the Arnoldi algorithm. Give a detailed operation count of the algorithm and compare it with the Gram-Schmidt and Modified Gram-Schmidt algorithm.

4. Derive the basic version of GMRES by using the standard formula (5.7) with $V = V_m$ and $W = AV_m$.

5. Derive a version of the DIOM algorithm which includes partial pivoting in the solution of the Hessenberg system.

6. Show how the GMRES and FOM methods will converge on the linear system $Ax = b$ when

$$A = \begin{pmatrix} & & & & 1 \\ 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and with $x_0 = 0$.

7. Give a full proof of Proposition 6.11.

8. Let a matrix A have the form

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

Assume that (full) GMRES is used to solve a linear system, with the coefficient matrix A. What is the maximum number of steps that GMRES would require to converge?

9. Let a matrix A have the form:

$$A = \begin{pmatrix} I & Y \\ 0 & S \end{pmatrix}.$$

Assume that (full) GMRES is used to solve a linear system with the coefficient matrix A. Let

$$r_0 = \begin{pmatrix} r_0^{(1)} \\ r_0^{(2)} \end{pmatrix}$$

be the initial residual vector. It is assumed that the degree of the minimal polynomial of $r_0^{(2)}$ with respect to S (i.e., its grade) is k. What is the maximum number of steps that GMRES would require to converge for this matrix? [Hint: Evaluate the sum $\sum_{i=0}^{k} \beta_i (A^{i+1} - A^i) r_0$ where $\sum_{i=0}^{k} \beta_i t^i$ is the minimal polynomial of $r_0^{(2)}$ with respect to S.]

10. Let

$$A = \begin{pmatrix} I & Y_2 & & & & \\ & I & Y_3 & & & \\ & & I & \ddots & & \\ & & & I & Y_{k-1} & \\ & & & & I & Y_k \\ & & & & & I \end{pmatrix}.$$

a. Show that $(I - A)^k = 0$.

b. Assume that (full) GMRES is used to solve a linear system with the coefficient matrix A. What is the maximum number of steps that GMRES would require to converge?

11. Show that if H_m is nonsingular, i.e., x_m^F is defined, and if $x_m^G = x_m^F$, then $r_m^G = r_m^F = 0$, i.e., both the GMRES and FOM solutions are exact. [Hint: use the relation (6.46) and Proposition 6.11 or Proposition 6.12.]

12. Derive the relation (6.49) from (6.47). [Hint: Use the fact that the vectors on the right-hand side of (6.47) are orthogonal.]

13. In the Householder-GMRES algorithm the approximate solution can be computed by formulas (6.25-6.27). What is the exact cost of this alternative (compare memory as well as arithmetic requirements)? How does it compare with the cost of keeping the v_i's?

14. An alternative to formulas (6.25-6.27) for accumulating the approximate solution in the Householder-GMRES algorithm without keeping the v_i's is to compute x_m as

$$x_m = x_0 + P_1 P_2 \ldots P_m y$$

where y is a certain n-dimensional vector to be determined. (1) What is the vector y for the above formula in order to compute the correct approximate solution x_m? [Hint: Exploit (6.11).] (2) Write down an alternative to formulas (6.25-6.27) derived from this approach. (3) Compare the cost of this approach with the cost of using (6.25-6.27).

15. Obtain the formula (6.76) from (6.75).

16. Show that the determinant of the matrix T_m in (6.82) is given by

$$\det\ (T_m) = \frac{1}{\prod_{i=0}^{m-1} \alpha_i}$$

17. The Lanczos algorithm is more closely related to the implementation of Algorithm 6.18 of the Conjugate Gradient algorithm. As a result the Lanczos coefficients δ_{j+1} and η_{j+1} are easier to extract from this algorithm than from Algorithm 6.17. Obtain formulas for these coefficients from the coefficients generated by Algorithm 6.18, as was done in Section 6.7.3 for the standard CG algorithm.

18. Show that if the rotations generated in the course of the GMRES (and DQGMRES) algorithm are such that

$$|c_m| \geq c > 0,$$

then GMRES, DQGMRES, and FOM will all converge.

19. Show the exact expression of the residual vector in the basis $v_1, v_2, \ldots, v_{m+1}$ for either GMRES or DQGMRES. [Hint: A starting point is (6.52).]

20. Prove that the inequality (6.56) is sharper than (6.53), in the sense that $\zeta_{m+1} \leq \sqrt{m - k + 11}$ (for $m \geq k$). [Hint: Use Cauchy-Schwarz inequality on (6.56).]

21. Denote by S_m the unit upper triangular matrix S in the proof of Theorem 6.1 which is obtained from the Gram-Schmidt process (exact arithmetic assumed) applied to the incomplete orthogonalization basis V_m. Show that the Hessenberg matrix $\bar{H}_m^Q$ obtained in the incomplete orthogonalization process is related to the Hessenberg matrix $\bar{H}_m^G$ obtained from the (complete) Arnoldi process by

$$\bar{H}_m^G = S_{m+1}^{-1}\bar{H}_m^Q S_m.$$

NOTES AND REFERENCES. Lemma 6.1 was proved by Roland Freund [95] in a slightly different form. Proposition 6.12 is due to Brown [43] who proved a number of other theoretical results, including Proposition 6.11. Recently, Cullum and Greenbaum [63] discussed further relationships between FOM and GMRES and other Krylov subspace methods.

The Conjugate Gradient method was developed independently and in different forms by Lanczos [142] and Hesteness and Stiefel [120]. The method was essentially viewed as a direct solution technique and was abandoned early on because it did not compare well with other existing techniques. For example, in inexact arithmetic, the method does not terminate in n steps as is predicted by the theory. This is caused by the severe loss of of orthogonality of vector quantities generated by the algorithm. As a result, research on Krylov-type methods remained dormant for over two decades thereafter. This changed in the early 1970s when several researchers discovered that this loss of orthogonality did not prevent convergence. The observations were made and explained for eigenvalue problems [158, 106] as well as linear systems [167]. The early to the middle 1980s saw the development of a new class of methods for solving nonsymmetric linear systems [13, 14, 127, 172, 173, 185, 218]. The works of Faber and Manteuffel [85] and Voevodin [219] showed that one could not find optimal methods which, like CG, are based on short-term recurrences. Many of the methods developed are mathematically equivalent, in the sense that they realize the same projection process, with different implementations.

The Householder version of GMRES is due to Walker [221]. The Quasi-GMRES algorithm described in Section 6.5.7 was initially described by Brown and Hindmarsh [44], although the direct version DQGMRES was only discussed recently in [187]. The proof of Theorem 6.1 can be found in [152] for the QMR algorithm.

The non-optimality of the Chebyshev polynomials on ellipses in the complex plane was established by Fischer and Freund [90]. Prior to this, a 1963 paper by Clayton [59] was believed to have established the optimality for the special case where the ellipse has real foci and γ is real.

Until recently, little attention has been given to block Krylov methods. In addition to their attraction for solving linear systems with several right-hand sides [177, 196], these techniques can also help reduce the effect of the sequential inner products in parallel environments and minimize I/O costs in out-of-core implementations. The block-GMRES algorithm is analyzed by Simoncini and Gallopoulos [197] and in [184]. Alternatives to GMRES which require fewer inner products have been proposed by Sadok [188] and Jbilou [125]. Sadok investigated a GMRES-like method based on the Hessenberg algorithm [227], while Jbilou proposed a multi-dimensional generalization of Gastinel's method seen in Exercise 2 of Chapter 5. ■

CHAPTER

7

KRYLOV SUBSPACE METHODS PART II

The previous chapter considered a number of Krylov subspace methods which relied on some form of orthogonalization of the Krylov vectors in order to compute an approximate solution. This chapter will describe a class of Krylov subspace methods which are instead based on a biorthogonalization algorithm due to Lanczos. These are projection methods that are intrinsically non-orthogonal. They have some appealing properties, but are harder to analyze theoretically.

7.1 LANCZOS BIORTHOGONALIZATION

The Lanczos biorthogonalization algorithm is an extension to nonsymmetric matrices of the symmetric Lanczos algorithm seen in the previous chapter. One such extension, the Arnoldi procedure, has already been seen. However, the nonsymmetric Lanczos algorithm is quite different in concept from Arnoldi's method because it relies on biorthogonal sequences instead of orthogonal sequences.

7.1.1 THE ALGORITHM

The algorithm proposed by Lanczos for nonsymmetric matrices builds a pair of biorthogonal bases for the two subspaces

$$\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$

and

$$\mathcal{K}_m(A^T, w_1) = \text{span}\{w_1, A^T w_1, \ldots, (A^T)^{m-1}w_1\}.$$

The algorithm that achieves this is the following.

ALGORITHM 7.1: The Lanczos Biorthogonalization Procedure

1. *Choose two vectors* v_1, w_1 *such that* $(v_1, w_1) = 1$.
2. *Set* $\beta_1 = \delta_1 \equiv 0, w_0 = v_0 \equiv 0$
3. *For* $j = 1, 2, \ldots, m$ *Do:*
4. $\quad \alpha_j = (Av_j, w_j)$
5. $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6. $\quad \hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7. $\quad \delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. *If* $\delta_{j+1} = 0$ *Stop*
8. $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9. $\quad w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10. $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
11. *EndDo*

Note that there are numerous ways to choose the scalars $\delta_{j+1}, \beta_{j+1}$ in lines 7 and 8. These two parameters are scaling factors for the two vectors v_{j+1} and w_{j+1} and can be selected in any manner to ensure that $(v_{j+1}, w_{j+1}) = 1$. As a result of lines 9 and 10 of the algorithm, it is only necessary to choose two scalars $\beta_{j+1}, \delta_{j+1}$ that satisfy the equality

$$\delta_{j+1}\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1}). \tag{7.1}$$

The choice taken in the above algorithm scales the two vectors so that they are divided by two scalars which have the same modulus. Both vectors can also be scaled by their 2-norms. In that case, the inner product of v_{j+1} and w_{j+1} is no longer equal to 1 and the algorithm must be modified accordingly; see Exercise 3.

Consider the case where the pair of scalars $\delta_{j+1}, \beta_{j+1}$ is *any pair* that satisfies the relation (7.1). Denote by T_m the tridiagonal matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & . & . & . & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix}. \tag{7.2}$$

If the determinations of $\beta_{j+1}, \delta_{j+1}$ of lines 7–8 are used, then the δ_j's are positive and $\beta_j = \pm\delta_j$.

Observe from the algorithm that the vectors v_i belong to $\mathcal{K}_m(A, v_1)$, while the w_j's are in $\mathcal{K}_m(A^T, w_1)$. In fact, the following proposition can be proved.

PROPOSITION 7.1 *If the algorithm does not break down before step* m*, then the vectors* $v_i, i = 1, \ldots, m$*, and* $w_j, j = 1, \ldots, m$*, form a biorthogonal system, i.e.,*

$$(v_j, w_i) = \delta_{ij} \quad 1 \le i,\ j \le m.$$

Moreover, $\{v_i\}_{i=1,2,\ldots,m}$ *is a basis of* $\mathcal{K}_m(A, v_1)$ *and* $\{w_i\}_{i=1,2,\ldots,m}$ *is a basis of* $\mathcal{K}_m(A^T, w_1)$ *and the following relations hold,*

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T, \tag{7.3}$$

$$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T, \tag{7.4}$$
$$W_m^T A V_m = T_m. \tag{7.5}$$

Proof. The biorthogonality of the vectors v_i, w_i will be shown by induction. By assumption $(v_1, w_1) = 1$. Assume now that the vectors $v_1, \ldots v_j$ and $w_1, \ldots w_j$ are biorthogonal, and let us prove that the vectors $v_1, \ldots v_{j+1}$ and $w_1, \ldots w_{j+1}$ are biorthogonal.

First, we show that $(v_{j+1}, w_i) = 0$ for $i \leq j$. When $i = j$, then

$$(v_{j+1}, w_j) = \delta_{j+1}^{-1}[(Av_j, w_j) - \alpha_j(v_j, w_j) - \beta_j(v_{j-1}, w_j)].$$

The last inner product in the above expression vanishes by the induction hypothesis. The two other terms cancel each other by the definition of α_j and the fact that $(v_j, w_j) = 1$. Consider now the inner product (v_{j+1}, w_i) with $i < j$,

$$\begin{aligned}(v_{j+1}, w_i) &= \delta_{j+1}^{-1}[(Av_j, w_i) - \alpha_j(v_j, w_i) - \beta_j(v_{j-1}, w_i)]\\ &= \delta_{j+1}^{-1}[(v_j, A^T w_i) - \beta_j(v_{j-1}, w_i)]\\ &= \delta_{j+1}^{-1}[(v_j, \beta_{i+1} w_{i+1} + \alpha_i w_i + \delta_i w_{i-1}) - \beta_j(v_{j-1}, w_i)].\end{aligned}$$

For $i < j - 1$, all of the inner products in the above expression vanish, by the induction hypothesis. For $i = j - 1$, the inner product is

$$\begin{aligned}(v_{j+1}, w_{j-1}) &= \delta_{j+1}^{-1}[(v_j, \beta_j w_j + \alpha_{j-1} w_{j-1} + \delta_{j-1} w_{j-2}) - \beta_j(v_{j-1}, w_{j-1})]\\ &= \delta_{j+1}^{-1}[\beta_j(v_j, w_j) - \beta_j(v_{j-1}, w_{j-1})]\\ &= 0.\end{aligned}$$

It can be proved in exactly the same way that $(v_i, w_{j+1}) = 0$ for $i \leq j$. Finally, by construction $(v_{j+1}, w_{j+1}) = 1$. This completes the induction proof. The proof of the matrix relations (7.3–7.5) is similar to that of the relations (6.4–6.6) in Arnoldi's method. ■

The relations (7.3–7.5) allow us to interpret the algorithm. The matrix T_m is the projection of A obtained from an oblique projection process onto $\mathcal{K}_m(A, v_1)$ and orthogonally to $\mathcal{K}_m(A^T, w_1)$. Similarly, T_m^T represents the projection of A^T on $\mathcal{K}_m(A^T, w_1)$ and orthogonally to $\mathcal{K}_m(A, v_1)$. Thus, an interesting new feature here is that the operators A and A^T play a dual role because similar operations are performed with them. In fact, two linear systems are solved implicitly, one with A and the other with A^T. If there were two linear systems to solve, one with A and the other with A^T, then this algorithm is suitable. Otherwise, the operations with A^T are essentially wasted. Later a number of alternative techniques developed in the literature will be introduced that avoid the use of A^T.

From a practical point of view, the Lanczos algorithm has a significant advantage over Arnoldi's method because it requires only a few vectors of storage, if no reorthogonalization is performed. Specifically, six vectors of length n are needed, plus some storage for the tridiagonal matrix, no matter how large m is.

On the other hand, there are potentially more opportunities for breakdown with the nonsymmetric Lanczos method. The algorithm will break down whenever δ_{j+1} as defined in line 7 vanishes. This is examined more carefully in the next section. In practice, the difficulties are more likely to be caused by the near occurrence of this phenomenon. A look at the algorithm indicates that the Lanczos vectors may have to be scaled by small quantities when this happens. After a few steps the cumulated effect of these scalings may introduce excessive rounding errors.

Since the subspace from which the approximations are taken is identical to that of Arnoldi's method, the same bounds for the distance $\|(I - \Pi_m)u\|_2$ are valid. However, this does not mean in any way that the approximations obtained by the two methods are likely to be similar in quality. The theoretical bounds shown in Chapter 5 indicate that the norm of the projector may play a significant role.

7.1.2 PRACTICAL IMPLEMENTATIONS

There are various ways to improve the standard nonsymmetric Lanczos algorithm which we now discuss briefly. A major concern here is the potential breakdowns or "near breakdowns" in the algorithm. There exist a number of approaches that have been developed to avoid such breakdowns. Other approaches do not attempt to eliminate the breakdown, but rather try to deal with it. The pros and cons of these strategies will be discussed after the various existing scenarios are described.

Algorithm 7.1 will abort in line 7 whenever,

$$(\hat{v}_{j+1}, \hat{w}_{j+1}) = 0. \tag{7.6}$$

This can arise in two different ways. Either one of the two vectors $\hat{v}_{j+1}$ or $\hat{w}_{j+1}$ vanishes, or they are both nonzero, but their inner product is zero. The first case is the "lucky breakdown" scenario which has been seen for symmetric matrices. Thus, if $\hat{v}_{j+1} = 0$ then $\mathrm{span}\{V_j\}$ is invariant and, as was seen in Chapter 5, the approximate solution is exact. If $\hat{w}_{j+1} = 0$ then $\mathrm{span}\{W_j\}$ is invariant. However, in this situation nothing can be said about the approximate solution for the linear system with A. If the algorithm is being used to solve a pair of linear systems, one with A and a *dual* system with A^T, then the approximate solution for the dual system will be exact in this case. The second scenario in which (7.6) can occur is when neither of the two vectors is zero, but their inner product is zero. Wilkinson (see [227], p. 389) called this a *serious breakdown*. Fortunately, there are cures for this problem which allow the algorithm to continue in most cases. The corresponding modifications of the algorithm are often put under the denomination *Look-Ahead Lanczos* algorithms. There are also rare cases of *incurable breakdowns* which will not be discussed here (see references [161] and [206]).

The main idea of Look-Ahead variants of the Lanczos algorithm is that the pair v_{j+2}, w_{j+2} can often be defined even though the pair v_{j+1}, w_{j+1} is not defined. The algorithm can be pursued from that iterate as before until a new breakdown is encountered. If the pair v_{j+2}, w_{j+2} cannot be defined then the pair v_{j+3}, w_{j+3} can be tried, and so on. To better explain the idea, it is best to refer to the connection with orthogonal polynomials mentioned earlier for the symmetric case. The relationship can be extended to the nonsymmetric case by defining the bilinear form on the subspace $\mathbb{P}_{m-1}$

$$< p, q > = (p(A)v_1, q(A^T)w_1). \tag{7.7}$$

Unfortunately, this is now an "indefinite inner product" in general since $< p, p >$ can be zero or even negative. Note that there is a polynomial p_j of degree j such that $\hat{v}_{j+1} = p_j(A)v_1$ and, in fact, the same polynomial intervenes in the equivalent expression of w_{j+1}.

More precisely, there is a scalar γ_j such that $\hat{w}_{j+1} = \gamma_j p_j(A^T)v_1$. Similar to the symmetric case, the nonsymmetric Lanczos algorithm attempts to compute a sequence of polynomials that are orthogonal with respect to the indefinite inner product defined above. If we define the moment matrix

$$M_k = \{< x^{i-1}, x^{j-1} >\}_{i,j=1,\ldots,k}$$

then this process is mathematically equivalent to the computation of the factorization

$$M_k = L_k U_k$$

of the moment matrix M_k, in which U_k is upper triangular and L_k is lower triangular. Note that M_k is a Hankel matrix, i.e., its coefficients m_{ij} are constant along anti-diagonals, i.e., for $i + j = constant$.

Because

$$< p_j, p_j >= \gamma_j (p_j(A)v_1, p_j(A^T)w_1),$$

we observe that there is a serious breakdown at step j if and only if the indefinite norm of the polynomial p_j at step j vanishes. If this polynomial is skipped, it may still be possible to compute p_{j+1} and continue to generate the sequence. To explain this simply, consider

$$q_j(t) = xp_{j-1}(t) \quad \text{and} \quad q_{j+1}(t) = x^2 p_{j-1}(t).$$

Both q_j and q_{j+1} are orthogonal to the polynomials $p_1, \ldots, p_{j-2}$. We can define (somewhat arbitrarily) $p_j = q_j$, and then p_{j+1} can be obtained by orthogonalizing q_{j+1} against p_{j-1} and p_j. It is clear that the resulting polynomial will then be orthogonal against all polynomials of degree $\leq j$; see Exercise 5. Therefore, the algorithm can be continued from step $j + 1$ in the same manner. Exercise 5 generalizes this for the case where k polynomials are skipped rather than just one. This is a simplified description of the mechanism which underlies the various versions of Look-Ahead Lanczos algorithms proposed in the literature. The Parlett-Taylor-Liu implementation [161] is based on the observation that the algorithm breaks because the pivots encountered during the LU factorization of the moment matrix M_k vanish. Then, divisions by zero are avoided by performing *implicitly* a pivot with a 2×2 matrix rather than using a standard 1×1 pivot.

The drawback of Look-Ahead implementations is the nonnegligible added complexity. Besides the difficulty of identifying these near breakdown situations, the matrix T_m ceases to be tridiagonal. Indeed, whenever a step is skipped, elements are introduced above the superdiagonal positions, in some subsequent step. In the context of linear systems, near breakdowns are rare and their effect generally benign. Therefore, a simpler remedy, such as restarting the Lanczos procedure, may well be adequate. For eigenvalue problems, Look-Ahead strategies may be more justified.

7.2 THE LANCZOS ALGORITHM FOR LINEAR SYSTEMS

We present in this section a brief description of the Lanczos method for solving nonsymmetric linear systems. Consider the (single) linear system:

$$Ax = b \tag{7.8}$$

where A is $n \times n$ and nonsymmetric. Suppose that a guess x_0 to the solution is available and let its residual vector be $r_0 = b - Ax_0$. Then the Lanczos algorithm for solving (7.8) can be described as follows.

ALGORITHM 7.2: Two-sided Lanczos Algorithm for Linear Systems

1. *Compute* $r_0 = b - Ax_0$ *and* $\beta := \|r_0\|_2$
2. *Run* m *steps of the nonsymmetric Lanczos Algorithm, i.e.,*
3. *Start with* $v_1 := r_0/\beta$*, and any* w_1 *such that* $(v_1, w_1) = 1$
4. *Generate the Lanczos vectors* $v_1, \ldots, v_m, w_1, \ldots, w_m$
5. *and the tridiagonal matrix* T_m *from Algorithm 7.1.*
6. *Compute* $y_m = T_m^{-1}(\beta e_1)$ *and* $x_m := x_0 + V_m y_m$.

Note that it is possible to incorporate a convergence test when generating the Lanczos vectors in the second step without computing the approximate solution explicitly. This is due to the following formula, which is similar to Equation (6.66) for the symmetric case,

$$\|b - Ax_j\|_2 = |\delta_{j+1} e_j^T y_j| \; \|v_{j+1}\|_2, \tag{7.9}$$

and which can be proved in the same way, by using (7.3). This formula gives us the residual norm inexpensively without generating the approximate solution itself.

7.3 THE BCG AND QMR ALGORITHMS

The Biconjugate Gradient (BCG) algorithm can be derived from Algorithm 7.1 in exactly the same way as the Conjugate Gradient method was derived from Algorithm 6.14. The algorithm was first proposed by Lanczos [142] in 1952 and then in a different form (Conjugate Gradient-like version) by Fletcher [92] in 1974. Implicitly, the algorithm solves not only the original system $Ax = b$ but also a dual linear system $A^T x^* = b^*$ with A^T. This dual system is often ignored in the formulations of the algorithm.

7.3.1 THE BICONJUGATE GRADIENT ALGORITHM

The Biconjugate Gradient (BCG) algorithm is a projection process onto

$$\mathcal{K}_m = span\{v_1, Av_1, \cdots, A^{m-1}v_1\}$$

orthogonally to

$$\mathcal{L}_m = span\{w_1, A^T w_1, \cdots, (A^T)^{m-1}w_1\}$$

taking, as usual, $v_1 = r_0/\|r_0\|_2$. The vector w_1 is arbitrary, provided $(v_1, w_1) \neq 0$, but it is often chosen to be equal to v_1. If there is a dual system $A^T x^* = b^*$ to solve with A^T, then w_1 is obtained by scaling the initial residual $b^* - A^T x_0^*$.

Proceeding in the same manner as for the derivation of the Conjugate Gradient algorithm from the symmetric Lanczos algorithm, we write the LDU decomposition of T_m as

$$T_m = L_m U_m \tag{7.10}$$

and define

$$P_m = V_m U_m^{-1}. \tag{7.11}$$

The solution is then expressed as

$$\begin{aligned} x_m &= x_0 + V_m T_m^{-1}(\beta e_1) \\ &= x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1) \\ &= x_0 + P_m L_m^{-1}(\beta e_1). \end{aligned}$$

Notice that the solution x_m is updatable from x_{m-1} in a similar way to the Conjugate Gradient algorithm. Like the Conjugate Gradient algorithm, the vectors r_j and r_j^* are in the same direction as v_{j+1} and w_{j+1}, respectively. Hence, they form a biorthogonal sequence. Define similarly the matrix

$$P_m^* = W_m L_m^{-T}. \tag{7.12}$$

Clearly, the column-vectors p_i^* of P_m^* and those p_i of P_m are A-conjugate, since,

$$(P_m^*)^T A P_m = L_m^{-1} W_m^T A V_m U_m^{-1} = L_m^{-1} T_m U_m^{-1} = I.$$

Utilizing this information, a Conjugate Gradient–like algorithm can be easily derived from the Lanczos procedure.

ALGORITHM 7.3: Biconjugate Gradient (BCG)

1. *Compute* $r_0 := b - Ax_0$. *Choose* r_0^* *such that* $(r_0, r_0^*) \neq 0$.
2. *Set,* $p_0 := r_0$, $p_0^* := r_0^*$
3. *For* $j = 0, 1, \ldots$, *until convergence Do:*
4. $\quad \alpha_j := (r_j, r_j^*)/(Ap_j, p_j^*)$
5. $\quad x_{j+1} := x_j + \alpha_j p_j$
6. $\quad r_{j+1} := r_j - \alpha_j A p_j$
7. $\quad r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
8. $\quad \beta_j := (r_{j+1}, r_{j+1}^*)/(r_j, r_j^*)$

9. $p_{j+1} := r_{j+1} + \beta_j p_j$
10. $p^*_{j+1} := r^*_{j+1} + \beta_j p^*_j$
11. EndDo

If a dual system with A^T is being solved, then in line 1 r_0^* should be defined as $r_0^* = b^* - A^T x_0^*$ and the update $x^*_{j+1} := x^*_j + \alpha_j p^*_j$ to the dual approximate solution must beinserted after line 5. The vectors produced by this algorithm satisfy a few biorthogonality properties stated in the following proposition.

PROPOSITION 7.2 *The vectors produced by the Biconjugate Gradient algorithm satisfy the following orthogonality properties:*

$$(r_j, r_i^*) = 0, \quad \text{for } i \neq j, \tag{7.13}$$

$$(Ap_j, p_i^*) = 0, \quad \text{for } i \neq j. \tag{7.14}$$

Proof. The proof is either by induction or by simply exploiting the relations between the vectors r_j, r_j^*, p_j, p_j^*, and the vector columns of the matrices V_m, W_m, P_m, P_m^*. This is left as an exercise. ∎

Example 7.1 Table 7.1 shows the results of applying the BCG algorithm with no preconditioning to three of the test problems described in Section 3.7. See Example 6.1 for the meaning of the column headers in the table. Recall that Iters really represents the number of matrix-by-vector multiplications rather the number of Biconjugate Gradient steps.

Matrix	Iters	Kflops	Residual	Error
F2DA	163	2974	0.17E-03	0.86E-04
F3D	123	10768	0.34E-04	0.17E-03
ORS	301	6622	0.50E-01	0.37E-02

Table 7.1 *A test run of BCG without preconditioning.*

Thus, the number 163 in the first line represents 81 steps of BCG, which require 81×2 matrix-by-vector products in the iteration, and an extra one to compute the initial residual.

7.3.2 QUASI-MINIMAL RESIDUAL ALGORITHM

The result of the Lanczos algorithm is a relation of the form

$$AV_m = V_{m+1}\bar{T}_m \tag{7.15}$$

in which $\bar{T}_m$ is the $(m+1) \times m$ tridiagonal matrix

$$\bar{T}_m = \begin{pmatrix} T_m \\ \delta_{m+1} e_m^T \end{pmatrix}.$$

Now (7.15) can be exploited in the same way as was done to develop GMRES. If v_1 is defined as a multiple of r_0, i.e., if $v_1 = \beta r_0$, then the residual vector associated with an approximate solution of the form

$$x = x_0 + V_m y$$

is given by

$$\begin{aligned} b - Ax &= b - A\left(x_0 + V_m y\right) \\ &= r_0 - AV_m y \\ &= \beta v_1 - V_{m+1}\bar{T}_m y \\ &= V_{m+1}\left(\beta e_1 - \bar{T}_m y\right). \end{aligned} \tag{7.16}$$

The norm of the residual vector is therefore

$$\|b - Ax\| = \|V_{m+1}\left(\beta e_1 - \bar{T}_m y\right)\|_2. \tag{7.17}$$

If the column-vectors of V_{m+1} were orthonormal, then we would have $\|b - Ax\| = \|\beta e_1 - \bar{T}_m y\|_2$, as in GMRES. Therefore, a least-squares solution could be obtained from the Krylov subspace by minimizing $\|\beta e_1 - \bar{T}_m y\|_2$ over y. In the Lanczos algorithm, the v_i's are not orthonormal. However, it is still a reasonable idea to minimize the function

$$J(y) \equiv \|\beta e_1 - \bar{T}_m y\|_2$$

over y and compute the corresponding approximate solution $x_0 + V_m y$. The resulting solution is called the *Quasi-Minimal Residual approximation*.

Thus, the Quasi-Minimal Residual (QMR) approximation from the m-th Krylov subspace is obtained as $x_0 + V_m y_m$, where y_m minimizes the function $J(y) = \|\beta e_1 - \bar{T}_m y\|_2$, i.e., just as in GMRES, except that the Arnoldi process is replaced by the Lanczos process. Because of the simple structure of the matrix $\bar{T}_m$, the following algorithm is obtained, which is adapted from the DQGMRES algorithm (Algorithm 6.13).

ALGORITHM 7.4: QMR

1. *Compute* $r_0 = b - Ax_0$ *and* $\gamma_1 := \|r_0\|_2$, $w_1 := v_1 := r_0/\gamma_1$
2. *For* $m = 1, 2, \ldots$, *until convergence Do:*
3.   *Compute* α_m, δ_{m+1} *and* v_{m+1}, w_{m+1} *as in Algorithm 7.1*
4.   *Update the QR factorization of* $\bar{T}_m$, *i.e.,*
5.   *Apply* Ω_i, $i = m-2, m-1$ *to the* m*-th column of* $\bar{T}_m$
6.   *Compute the rotation coefficients* c_m, s_m *by (6.31)*
7.   *Apply rotation* Ω_m, *to* $\bar{T}_m$ *and* $\bar{g}_m$, *i.e., compute:*
8. $\gamma_{m+1} := -s_m \gamma_m$,
9. $\gamma_m := c_m \gamma_m$, *and,*
10. $\alpha_m := c_m \alpha_m + s_m \delta_{m+1} \left(= \sqrt{\delta_{m+1}^2 + \alpha_m^2}\right)$
11. $p_m = \left(v_m - \sum_{i=m-2}^{m-1} t_{im} p_i\right)/t_{mm}$
12. $x_m = x_{m-1} + \gamma_m p_m$
13.   *If* $|\gamma_{m+1}|$ *is small enough then Stop*
14. *EndDo*

The following proposition establishes a result on the residual norm of the solution. It is similar to Proposition 6.9.

PROPOSITION 7.3 *The residual norm of the approximate solution x_m satisfies the relation*

$$\|b - Ax_m\| \le \|V_{m+1}\|_2 \, |s_1 s_2 \dots s_m| \, \|r_0\|_2. \tag{7.18}$$

Proof. According to (7.16) the residual norm is given by

$$b - Ax_m = V_{m+1}[\beta e_1 - \bar{T}_m y_m] \tag{7.19}$$

and using the same notation as in Proposition 6.9, referring to (6.37)

$$\|\beta e_1 - \bar{H}_m y\|_2^2 = |\gamma_{m+1}|^2 + \|g_m - R_m y\|_2^2$$

in which $g_m - R_m y = 0$ by the minimization procedure. In addition, by (6.40) we have

$$\gamma_{m+1} = s_1 \dots s_m \gamma_1.$$

The result follows immediately using (7.19). ■

The following simple upper bound for $\|V_{m+1}\|_2$ can be used to estimate the residual norm:

$$\|V_{m+1}\|_2 \le \left[\sum_{i=1}^{m+1} \|v_i\|_2^2 \right]^{1/2}.$$

Observe that ideas similar to those used for DQGMRES can be exploited to obtain a better estimate of the residual norm. Also, note that the relation (6.57) for DQGMRES holds. More interestingly, Theorem 6.1 is also valid and it is now restated for QMR.

THEOREM 7.1 *Assume that the Lanczos algorithm does not break down on or before step m and let V_{m+1} be the Lanczos basis obtained at step m. Let r_m^Q and r_m^G be the residual norms obtained after m steps of the QMR and GMRES algorithms, respectively. Then,*

$$\|r_m^Q\|_2 \le \kappa_2(V_{m+1}) \|r_m^G\|_2.$$

The proof of this theorem is essentially identical with that of Theorem 6.1. Note that V_{m+1} is now known to be of full rank, so we need not make this assumption as in Theorem 6.1.

7.4 TRANSPOSE-FREE VARIANTS

Each step of the Biconjugate Gradient algorithm and QMR requires a matrix-by-vector product with both A and A^T. However, observe that the vectors p_i^* or w_j generated with A^T do not contribute directly to the solution. Instead, they are used only to obtain the scalars needed in the algorithm, e.g., the scalars α_j and β_j for BCG. The question arises as to whether or not it is possible to bypass the use of the transpose of A and still generate iterates that are related to those of the BCG algorithm. One of the motivations for this question is that, in some applications, A is available only through some approximations and not explicitly. In such situations, the transpose of A is usually not available. A simple example is when a CG-like algorithm is used in the context of Newton's iteration for solving $F(u) = 0$. The linear system that arises at each Newton step can be solved without having to compute the Jacobian $J(u_k)$ at the current iterate u_k explicitly, by using the difference formula

$$J(u_k)v = \frac{F(u_k + \epsilon v) - F(u_k)}{\epsilon}.$$

This allows the action of this Jacobian to be computed on an arbitrary vector v. Unfortunately, there is no similar formula for performing operations with the transpose of $J(u_k)$.

7.4.1 CONJUGATE GRADIENT SQUARED

The Conjugate Gradient Squared algorithm was developed by Sonneveld in 1984 [201], mainly to avoid using the transpose of A in the BCG and to gain faster convergence for roughly the same computational cost. The main idea is based on the following simple observation. In the BCG algorithm, the residual vector at step j can be expressed as

$$r_j = \phi_j(A)r_0 \tag{7.20}$$

where ϕ_j is a certain polynomial of degree j satisfying the constraint $\phi_j(0) = 1$. Similarly, the conjugate-direction polynomial $\pi_j(t)$ is given by

$$p_j = \pi_j(A)r_0, \tag{7.21}$$

in which π_j is a polynomial of degree j. From the algorithm, observe that the directions r_j^* and p_j^* are defined through the same recurrences as r_j and p_j in which A is replaced by A^T and, as a result,

$$r_j^* = \phi_j(A^T)r_0^*, \quad p_j^* = \pi_j(A^T)r_0^*.$$

Also, note that the scalar α_j in BCG is given by

$$\alpha_j = \frac{(\phi_j(A)r_0, \phi_j(A^T)r_0^*)}{(A\pi_j(A)r_0, \pi_j(A^T)r_0^*)} = \frac{(\phi_j^2(A)r_0, r_0^*)}{(A\pi_j^2(A)r_0, r_0^*)}$$

which indicates that if it is possible to get a recursion for the vectors $\phi_j^2(A)r_0$ and $\pi_j^2(A)r_0$, then computing α_j and, similarly, β_j causes no problem. Hence, the idea of seeking an algorithm which would give a sequence of iterates whose residual norms r_j' satisfy

$$r_j' = \phi_j^2(A)r_0. \tag{7.22}$$

The derivation of the method relies on simple algebra only. To establish the desired recurrences for the squared polynomials, start with the recurrences that define ϕ_j and π_j, which are,

$$\phi_{j+1}(t) = \phi_j(t) - \alpha_j t\pi_j(t), \tag{7.23}$$

$$\pi_{j+1}(t) = \phi_{j+1}(t) + \beta_j\pi_j(t). \tag{7.24}$$

If the above relations are squared we get

$$\phi_{j+1}^2(t) = \phi_j^2(t) - 2\alpha_j t\pi_j(t)\phi_j(t) + \alpha_j^2 t^2 \pi_j^2(t),$$
$$\pi_{j+1}^2(t) = \phi_{j+1}^2(t) + 2\beta_j\phi_{j+1}(t)\pi_j(t) + \beta_j^2\pi_j(t)^2.$$

If it were not for the cross terms $\pi_j(t)\phi_j(t)$ and $\phi_{j+1}(t)\pi_j(t)$ on the right-hand sides, these equations would form an updatable recurrence system. The solution is to introduce one of these two cross terms, namely, $\phi_{j+1}(t)\pi_j(t)$, as a third member of the recurrence. For the other term, i.e., $\pi_j(t)\phi_j(t)$, we can exploit the relation

$$\phi_j(t)\pi_j(t) = \phi_j(t)\left(\phi_j(t) + \beta_{j-1}\pi_{j-1}(t)\right) = \phi_j^2(t) + \beta_{j-1}\phi_j(t)\pi_{j-1}(t).$$

By putting these relations together the following recurrences can be derived, in which the variable (t) is omitted where there is no ambiguity:

$$\phi_{j+1}^2 = \phi_j^2 - \alpha_j t\left(2\phi_j^2 + 2\beta_{j-1}\phi_j\pi_{j-1} - \alpha_j t\,\pi_j^2\right) \tag{7.25}$$
$$\phi_{j+1}\pi_j = \phi_j^2 + \beta_{j-1}\phi_j\pi_{j-1} - \alpha_j t\,\pi_j^2 \tag{7.26}$$
$$\pi_{j+1}^2 = \phi_{j+1}^2 + 2\beta_j\phi_{j+1}\pi_j + \beta_j^2\pi_j^2. \tag{7.27}$$

These recurrences are at the basis of the algorithm. If we define

$$r_j = \phi_j^2(A)r_0, \tag{7.28}$$
$$p_j = \pi_j^2(A)r_0, \tag{7.29}$$
$$q_j = \phi_{j+1}(A)\pi_j(A)r_0, \tag{7.30}$$

then the above recurrences for the polynomials translate into

$$r_{j+1} = r_j - \alpha_j A\left(2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A\,p_j\right), \tag{7.31}$$
$$q_j = r_j + \beta_{j-1}q_{j-1} - \alpha_j A\,p_j, \tag{7.32}$$
$$p_{j+1} = r_{j+1} + 2\beta_j q_j + \beta_j^2 p_j. \tag{7.33}$$

It is convenient to define the auxiliary vector

$$d_j = 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A p_j.$$

With this we obtain the following sequence of operations to compute the approximate solution, starting with $r_0 := b - Ax_0$, $p_0 := r_0$, $q_0 := 0$, $\beta_0 := 0$.

- $\alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*)$
- $d_j = 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A p_j$
- $q_j = r_j + \beta_{j-1}q_{j-1} - \alpha_j A p_j$
- $x_{j+1} = x_j + \alpha_j d_j$
- $r_{j+1} = r_j - \alpha_j A d_j$
- $\beta_j = (r_{j+1}, r_0^*)/(r_j, r_0^*)$
- $p_{j+1} = r_{j+1} + \beta_j(2q_j + \beta_j p_j)$.

A slight simplification to the algorithm can be made by using the auxiliary vector $u_j = r_j + \beta_{j-1}q_{j-1}$. This definition leads to the relations

$$d_j = u_j + q_j,$$

$$q_j = u_j - \alpha_j A p_j,$$
$$p_{j+1} = u_{j+1} + \beta_j (q_j + \beta_j p_j),$$

and as a result the vector d_j is no longer needed. The resulting algorithm is given below.

ALGORITHM 7.5: Conjugate Gradient Squared

1. *Compute* $r_0 := b - Ax_0$; r_0^* *arbitrary.*
2. *Set* $p_0 := u_0 := r_0$.
3. *For* $j = 0, 1, 2 \ldots,$ *until convergence Do:*
4. $\quad \alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $\quad q_j = u_j - \alpha_j A p_j$
6. $\quad x_{j+1} = x_j + \alpha_j (u_j + q_j)$
7. $\quad r_{j+1} = r_j - \alpha_j A(u_j + q_j)$
8. $\quad \beta_j = (r_{j+1}, r_0^*)/(r_j, r_0^*)$
9. $\quad u_{j+1} = r_{j+1} + \beta_j q_j$
10. $\quad p_{j+1} = u_{j+1} + \beta_j (q_j + \beta_j p_j)$
11. *EndDo*

Observe that there are no matrix-by-vector products with the transpose of A. Instead, two matrix-by-vector products with the matrix A are now performed at each step. In general, one should expect the resulting algorithm to converge twice as fast as BCG. Therefore, what has essentially been accomplished is to replace the matrix-by-vector products with A^T by more useful work.

The Conjugate Gradient Squared algorithm works quite well in many cases. However, one difficulty is that, since the polynomials are squared, rounding errors tend to be more damaging than in the standard BCG algorithm. In particular, very high variations of the residual vectors often cause the residual norms computed from the result of line 7 of the above algorithm to become inaccurate.

7.4.2 BICGSTAB

The CGS algorithm is based on squaring the residual polynomial, and, in cases of irregular convergence, this may lead to substantial build-up of rounding errors, or possibly even overflow. The Biconjugate Gradient Stabilized (BICGSTAB) algorithm is a variation of CGS which was developed to remedy this difficulty. Instead of seeking a method which delivers a residual vector of the form r'_j defined by (7.22), BICGSTAB produces iterates whose residual vectors are of the form

$$r'_j = \psi_j(A)\phi_j(A)r_0, \tag{7.34}$$

in which, as before, $\phi_j(t)$ is the residual polynomial associated with the BCG algorithm and $\psi_j(t)$ is a new polynomial which is defined recursively at each step with the goal of "stabilizing" or "smoothing" the convergence behavior of the original algorithm. Specifically, $\psi_j(t)$ is defined by the simple recurrence,

$$\psi_{j+1}(t) = (1 - \omega_j t)\psi_j(t) \tag{7.35}$$

in which the scalar ω_j is to be determined. The derivation of the appropriate recurrence relations is similar to that of CGS. Ignoring the scalar coefficients at first, we start with a relation for the residual polynomial $\psi_{j+1}\phi_{j+1}$. We immediately obtain

$$\psi_{j+1}\phi_{j+1} = (1-\omega_j t)\psi_j(t)\phi_{j+1} \tag{7.36}$$
$$= (1-\omega_j t)\ (\psi_j\phi_j - \alpha_j t\psi_j\pi_j) \tag{7.37}$$

which is updatable provided a recurrence relation is found for the products $\psi_j\pi_j$. For this, write

$$\psi_j\pi_j = \psi_j(\phi_j + \beta_{j-1}\pi_{j-1}) \tag{7.38}$$
$$= \psi_j\phi_j + \beta_{j-1}(1-\omega_{j-1}t)\psi_{j-1}\pi_{j-1}. \tag{7.39}$$

Define,

$$r_j = \phi_j(A)\psi_j(A)r_0,$$
$$p_j = \psi_j(A)\pi_j(A)r_0.$$

According to the above formulas, these vectors can be updated from a double recurrence provided the scalars α_j and β_j were computable. This recurrence is

$$r_{j+1} = (I-\omega_j A)(r_j - \alpha_j Ap_j) \tag{7.40}$$
$$p_{j+1} = r_{j+1} + \beta_j(I-\omega_j A)p_j.$$

Consider now the computation of the scalars needed in the recurrence. According to the original BCG algorithm, $\beta_j = \rho_{j+1}/\rho_j$ with

$$\rho_j = (\phi_j(A)r_0, \phi_j(A^T)r_0^*) = (\phi_j(A)^2 r_0, r_0^*)$$

Unfortunately, ρ_j is not computable from these formulas because none of the vectors $\phi_j(A)r_0$, $\phi_j(A^T)r_0^*$ or $\phi_j(A)^2 r_0$ is available. However, ρ_j can be related to the scalar

$$\tilde{\rho}_j = (\phi_j(A)r_0, \psi_j(A^T)r_0^*)$$

which is computable via

$$\tilde{\rho}_j = (\phi_j(A)r_0, \psi_j(A^T)r_0^*) = (\psi_j(A)\phi_j(A)r_0, r_0^*) = (r_j, r_0^*).$$

To relate the two scalars ρ_j and $\tilde{\rho}_j$, expand $\psi_j(A^T)r_0^*$ explicitly in the power basis, to obtain

$$\tilde{\rho}_j = \left(\phi_j(A)r_0,\ \eta_1^{(j)}(A^T)^j r_0^* + \eta_2^{(j)}(A^T)^{j-1}r_0^* + \dots\right).$$

Since $\phi_j(A)r_0$ is orthogonal to all vectors $(A^T)^k r_0^*$, with $k < j$, only the leading power is relevant in the expansion on the right side of the above inner product. In particular, if $\gamma_1^{(j)}$ is the leading coefficient for the polynomial $\phi_j(t)$, then

$$\tilde{\rho}_j = \left(\phi_j(A)r_0,\ \frac{\eta_1^{(j)}}{\gamma_1^{(j)}}\phi_j(A^T)r_0\right) = \frac{\eta_1^{(j)}}{\gamma_1^{(j)}}\,\rho_j.$$

When examining the recurrence relations for ϕ_{j+1} and ψ_{j+1}, leading coefficients for these polynomials are found to satisfy the relations

$$\eta_1^{(j+1)} = -\omega_j\eta_1^{(j)}, \quad \gamma_1^{(j+1)} = -\alpha_j\gamma_1^{(j)},$$

and as a result

$$\frac{\tilde{\rho}_{j+1}}{\tilde{\rho}_j} = \frac{\omega_j}{\alpha_j} \frac{\rho_{j+1}}{\rho_j}$$

which yields the following relation for β_j:

$$\beta_j = \frac{\tilde{\rho}_{j+1}}{\tilde{\rho}_j} \times \frac{\alpha_j}{\omega_j}. \tag{7.41}$$

Similarly, a simple recurrence formula for α_j can be derived. By definition,

$$\alpha_j = \frac{(\phi_j(A)r_0, \phi_j(A^T)r_0^*)}{(A\pi_j(A)r_0, \pi_j(A^T)r_0^*)}$$

and as in the previous case, the polynomials in the right sides of the inner products in both the numerator and denominator can be replaced by their leading terms. However, in this case the leading coefficients for $\phi_j(A^T)r_0^*$ and $\pi_j(A^T)r_0^*$ are identical, and therefore,

$$\begin{aligned}\alpha_j &= \frac{(\phi_j(A)r_0, \phi_j(A^T)r_0^*)}{(A\pi_j(A)r_0, \phi_j(A^T)r_0^*)} \\ &= \frac{(\phi_j(A)r_0, \psi_j(A^T)r_0^*)}{(A\pi_j(A)r_0, \psi_j(A^T)r_0^*)} \\ &= \frac{(\psi_j(A)\phi_j(A)r_0, r_0^*)}{(A\psi_j(A)\pi_j(A)r_0, r_0^*)}.\end{aligned}$$

Since $p_j = \psi_j(A)\pi_j(A)r_0$, this yields,

$$\alpha_j = \frac{\tilde{\rho}_j}{(Ap_j, r_0^*)}. \tag{7.42}$$

Next, the parameter ω_j must be defined. This can be thought of as an additional free parameter. One of the simplest choices, and perhaps the most natural, is to select ω_j to achieve a steepest descent step in the residual direction obtained before multiplying the residual vector by $(I - \omega_j A)$ in (7.40). In other words, ω_j is chosen to minimize the 2-norm of the vector $(I - \omega_j A)\psi_j(A)\phi_{j+1}(A)r_0$. Equation (7.40) can be rewritten as

$$r_{j+1} = (I - \omega_j A)s_j$$

in which

$$s_j \equiv r_j - \alpha_j A p_j.$$

Then the optimal value for ω_j is given by

$$\omega_j = \frac{(As_j, s_j)}{(As_j, As_j)}. \tag{7.43}$$

Finally, a formula is needed to update the approximate solution x_{j+1} from x_j. Equation (7.40) can be rewritten as

$$r_{j+1} = s_j - \omega_j A s_j = r_j - \alpha_j A p_j - \omega_j A s_j$$

which yields

$$x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j.$$

After putting these relations together, we obtain the final form of the BICGSTAB algorithm, due to van der Vorst [210].

ALGORITHM 7.6: BICGSTAB

1. *Compute* $r_0 := b - Ax_0$; r_0^* *arbitrary;*
2. $p_0 := r_0$.
3. *For* $j = 0, 1, \ldots$, *until convergence Do:*
4. $\quad \alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $\quad s_j := r_j - \alpha_j A p_j$
6. $\quad \omega_j := (As_j, s_j)/(As_j, As_j)$
7. $\quad x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8. $\quad r_{j+1} := s_j - \omega_j A s_j$
9. $\quad \beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $\quad p_{j+1} := r_{j+1} + \beta_j (p_j - \omega_j A p_j)$
11. *EndDo*

Example 7.2 Table 7.2 shows the results of applying the BICGSTAB algorithm with no preconditioning to three of the test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	96	2048	0.14E-02	0.77E-04
F3D	64	6407	0.49E-03	0.17E-03
ORS	208	5222	0.22E+00	0.68E-04

Table 7.2 *A test run of BICGSTAB with no preconditioning.*

See Example 6.1 for the meaning of the column headers in the table. The number of matrix-by-vector multiplications required to converge is larger than with BCG. Thus, using the number of matrix-by-vector products as a criterion, BCG is more expensive than BICGSTAB in all three examples. For problem 3, the number of steps for BCG exceeds the limit of 300. If the number of steps is used as a criterion, then the two methods come very close for the second problem [61 steps for BCG versus 64 for BICGSTAB]. However, BCG is slightly faster for Problem 1. Observe also that the total number of operations favors BICGSTAB. This illustrates the main weakness of BCG as well as QMR, namely, the matrix-by-vector products with the transpose are essentially wasted unless a dual system with A^T must be solved simultaneously.

7.4.3 TRANSPOSE-FREE QMR (TFQMR)

The Transpose-Free QMR algorithm of Freund [95] is derived from the CGS algorithm. Observe that x_j can be updated in two half-steps in line 6 of Algorithm 7.5, namely, $x_{j+\frac{1}{2}} = x_j + \alpha_j u_j$ and $x_{j+1} = x_{j+\frac{1}{2}} + \alpha_j q_j$. This is only natural since the actual update from one iterate to the next involves two matrix-by-vector multiplications, i.e., the degree of the residual polynomial is increased by two. In order to avoid indices that are multiples of $\frac{1}{2}$, it is convenient when describing TFQMR to double all subscripts in the CGS algorithm. With this change of notation, the main steps of the Algorithm 7.5 (CGS) become

$$\alpha_{2j} = (r_{2j}, r_0^*)/(Ap_{2j}, r_0^*) \tag{7.44}$$
$$q_{2j} = u_{2j} - \alpha_{2j}Ap_{2j} \tag{7.45}$$
$$x_{2j+2} = x_{2j} + \alpha_{2j}(u_{2j} + q_{2j}) \tag{7.46}$$
$$r_{2j+2} = r_{2j} - \alpha_{2j}A(u_{2j} + q_{2j}) \tag{7.47}$$
$$\beta_{2j} = (r_{2j+2}, r_0^*)/(r_{2j}, r_0^*) \tag{7.48}$$
$$u_{2j+2} = r_{2j+2} + \beta_{2j}q_{2j} \tag{7.49}$$
$$p_{2j+2} = u_{2j+2} + \beta_{2j}(q_{2j} + \beta p_{2j}). \tag{7.50}$$

The initialization is identical with that of Algorithm 7.5. The update of the approximate solution in (7.46) can now be split into the following two half-steps:

$$x_{2j+1} = x_{2j} + \alpha_{2j}u_{2j} \tag{7.51}$$
$$x_{2j+2} = x_{2j+1} + \alpha_{2j}q_{2j}. \tag{7.52}$$

This can be simplified by defining the vectors u_m for odd m as $u_{2j+1} = q_{2j}$. Similarly, the sequence of α_m is defined for odd values of m as $\alpha_{2j+1} = \alpha_{2j}$. In summary,

$$\text{for } m \text{ odd define:} \quad \begin{cases} u_m & \equiv & q_{m-1} \\ \alpha_m & \equiv & \alpha_{m-1} \end{cases}. \tag{7.53}$$

With these definitions, the relations (7.51–7.52) are translated into the single equation

$$x_m = x_{m-1} + \alpha_{m-1}u_{m-1},$$

which is valid whether m is even or odd. The intermediate iterates x_m, with m odd, which are now defined do not exist in the original CGS algorithm. For even values of m the sequence x_m represents the original sequence or iterates from the CGS algorithm. It is convenient to introduce the $N \times m$ matrix,

$$U_m = [u_0, \ldots, u_{m-1}]$$

and the m-dimensional vector

$$z_m = (\alpha_0, \alpha_1, \ldots, \alpha_{m-1})^T.$$

The general iterate x_m satisfies the relation

$$x_m = x_0 + U_m z_m \tag{7.54}$$
$$= x_{m-1} + \alpha_{m-1}u_{m-1}. \tag{7.55}$$

From the above equation, it is clear that the residual vectors r_m are related to the u-vectors by the relations

$$r_m = r_0 - AU_m z_m \tag{7.56}$$
$$= r_{m-1} - \alpha_{m-1}Au_{m-1}. \tag{7.57}$$

Next, a relation similar to the relation (6.5) seen for FOM and GMRES will be extracted using the matrix AU_m. As a result of (7.57), the following relation holds:

$$Au_i = \frac{1}{\alpha_i}(r_i - r_{i+1}).$$

Translated in matrix form, this relation becomes

$$AU_m = R_{m+1}\bar{B}_m \tag{7.58}$$

where

$$R_k = [r_0, r_1, \ldots, r_{k-1}] \tag{7.59}$$

and where $\bar{B}_m$ is the $(m+1) \times m$ matrix,

$$\bar{B}_m = \begin{pmatrix} 1 & 0 & \ldots & \ldots & 0 \\ -1 & 1 & & & \vdots \\ 0 & -1 & 1 & \ldots & \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & -1 & 1 \\ 0 & \ldots & & & -1 \end{pmatrix} \times \text{ diag } \left\{ \frac{1}{\alpha_0}, \frac{1}{\alpha_1}, \ldots \frac{1}{\alpha_{m-1}} \right\}. \tag{7.60}$$

The columns of R_{m+1} can be rescaled, for example, to make each of them have a 2-norm equal to one, by multiplying R_{m+1} to the right by a diagonal matrix. Let this diagonal matrix be the inverse of the matrix

$$\Delta_{m+1} = \text{diag } [\delta_0, \delta_1, \ldots, \delta_m].$$

Then,

$$AU_m = R_{m+1}\Delta_{m+1}^{-1}\Delta_{m+1}\bar{B}_m. \tag{7.61}$$

With this, equation (7.56) becomes

$$r_m = r_0 - AU_m z_m = R_{m+1}\left[e_1 - \bar{B}_m z_m\right] \tag{7.62}$$

$$= R_{m+1}\Delta_{m+1}^{-1}\left[\delta_0 e_1 - \Delta_{m+1}\bar{B}_m z_m\right]. \tag{7.63}$$

By analogy with the GMRES algorithm, define

$$\bar{H}_m \equiv \Delta_{m+1}\bar{B}_m.$$

Similarly, define H_m to be the matrix obtained from $\bar{H}_m$ by deleting its last row. It is easy to verify that the CGS iterates x_m (now defined for all integers $m = 0, 1, 2, \ldots$) satisfy the same definition as FOM, i.e.,

$$x_m = x_0 + U_m H_m^{-1}(\delta_0 e_1). \tag{7.64}$$

It is also possible to extract a GMRES-like solution from the relations (7.61) and (7.63), similar to DQGMRES. In order to minimize the residual norm over the Krylov subspace, the 2-norm of the right-hand side of (7.63) would have to be minimized, but this is not practical since the columns of $R_{m+1}\Delta_{m+1}^{-1}$ are not orthonormal as in GMRES. However, the 2-norm of $\delta_0 e_1 - \Delta_{m+1}\bar{B}_m z$ can be minimized over z, as was done for the QMR and DQGMRES algorithms.

This defines the TFQMR iterates theoretically. However, it is now necessary to find a formula for expressing the iterates in a progressive way. There are two ways to proceed.

The first follows DQGMRES closely, defining the least-squares solution progressively and exploiting the structure of the matrix R_m to obtain a formula for x_m from x_{m-1}. Because of the special structure of $\bar{H}_m$, this is equivalent to using the DQGMRES algorithm with $k = 1$. The second way to proceed exploits Lemma 6.1 seen in the previous chapter. This lemma, which was shown for the FOM/GMRES pair, is also valid for the CGS/TFQMR pair. There is no fundamental difference between the two situations. Thus, the TFQMR iterates satisfy the relation

$$x_m - x_{m-1} = c_m^2 \left(\tilde{x}_m - x_{m-1}\right) \tag{7.65}$$

where the tildes are now used to denote the CGS iterate. Setting

$$d_m \equiv \frac{1}{\alpha_{m-1}} \left(\tilde{x}_m - x_{m-1}\right) = \frac{1}{c_m^2 \alpha_{m-1}} \left(x_m - x_{m-1}\right) \tag{7.66}$$

$$\eta_m \equiv c_m^2 \alpha_{m-1},$$

the above expression for x_m becomes

$$x_m = x_{m-1} + \eta_m d_m. \tag{7.67}$$

Now observe from (7.55) that the CGS iterates $\tilde{x}_m$ satisfy the relation

$$\tilde{x}_m = \tilde{x}_{m-1} + \alpha_{m-1} u_{m-1}. \tag{7.68}$$

From the above equations, a recurrence relation from d_m can be extracted. The definition of d_m and the above relations yield

$$\begin{aligned} d_m &= \frac{1}{\alpha_{m-1}} \left(\tilde{x}_m - \tilde{x}_{m-1} + \tilde{x}_{m-1} - x_{m-1}\right) \\ &= u_{m-1} + \frac{1}{\alpha_{m-1}} \left(\tilde{x}_{m-1} - x_{m-2} - (x_{m-1} - x_{m-2})\right) \\ &= u_{m-1} + \frac{1 - c_{m-1}^2}{\alpha_{m-1}} \left(\tilde{x}_{m-1} - x_{m-2}\right). \end{aligned}$$

Therefore,

$$d_m = u_{m-1} + \frac{(1 - c_{m-1}^2)\eta_{m-1}}{c_{m-1}^2 \alpha_{m-1}} d_{m-1}.$$

The term $(1 - c_{m-1}^2)/c_{m-1}^2$ is the squared tangent of the angle used in the $(m-1) - st$ rotation. This tangent will be denoted by θ_{m-1}, and we have

$$\theta_m = \frac{s_m}{c_m}, \quad c_m^2 = \frac{1}{1 + \theta_m^2}, \quad d_{m+1} = u_m + \frac{\theta_m^2 \eta_m}{\alpha_m} d_m.$$

The angle used in the m-th rotation, or equivalently c_m, can be obtained by examining the matrix $\bar{H}_m$:

$$\bar{H}_m = \begin{pmatrix} \delta_0 & 0 & \dots & \dots & 0 \\ -\delta_1 & \delta_1 & & & \vdots \\ 0 & -\delta_2 & \delta_2 & \dots & \\ \vdots & & \ddots & \ddots & \vdots \\ \vdots & & & -\delta_m & \delta_m \\ 0 & \dots & & & -\delta_{m+1} \end{pmatrix} \times \text{ diag } \left\{ \frac{1}{\alpha_i} \right\}_{i=0,\dots,m-1} . \tag{7.69}$$

The diagonal matrix in the right-hand side scales the columns of the matrix. It is easy to see that it has no effect on the determination of the rotations. Ignoring this scaling, the above matrix becomes, after j rotations,

$$\begin{pmatrix} \star & \star & & & & & \\ & \star & \star & & & & \\ & & \ddots & \ddots & & & \\ & & & \tau_j & 0 & & \\ & & & -\delta_{j+1} & \delta_{j+1} & & \\ & & & & \ddots & \ddots & \\ & & & & & -\delta_m & \delta_m \\ & & & & & & -\delta_{m+1} \end{pmatrix}.$$

The next rotation is then determined by,

$$s_{j+1} = \frac{-\delta_{j+1}}{\sqrt{\tau_j^2 + \delta_{j+1}^2}}, \quad c_{j+1} = \frac{\tau_j}{\sqrt{\tau_j^2 + \delta_{j+1}^2}}, \quad \theta_{j+1} = \frac{-\delta_{j+1}}{\tau_j}.$$

In addition, after this rotation is applied to the above matrix, the diagonal element δ_{j+1} which is in position $(j+1, j+1)$ is transformed into

$$\tau_{j+1} = \delta_{j+1} \times c_{j+1} = \frac{\tau_j \delta_{j+1}}{\sqrt{\tau_j^2 + \delta_{j+1}^2}} = -\tau_j s_{j+1} = -\tau_j \theta_{j+1} c_{j+1}. \tag{7.70}$$

The above relations enable us to update the direction d_m and the required quantities c_m and η_m. Since only the squares of these scalars are invoked in the update of the direction d_{m+1}, a recurrence for their absolute values is sufficient. This gives the following recurrences which will be used in the algorithm:

$$\begin{aligned} d_{m+1} &= u_m + (\theta_m^2/\alpha_m)\eta_m d_m \\ \theta_{m+1} &= \delta_{m+1}/\tau_m \\ c_{m+1} &= \left(1 + \theta_{m+1}^2\right)^{-\frac{1}{2}} \\ \tau_{m+1} &= \tau_m \theta_{m+1} c_{m+1} \\ \eta_{m+1} &= c_{m+1}^2 \alpha_m. \end{aligned}$$

Before writing down the algorithm, a few relations must be exploited. Since the vectors r_m are no longer the actual residuals in the algorithm, we change the notation to w_m. These residual vectors can be updated by the formula

$$w_m = w_{m-1} - \alpha_{m-1} A u_{m-1}.$$

The vectors Au_i can be used to update the vectors

$$v_{2j} \equiv Ap_{2j}$$

which are needed in the CGS algorithm. Multiplying (7.50) by A results in

$$Ap_{2j} = Au_{2j} + \beta_{2j-2}(Aq_{2j-2} + \beta_j Ap_{2j-2})$$

which, upon substituting the relation

$$q_{2j} = u_{2j+1}$$

translates into

$$v_{2j} = Au_{2j} + \beta_{2j-2}(Au_{2j-1} + \beta_{2j-2}v_{2j-2}).$$

Also, observe that the recurrences in (7.45) and (7.49) for q_{2j} and u_{2j+2}, respectively, become

$$u_{2j+1} = u_{2j} - \alpha_{2j}v_{2j}$$
$$u_{2j+2} = w_{2j+2} + \beta_{2j}u_{2j+1}.$$

The first equation should be used to compute u_{m+1} when m is even, and the second when m is odd. In the following algorithm, the normalization $\delta_m = \|w_m\|_2$, which normalize each column of R_m to have 2-norm unity, is used.

ALGORITHM 7.7: Transpose-Free QMR (TFQMR)

Compute $w_0 = u_0 = r_0 = b - Ax_0$, $v_0 = Au_0$, $d_0 = 0$;
$\tau_0 = \|r_0\|_2$, $\theta_0 = \eta_0 = 0$.
Choose r_0^* *such that* $\rho_0 \equiv (r_0^*, r_0) \neq 0$.
For $m = 0, 1, 2, \ldots,$ *until convergence Do:*
  If m *is even then*
    $\alpha_{m+1} = \alpha_m = \rho_m/(v_m, r_0^*)$
    $u_{m+1} = u_m - \alpha_m v_m$
  EndIf
  $w_{m+1} = w_m - \alpha_m Au_m$
  $d_{m+1} = u_m + (\theta_m^2/\alpha_m)\eta_m d_m$
  $\theta_{m+1} = \|w_{m+1}\|_2/\tau_m$; $c_{m+1} = \left(1 + \theta_{m+1}^2\right)^{-\frac{1}{2}}$
  $\tau_{m+1} = \tau_m\theta_{m+1}c_{m+1}$; $\eta_{m+1} = c_{m+1}^2\alpha_m$
  $x_{m+1} = x_m + \eta_{m+1}d_{m+1}$
  If m *is odd then*
    $\rho_{m+1} = (r_{m+1}, r_0^*)$; $\beta_{m-1} = \rho_{m+1}/\rho_{m-1}$
    $u_{m+1} = w_{m+1} + \beta_{m-1}u_m$
    $v_{m+1} = Au_{m+1} + \beta_{m-1}(Au_m + \beta_{m-1}v_{m-1})$
  EndIf
EndDo

Notice that the quantities in the odd m loop are only defined for even values of m. The residual norm of the approximate solution x_m is not available from the above algorithm as it is described. However, good estimates can be obtained using similar strategies to those used for DQGMRES. Referring to GMRES, an interesting observation is that the recurrence (6.40) is identical with the recurrence of the scalars τ_j's. In addition, these two sequences start with the same values, δ_0 for the τ's and β for the γ's. Therefore,

$$\gamma_{m+1} = \tau_m.$$

Recall that γ_{m+1} is the residual for the $(m+1) \times m$ least-squares problem

$$\min_z \|\delta_0 e_1 - \bar{H}_m z\|_2.$$

Hence, a relation similar to that for DQGMRES holds, namely,

$$\|b - Ax_m\| \leq \sqrt{m+1}\tau_m. \tag{7.71}$$

This provides a readily computable estimate of the residual norm. Another point that should be made is that it is possible to use the scalars s_m, c_m in the recurrence instead of the pair c_m, θ_m, as was done above. In this case, the proper recurrences are

$$\begin{aligned}
d_{m+1} &= u_m + (s_m^2/\alpha_m)\alpha_{m-1} d_m \\
s_{m+1} &= \delta_{m+1}/\sqrt{\tau_m^2 + \delta_{m+1}^2} \\
c_{m+1} &= \tau_m/\sqrt{\tau_m^2 + \delta_{m+1}^2} \\
\tau_{m+1} &= \tau_m s_{m+1} \\
\eta_{m+1} &= c_{m+1}^2 \alpha_m.
\end{aligned}$$

Example 7.3 Table 7.3 shows the results when TFQMR algorithm without preconditioning is applied to three of the test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	112	2736	0.46E-04	0.68E-04
F3D	78	8772	0.52E-04	0.61E-03
ORS	252	7107	0.38E-01	0.19E-03

Table 7.3 *A test run of TFQMR with no preconditioning.*

See Example 6.1 for the meaning of the column headers in the table. The number of steps is slightly higher than that of BICGSTAB. Comparing with BCG, we note that each step of BCG requires two matrix-by-vector products compared with one for TFQMR and BICGSTAB. Thus, using the number of matrix-by-vector products as a criterion, BCG is more expensive than TFQMR in all cases, as is shown in the "Iters" columns. If the number of steps is used as a criterion, then BCG is just slightly better for Problems 1 and 2. A comparison is not possible for Problem 3, since the number of matrix-by-vector products required for convergence exceeds the limit of 300. In general, the number of steps required for convergence is similar for BICGSTAB and TFQMR. A comparison with the methods seen in the previous chapter indicates that in many cases, GMRES will be faster if the problem is well conditioned, resulting in a moderate number of steps required to converge. If many steps (say, in the hundreds) are required, then BICGSTAB and TFQMR may perform better. If memory is not an issue, GMRES or DQGMRES, with a large number of directions, is often the most reliable choice. The issue then is one of trading ribustness for memory usage. In general, a sound strategy is to focus on finding a good preconditioner rather than the best accelerator.

EXERCISES

1. Consider the following modification of the Lanczos algorithm, Algorithm 7.1. We replace line 6 by

$$\hat{w}_{j+1} = A^T w_j - \sum_{i=1}^{j} h_{ij} w_i$$

where the scalars h_{ij} are arbitrary. Lines 5 and 7 through 10 remain the same but line 4 in which α_j is computed must be changed.

a. Show how to modify line 4 to ensure that the vector $\hat{v}_{j+1}$ is orthogonal against the vectors w_i, for $i = 1, \ldots, j$.

b. Prove that the vectors v_i's and the matrix T_m do not depend on the choice of the h_{ij}'s.

c. Consider the simplest possible choice, namely, $h_{ij} \equiv 0$ for all i, j. What are the advantages and potential difficulties with this choice?

2. Assume that the Lanczos algorithm does not break down before step m, i.e., that it is possible to generate $v_1, \ldots v_{m+1}$. Show that V_{m+1} and W_{m+1} are both of full rank.

3. Develop a modified version of the non-Hermitian Lanczos algorithm that produces a sequence of vectors v_i, w_i such that each v_i is orthogonal to every w_j with $j \neq i$ and $\|v_i\|_2 = \|w_i\|_2 = 1$ for all i. What does the projected problem become?

4. Develop a version of the non-Hermitian Lanczos algorithm that produces a sequence of vectors v_i, w_i which satisfy

$$(v_i, w_j) = \pm\delta_{ij},$$

but such that the matrix T_m is Hermitian tridiagonal. What does the projected problem become in this situation?

5. Using the notation of Section 7.1.2 prove that $q_{j+k}(t) = t^k p_j(t)$ is orthogonal to the polynomials $p_1, p_2, \ldots, p_{j-k}$, assuming that $k \leq j$. Show that if q_{j+k} is orthogonalized against $p_1, p_2, \ldots, p_{j-k}$, the result would be orthogonal to all polynomials of degree $< j+k$. Derive a general Look-Ahead non-Hermitian Lanczos procedure based on this observation.

6. Consider the matrices $V_m = [v_1, \ldots, v_m]$ and $W_m = [w_1, \ldots, w_m]$ obtained from the Lanczos biorthogonalization algorithm. (a) What are the matrix representations of the (oblique) projector onto $\mathcal{K}_m(A, v_1)$ orthogonal to the subspace $\mathcal{K}_m(A^T, w_1)$, and the projector onto $\mathcal{K}_m(A^T, w_1)$ orthogonally to the subspace $\mathcal{K}_m(A, v_1)$? (b) Express a general condition for the existence of an oblique projector onto K, orthogonal to L. (c) How can this condition be interpreted using the Lanczos vectors and the Lanczos algorithm?

7. Show a three-term recurrence satisfied by the residual vectors r_j of the BCG algorithm. Include the first two iterates to start the recurrence. Similarly, establish a three-term recurrence for the conjugate direction vectors p_j in BCG.

8. Let $\phi_j(t)$ and $\pi_j(t)$ be the residual polynomial and the conjugate direction polynomial, respectively, for the BCG algorithm, as defined in Section 7.4.1. Let $\psi_j(t)$ be any other polynomial sequence which is defined from the recurrence

$$\psi_0(t) = 1, \quad \psi_1(t) = (1 - \xi_0 t)\psi_0(t)$$
$$\psi_{j+1}(t) = (1 + \eta_j - \xi_j t)\psi_j(t) - \eta_j \psi_{j-1}(t)$$

a. Show that the polynomials ψ_j are consistent, i.e., $\psi_j(0) = 1$ for all $j \ge 0$.

b. Show the following relations

$$\psi_{j+1}\phi_{j+1} = \psi_j\phi_{j+1} - \eta_j(\psi_{j-1} - \psi_j)\phi_{j+1} - \xi_j t\psi_j\phi_{j+1}$$
$$\psi_j\phi_{j+1} = \psi_j\phi_j - \alpha_j t\psi_j\pi_j$$
$$(\psi_{j-1} - \psi_j)\phi_{j+1} = \psi_{j-1}\phi_j - \psi_j\phi_{j+1} - \alpha_j t\psi_{j-1}\pi_j$$
$$\psi_{j+1}\pi_{j+1} = \psi_{j+1}\phi_{j+1} - \beta_j\eta_j\psi_{j-1}\pi_j + \beta_j(1 + \eta_j)\psi_j\pi_j - \beta_j\xi_j t\psi_j\pi_j$$
$$\psi_j\pi_{j+1} = \psi_j\phi_{j+1} + \beta_j\psi_j\pi_j.$$

c. Defining,

$$t_j = \psi_j(A)\phi_{j+1}(A)r_0, \quad y_j = (\psi_{j-1}(A) - \psi_j(A))\phi_{j+1}(A)r_0,$$
$$p_j = \psi_j(A)\pi_j(A)r_0, \qquad s_j = \psi_{j-1}(A)\pi_j(A)r_0$$

show how the recurrence relations of the previous question translate for these vectors.

d. Find a formula that allows one to update the approximation x_{j+1} from the vectors x_{j-1}, x_j and t_j, p_j, y_j, s_j defined above.

e. Proceeding as in BICGSTAB, find formulas for generating the BCG coefficients α_j and β_j from the vectors defined in the previous question.

9. Prove the expression (7.64) for the CGS approximation defined by (7.54–7.55). Is the relation valid for any choice of scaling Δ_{m+1}?

10. Prove that the vectors r_j and r_i^* produced by the BCG algorithm are orthogonal to each other when $i \neq j$, while the vectors p_i and p_j^* are A-orthogonal, i.e., $(Ap_j, p_i^*) = 0$ for $i \neq j$.

11. The purpose of this exercise is to develop block variants of the Lanczos algorithm. Consider a two-sided analogue of the Block-Arnoldi algorithm, in its variant of Algorithm 6.23. Formally, the general step that defines the biorthogonalization process, for $j \ge p$, is as follows:

1. Orthogonalize Av_{j-p+1} versus $w_1, w_2, \ldots, w_j$ (by subtracting a linear combination of $v_1, \ldots, v_j$ from Av_{j-p+1}). Call v the resulting vector.
2. Orthogonalize $A^T w_{j-p+1}$ versus $v_1, v_2, \ldots, v_j$ (by subtracting a linear combination of $w_1, \ldots, w_j$ from $A^T w_{j-p+1}$). Call w the resulting vector.
3. Normalize the two vectors v and w so that $(v, w) = 1$ to get v_{j+1} and w_{j+1}.

Here, p is the block size and it is assumed that the initial blocks are biorthogonal: $(v_i, w_j) = \delta_{ij}$ for $i, j \le p$.

a. Show that Av_{j-p+1} needs only to be orthogonalized against the $2p$ previous w_i's instead of all of them. Similarly, $A^T w_{j-p+1}$ must be orthogonalized only against the $2p$ previous v_i's.

b. Write down the algorithm completely. Show the orthogonality relations satisfied by the vectors v_i and w_j. Show also relations similar to (7.3) and (7.4).

c. We now assume that the two sets of vectors v_i and w_j have different block sizes. Call q the block-size for the w's. Line 2 of the above formal algorithm is changed into:

2a. Orthogonalize $A^T w_{j-q+1}$ versus $v_1, v_2, \ldots, v_j$ $(\cdots)$. Call w the resulting vector.

and the rest remains unchanged. The initial vectors are again biorthogonal: $(v_i, w_j) = \delta_{ij}$ for $i \leq p$ and $j \leq q$. Show that now Av_{j-p+1} needs only to be orthogonalized against the $q+p$ previous w_i's instead of all of them. Show a simlar result for the w_j's.

d. Show how a block version of BCG and QMR can be developed based on the algorithm resulting from question (c).

NOTES AND REFERENCES. At the time of this writing there is still much activity devoted to the class of methods covered in this chapter. Two of the starting points in this direction are the papers by Sonneveld [201] and Freund and Nachtigal [97]. The more recent BICGSTAB [210] has been developed to cure some of the numerical problems that plague CGS. There have been a few recent additions and variations to the basic BCG, BICGSTAB, and TFQMR techniques; see [42, 47, 113, 114, 192], among many others. A number of variations have been developed to cope with the breakdown of the underlying Lanczos or BCG algorithm; see, for example, [41, 20, 96, 192, 231]. Finally, block methods have also been developed [5].

Many of the Lanczos-type algorithms developed for solving linear systems are rooted in the theory of orthogonal polynomials and Padé approximation. Lanczos himself certainly used this viewpoint when he wrote his breakthrough papers [140, 142] in the early 1950s. The monogram by Brezinski [38] gives an excellent coverage of the intimate relations between approximation theory and the Lanczos-type algorithms. Freund [94] establishes these relations for quasi-minimal residual methods. A few optimality properties for the class of methods presented in this chapter can be proved using a variable metric, i.e., an inner product which is different at each step [21]. A recent survey by Weiss [224] presents a framework for Krylov subspace methods explaining some of these optimality properties and the interrelationships between Krylov subspace methods. Several authors discuss a class of techniques known as residual smoothing; see for example [191, 234, 224, 40]. These techniques can be applied to any iterative sequence x_k to build a new sequence of iterates y_k by combining y_{k-1} with the difference $x_k - y_{k-1}$. A remarkable result shown by Zhou and Walker [234] is that the iterates of the QMR algorithm can be obtained from those of the BCG as a particular case of residual smoothing.

A number of projection-type methods on Krylov subspaces, other than those seen in this chapter and the previous one are described in [1]. The group of rank-k update methods discussed by Eirola and Nevanlinna [79] and Deufflhard et al. [70] is closely related to Krylov subspace methods. In fact, GMRES can be viewed as a particular example of these methods. Also of interest and not covered in this book are the *vector extrapolation* techniques which are discussed, for example, in the books Brezinski [38], Brezinski and Radivo Zaglia [39] and the articles [199] and [126]. Connections between these methods and Krylov subspace methods, have been uncovered, and are discussed by Brezinski [38] and Sidi [195]. ■