Chapter 5

BiCG and Related Methods

Since the GMRES method for non-Hermitian problems requires increasing amounts of work and storage per iteration, it is important to consider other methods with a fixed amount of work and storage, even though they will require more iterations to reduce the 2-norm of the residual to a given level. Several such methods have already been presented, e.g., simple iteration, Orthomin(j), and GMRES(j). All have the possibility of failure: simple iteration may diverge, Orthomin(j) may encounter an undefined coefficient, and both Orthomin(j) and GMRES(j) may stagnate (cease to reduce the residual norm).

In this chapter we consider several other iteration methods that, in practice, have often been found to perform better than the previously listed algorithms. These algorithms also have the possibility of failure, although that can be alleviated through the use of *look-ahead*. With look-ahead, however, the methods no longer require a fixed amount of work and storage per iteration. The work and storage grows with the number of look-ahead steps, just as it grows in GMRES. Unfortunately, there are no a priori theoretical estimates comparing the error at each step of these methods to that of the optimal GMRES approximation, unless an unlimited number of look-ahead steps are allowed. This problem is discussed further in Chapter 6.

5.1. The Two-Sided Lanczos Algorithm.

When the matrix A is Hermitian, the Gram–Schmidt procedure for constructing an orthonormal basis for the Krylov space of A reduces to a 3-term recurrence. Unfortunately, this is not the case when A is non-Hermitian. One can, however, use a *pair* of 3-term recurrences, one involving A and the other involving A^H, to construct *biorthogonal* bases for the Krylov spaces corresponding to A and A^H. Let $\mathcal{K}_k(B, v)$ denote the Krylov space span$\{v, Bv, \ldots, B^{k-1}v\}$. Then one constructs two sets of vectors—$v_1, \ldots, v_k \in \mathcal{K}_k(A, r_0)$ and $w_1, \ldots, w_k \in \mathcal{K}_k(A^H, \hat{r}_0)$—such that $\langle v_i, w_j \rangle = 0$ for $i \neq j$. This procedure is called the *two-sided Lanczos algorithm.*

```
**Two-Sided Lanczos Algorithm** (without look-ahead).

Given $r_0$ and $\hat{r}_0$ with $\langle r_0, \hat{r}_0 \rangle \neq 0$, set $v_1 = r_0/\|r_0\|$ and $w_1 = \hat{r}_0/\langle \hat{r}_0, v_1 \rangle$.
Set $\beta_0 = \gamma_0 = 0$ and $v_0 = w_0 = 0$. For $j = 1, 2, \ldots,$

Compute $Av_j$ and $A^H w_j$.

Set $\alpha_j = \langle Av_j, w_j \rangle$.

Compute $\tilde{v}_{j+1} = Av_j - \alpha_j v_j - \beta_{j-1} v_{j-1}$ and $\tilde{w}_{j+1} = A^H w_j - \bar{\alpha}_j w_j - \gamma_{j-1} w_{j-1}$.

Set $\gamma_j = \|\tilde{v}_{j+1}\|$ and $v_{j+1} = \tilde{v}_{j+1}/\gamma_j$.

Set $\beta_j = \langle v_{j+1}, \tilde{w}_{j+1} \rangle$ and $w_{j+1} = \tilde{w}_{j+1}/\bar{\beta}_j$.
```

Here we have given the non-Hermitian Lanczos formulation that scales so that each basis vector v_j has norm 1 and $\langle w_j, v_j \rangle = 1$. The scaling of the basis vectors can be chosen differently. Another formulation of the algorithm uses the ordinary transpose A^T, instead of A^H.

Letting V_k be the matrix with columns $v_1, \ldots, v_k$ and W_k be the matrix with columns $w_1, \ldots, w_k$, this pair of recurrences can be written in matrix form as

$$AV_k = V_k T_k + \gamma_k v_{k+1} \xi_k^T = V_{k+1} T_{k+1,k}, \tag{5.1}$$

$$A^H W_k = W_k T_k^H + \bar{\beta}_k w_{k+1} \xi_k^T = W_{k+1} \hat{T}_{k+1,k}, \tag{5.2}$$

where T_k is the k-by-k tridiagonal matrix of recurrence coefficients

$$T_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \gamma_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \gamma_{k-1} & \alpha_k \end{pmatrix}.$$

The $k+1$-by-k matrices $T_{k+1,k}$ and $\hat{T}_{k+1,k}$ have T_k and T_k^H, respectively, as their top k-by-k blocks, and their last rows consist of zeros except for the last entry, which is γ_k and $\bar{\beta}_k$, respectively. The biorthogonality condition implies that

$$V_k^H W_k = I. \tag{5.3}$$

Note that if $A = A^H$ and $\hat{r}_0 = r_0$, then the two-sided Lanczos recurrence reduces to the ordinary Hermitian Lanczos process.

THEOREM 5.1.1. *If the two-sided Lanczos vectors are defined at steps* $1, \ldots, k+1$, *that is, if* $\langle v_j, w_j \rangle \neq 0$, $j = 1, \ldots, k+1$, *then*

$$\langle v_i, w_j \rangle = 0 \quad \forall i \neq j, \quad i, j \leq k+1. \tag{5.4}$$

Proof. Assume that (5.4) holds for $i, j \leq k$. The choice of the coefficients β_j and γ_j assures that for all j, $\langle w_j, v_j \rangle = 1$ and $\|v_j\| = 1$. By construction of the coefficient α_k, we have, using the induction hypothesis,

$$\langle \tilde{v}_{k+1}, w_k \rangle = \langle Av_k, w_k \rangle - \alpha_k = 0,$$

$$\langle \tilde{w}_{k+1}, v_k \rangle = \langle A^H w_k, v_k \rangle - \bar{\alpha}_k = 0.$$

Using the recurrences for $\tilde{v}_{k+1}$ and $\tilde{w}_k$ along with the induction hypothesis, we have

$$\langle \tilde{v}_{k+1}, w_{k-1} \rangle = \langle Av_k, w_{k-1} \rangle - \beta_{k-1} = \langle v_k, A^H w_{k-1} \rangle - \beta_{k-1} = \langle v_k, \tilde{w}_k \rangle - \beta_{k-1} = 0,$$

and, similarly, it follows that $\langle \tilde{w}_{k+1}, v_{k-1} \rangle = 0$. Finally, for $j < k-1$, we have

$$\langle \tilde{v}_{k+1}, w_j \rangle = \langle Av_k, w_j \rangle = \langle v_k, A^H w_j \rangle = \langle v_k, \tilde{w}_{j+1} \rangle = 0,$$

and, similarly, it is seen that $\langle \tilde{w}_{k+1}, v_j \rangle = 0$. Since v_{k+1} and w_{k+1} are just multiples of $\tilde{v}_{k+1}$ and $\tilde{w}_{k+1}$, the result (5.4) is proved. □

The vectors generated by the two-sided Lanczos process can become undefined in two different situations. First, if $\tilde{v}_{j+1} = 0$ or $\tilde{w}_{j+1} = 0$, then the Lanczos algorithm has found an invariant subspace. If $\tilde{v}_{j+1} = 0$, then the right Lanczos vectors $v_1, \ldots, v_j$ form an A-invariant subspace. If $\tilde{w}_{j+1} = 0$, then the left Lanczos vectors $w_1, \ldots, w_j$ form an A^H-invariant subspace. This is referred to as *regular termination.*

The second case, referred to as *serious breakdown*, occurs when $\langle \tilde{v}_{j+1}, \tilde{w}_{j+1} \rangle = 0$ but neither $\tilde{v}_{j+1} = 0$ nor $\tilde{w}_{j+1} = 0$. In this case, nonzero vectors $v_{j+1} \in \mathcal{K}_{j+1}(A, r_0)$ and $w_{j+1} \in \mathcal{K}_{j+1}(A^H, \hat{r}_0)$ satisfying $\langle v_{j+1}, w_i \rangle = \langle w_{j+1}, v_i \rangle = 0$ for all $i \leq j$ simply do not exist. Note, however, that while such vectors may not exist at step $j+1$, at some later step $j+\ell$ there may be nonzero vectors $v_{j+\ell} \in \mathcal{K}_{j+\ell}(A, r_0)$ and $w_{j+\ell} \in \mathcal{K}_{j+\ell}(A^H, \hat{r}_0)$ such that $v_{j+\ell}$ is orthogonal to $\mathcal{K}_{j+\ell-1}(A^H, \hat{r}_0)$ and $w_{j+\ell}$ is orthogonal to $\mathcal{K}_{j+\ell-1}(A, r_0)$. Procedures that simply skip steps at which the Lanczos vectors are undefined and construct the Lanczos vectors for the steps at which they are defined are referred to as *look-ahead* Lanczos methods. We will not discuss look-ahead Lanczos methods here but refer the reader to [101, 113, 20] for details.

5.2. The Biconjugate Gradient Algorithm.

Let us assume for the moment that the Lanczos recurrence does not break down. (From here on, we will refer to the two-sided Lanczos algorithm as simply the Lanczos algorithm, since it is the only Lanczos algorithm for non-Hermitian matrices.) Then the basis vectors might be used to approximate the solution of a linear system, as was done in the Hermitian case. If x_k is taken to be of the form

$$x_k = x_0 + V_k y_k, \tag{5.5}$$

then there are several natural ways to choose the vector y_k. One choice is to force $r_k = r_0 - AV_k y_k$ to be orthogonal to $w_1, \dots, w_k$. This leads to the equation

$$W_k^H r_k = W_k^H r_0 - W_k^H A V_k y_k = 0.$$

It follows from (5.1) and the biorthogonality condition (5.3) that $W_k^H A V_k = T_k$ and that $W_k^H r_0 = \beta \xi_1$, $\beta = \|r_0\|$, so the equation for y_k becomes

$$T_k y_k = \beta \xi_1. \tag{5.6}$$

When A is Hermitian and $\hat{r}_0 = r_0$, this reduces to the CG algorithm, as was described in section 2.5. If T_k is singular, this equation may have no solution. In this case, there is no approximation x_k of the form (5.5) for which $W_k^H r_k = 0$. Note that this type of failure is *different* from the possible breakdown of the underlying Lanczos recurrence. The Lanczos vectors may be well defined, but if the tridiagonal matrix is singular or near singular, an algorithm that attempts to solve this linear system will have difficulty. An algorithm that attempts to generate approximations of the form (5.5), where y_k satisfies (5.6), is called the *biconjugate gradient* (BiCG) algorithm.

The BiCG algorithm can be derived from the non-Hermitian Lanczos process and the LDU-factorization of the tridiagonal matrix T_k in much the same way that the CG algorithm was derived from the Hermitian Lanczos process in section 2.5. As for CG, failure of the BiCG method occurs when a singular tridiagonal matrix is encountered, and, with the standard implementation of the algorithm, one cannot recover from a singular tridiagonal matrix at one step, even if later tridiagonal matrices are well conditioned. We will not carry out this derivation since it is essentially the same as that in section 2.5 but will simply state the algorithm as follows.

Biconjugate Gradient Algorithm (BiCG).

Given x_0, compute $r_0 = b - Ax_0$, and set $p_0 = r_0$. Choose $\hat{r}_0$ such that $\langle r_0, \hat{r}_0 \rangle \neq 0$, and set $\hat{p}_0 = \hat{r}_0$. For $k = 1, 2, \dots$

Set $x_k = x_{k-1} + a_{k-1} p_{k-1}$, where $a_{k-1} = \frac{\langle r_{k-1}, \hat{r}_{k-1} \rangle}{\langle A p_{k-1}, \hat{p}_{k-1} \rangle}$.

Compute $r_k = r_{k-1} - a_{k-1} A p_{k-1}$ and $\hat{r}_k = \hat{r}_{k-1} - \bar{a}_{k-1} A^H \hat{p}_{k-1}$.

Set $p_k = r_k + b_{k-1} p_{k-1}$ and $\hat{p}_k = \hat{r}_k + \bar{b}_{k-1} \hat{p}_{k-1}$, where $b_{k-1} = \frac{\langle r_k, \hat{r}_k \rangle}{\langle r_{k-1}, \hat{r}_{k-1} \rangle}$.

A better implementation can be derived from that of the QMR algorithm described in section 5.3.

5.3. The Quasi-Minimal Residual Algorithm.

In the *quasi-minimal residual* (QMR) algorithm, the approximate solution x_k is again taken to be of the form (5.5), but now y_k is chosen to minimize a quantity that is closely related to the 2-norm of the residual. Since $r_k = r_0 - AV_k y_k$, we can write

$$r_k = V_{k+1}(\beta\xi_1 - T_{k+1,k}y_k), \tag{5.7}$$

so the norm of r_k satisfies

$$\|r_k\| \le \|V_{k+1}\| \cdot \|\beta\xi_1 - T_{k+1,k}y_k\|. \tag{5.8}$$

Since the columns of V_{k+1} are not orthogonal, it would be difficult to choose y_k to minimize $\|r_k\|$, but y_k can easily be chosen to minimize the second factor in (5.8). Since the columns of V_{k+1} each have norm one, the first factor in (5.8) satisfies $\|V_{k+1}\| \le \sqrt{k+1}$. In the QMR method, y_k solves the least squares problem

$$\min_y \|\beta\xi_1 - T_{k+1,k}y\|, \tag{5.9}$$

which always has a solution, even if the tridiagonal matrix T_k is singular. Thus the QMR iterates are defined provided that the underlying Lanczos recurrence does not break down.

The norm of the QMR residual can be related to that of the optimal GMRES residual as follows.

THEOREM 5.3.1 (Nachtigal [101]). *If r_k^G denotes the GMRES residual at step k and r_k^Q denotes the QMR residual at step k, then*

$$\|r_k^Q\| \le \kappa(V_{k+1})\ \|r_k^G\|, \tag{5.10}$$

where V_{k+1} is the matrix of basis vectors for the space $\mathcal{K}_{k+1}(A, r_0)$ constructed by the Lanczos algorithm and $\kappa(\cdot)$ denotes the condition number.

Proof. The GMRES residual is also of the form (5.7), but the vector y_k^G is chosen to minimize the 2-norm of the GMRES residual. It follows that

$$\|r_k^G\| \ge \sigma_{min}(V_{k+1})\ \min_y \|\beta\xi_1 - T_{k+1,k}y\|,$$

where $\sigma_{min}(V_{k+1})$ is the smallest singular value. Combining this with inequality (5.8) for the QMR residual gives the desired result (5.10). □

Unfortunately, the condition number of the basis vectors V_{k+1} produced by the non-Hermitian Lanczos algorithm cannot be bounded a priori. This matrix may be ill conditioned, even if the Lanczos vectors are well defined. If one could devise a short recurrence that would generate *well-conditioned* basis vectors, then one could use the quasi-minimization strategy (5.9) to solve the problem addressed in Chapter 6.

The actual implementation of the QMR algorithm, without saving all of the Lanczos vectors, is similar to that of the MINRES algorithm described in section 2.5. The least squares problem (5.9) is solved by factoring the $k+1$-by-k matrix $T_{k+1,k}$ into the product of a $k+1$-by-$k+1$ unitary matrix F^H

and a $k+1$-by-k upper triangular matrix R. This is accomplished by using k Givens rotations $F_1, \ldots, F_k$, where F_i rotates the unit vectors ξ_i and ξ_{i+1} through angle θ_i. Since $T_{k+1,k}$ is tridiagonal, R has the form

$$R = \begin{pmatrix} \rho_1 & \sigma_1 & \tau_1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \tau_{k-2} \\ & & & \ddots & \sigma_{k-1} \\ & & & & \rho_k \\ 0 & \ldots & \ldots & \ldots & 0 \end{pmatrix}.$$

The QR decomposition of $T_{k+1,k}$ is easily updated from that of $T_{k,k-1}$. To obtain R, first premultiply the last column of $T_{k+1,k}$ by the rotations from steps $k-2$ and $k-1$ to obtain a matrix of the form

$$F_{k-1}F_{k-2}(F_{k-3}\cdots F_1)T_{k+1,k} = \begin{pmatrix} x & x & x & & 0 \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & x \\ & & & \ddots & x \\ & & & & d \\ 0 & \ldots & \ldots & \ldots & h \end{pmatrix},$$

where the x's denote nonzeros and where the $(k+1,k)$-entry, h, is just γ_k, since this entry is unaffected by the previous rotations. The next rotation, F_k, is chosen to annihilate this entry by setting $c_k = |d|/\sqrt{|d|^2+|h|^2}$; $\bar{s}_k = c_k h/d$ if $d \neq 0$, and $c_k = 0$, $\bar{s}_k = 1$ if $d = 0$. To solve the least squares problem, the successive rotations are also applied to the right-hand side vector $\beta\xi_1$ to obtain $g = F_k \cdots F_1 \beta \xi_1$. Clearly, g differs from the corresponding vector at step $k-1$ only in positions k and $k+1$. If $R_{k\times k}$ denotes the top k-by-k block of R and $g_{k\times 1}$ denotes the first k entries of g, then the solution to the least squares problem is the solution of the triangular linear system

$$R_{k\times k} y_k = g_{k\times 1}.$$

In order to update the iterates x_k, we define auxiliary vectors

$$P_k \equiv (p_0, \ldots, p_{k-1}) \equiv V_k R_{k\times k}^{-1}.$$

Then since

$$x_k = x_0 + V_k y_k = x_0 + P_k g_{k\times 1}$$

and

$$x_{k-1} = x_0 + P_{k-1} g_{k-1\times 1},$$

we can write

$$x_k = x_{k-1} + a_{k-1} p_{k-1}, \tag{5.11}$$

where a_{k-1} is the kth entry of g. Finally, from the equation $P_k R_{k\times k} = V_k$, we can update the auxiliary vectors using

$$p_{k-1} = \frac{1}{\rho_k}(v_k - \sigma_{k-1} p_{k-2} - \tau_{k-2} p_{k-3}). \tag{5.12}$$

This leads to the following implementation of the QMR algorithm.

Algorithm 5. Quasi-Minimal Residual Method (QMR) (without look-ahead).

Given x_0, compute $r_0 = b - Ax_0$ and set $v_1 = r_0/\|r_0\|$.
Given $\hat{r}_0$, set $w_1 = \hat{r}_0/\|\hat{r}_0\|$. Initialize $\xi = (1, 0, \ldots, 0)^T$, $\beta = \|r_0\|$.
For $k = 1, 2, \ldots,$

Compute v_{k+1}, w_{k+1}, $\alpha_k \equiv T(k,k)$, $\beta_k \equiv T(k, k+1)$, and $\gamma_k \equiv T(k+1, k)$, using the two-sided Lanczos algorithm.

Apply F_{k-2} and F_{k-1} to the last column of T; that is,

$$\begin{pmatrix} T(k-2,k) \\ T(k-1,k) \end{pmatrix} \leftarrow \begin{pmatrix} c_{k-2} & s_{k-2} \\ -\bar{s}_{k-2} & c_{k-2} \end{pmatrix} \begin{pmatrix} 0 \\ T(k-1,k) \end{pmatrix}, \quad \text{if } k > 2,$$

$$\begin{pmatrix} T(k-1,k) \\ T(k,k) \end{pmatrix} \leftarrow \begin{pmatrix} c_{k-1} & s_{k-1} \\ -\bar{s}_{k-1} & c_{k-1} \end{pmatrix} \begin{pmatrix} T(k-1,k) \\ T(k,k) \end{pmatrix}, \quad \text{if } k > 1.$$

Compute the kth rotation c_k and s_k, to annihilate the $(k+1, k)$ entry of T.[1]

Apply kth rotation to ξ and to last column of T:

$$\begin{pmatrix} \xi(k) \\ \xi(k+1) \end{pmatrix} \leftarrow \begin{pmatrix} c_k & s_k \\ -\bar{s}_k & c_k \end{pmatrix} \begin{pmatrix} \xi(k) \\ 0 \end{pmatrix}$$

$$T(k,k) \leftarrow c_k T(k,k) + s_k T(k+1,k), \qquad T(k+1,k) \leftarrow 0.$$

Compute $p_{k-1} = [v_k - T(k-1,k)p_{k-2} - T(k-2,k)p_{k-3}]/T(k,k)$, where undefined terms are zero for $k \leq 2$.

Set $x_k = x_{k-1} + \beta\xi(k)p_{k-1}$.

[1]The formula is $c_k = |T(k,k)|/\sqrt{|T(k,k)|^2 + |T(k+1,k)|^2}$, $\bar{s}_k = c_k T(k+1,k)/T(k,k)$, but a more robust implementation should be used. See, for example, BLAS routine DROTG [32].

5.4. Relation Between BiCG and QMR.

The observant reader may have noted that the solutions of the linear system (5.6) and the least squares problem (5.9) are closely related. Hence one might expect a close relationship between the residual norms in the BiCG and QMR algorithms. Here we establish such a relationship, assuming that the Lanczos vectors are well defined and that the tridiagonal matrix in (5.6) is nonsingular.

We begin with a general theorem about the relationship between upper Hessenberg linear systems and least squares problems. Let H_k, $k = 1, 2, \ldots$, denote a family of upper Hessenberg matrices, where H_k is k-by-k and H_{k-1} is the $k-1$-by-$k-1$ principal submatrix of H_k. For each k, define the $k+1$-by-k matrix $H_{k+1,k}$ by

$$H_{k+1,k} = \begin{pmatrix} H_k \\ h_{k+1,k}\xi_k^T \end{pmatrix}.$$

The matrix $H_{k+1,k}$ can be factored in the form $F^H R$, where F is a $k+1$-by-$k+1$ unitary matrix and R is a $k+1$ by k upper triangular matrix. This factorization can be performed using plane rotations in the manner described for the GMRES algorithm in section 2.4:

$$(F_k \cdots F_1)\, H_{k+1,k} = R, \quad \text{where } F_i = \begin{pmatrix} I_{i-1} & & & \\ & c_i & -s_i & \\ & s_i & c_i & \\ & & & I_{k-i} \end{pmatrix}.$$

Note that the first $k-1$ sines and cosines s_i, c_i, $i = 1, \ldots, k-1$, are those used in the factorization of $H_{k,k-1}$.

Let $\beta > 0$ be given and assume that H_k is nonsingular. Let $\tilde{y}_k$ denote the solution of the linear system $H_k y = \beta\xi_1$, and let y_k denote the solution of the least squares problem $\min_y \|H_{k+1,k}y - \beta\xi_1\|$. Finally, let

$$\tilde{\nu}_k = H_{k+1,k}\tilde{y}_k - \beta\xi_1, \quad \nu_k = H_{k+1,k}y_k - \beta\xi_1.$$

LEMMA 5.4.1. *Using the above notation, the norms of ν_k and $\tilde{\nu}_k$ are related to the sines and cosines of the Givens rotations by*

$$(5.13) \qquad \|\nu_k\| = \beta|s_1 s_2 \cdots s_k| \quad \textit{and} \quad \|\tilde{\nu}_k\| = \beta\frac{1}{|c_k|}|s_1 s_2 \cdots s_k|.$$

It follows that

$$(5.14) \qquad \|\tilde{\nu}_k\| = \frac{\|\nu_k\|}{\sqrt{1 - (\|\nu_k\|/\|\nu_{k-1}\|)^2}}.$$

Proof. The least squares problem with the extended Hessenberg matrix $H_{k+1,k}$ can be written in the form

$$\min_y \|H_{k+1,k}y - \beta\xi_1\| = \min_y \|F(H_{k+1,k}y - \beta\xi_1)\| = \min_y \|Ry - \beta F\xi_1\|,$$

and the solution y_k is determined by solving the upper triangular linear system with coefficient matrix equal to the top k-by-k block of R and right-hand side equal to the first k entries of $\beta F\xi_1$. The remainder $Ry_k - \beta F\xi_1$ is therefore zero except for the last entry, which is just the last entry of $-\beta F\xi_1 = -\beta(F_k \cdots F_1)\xi_1$, which is easily seen to be $-\beta s_1 \cdots s_k$. This establishes the first equality in (5.13).

For the linear system solution $\tilde{y}_k = H_k^{-1}\beta\xi_1$, we have

$$\tilde{\nu}_k = H_{k+1,k}H_k^{-1}\beta\xi_1 - \beta\xi_1,$$

which is zero except for the last entry, which is $\beta h_{k+1,k}$ times the $(k,1)$-entry of H_k^{-1}. Now H_k can be factored in the form $\tilde{F}^H\tilde{R}$, where $\tilde{F} = \tilde{F}_{k-1} \cdots \tilde{F}_1$ and $\tilde{F}_i$ is the k-by-k principal submatrix of F_i. The matrix $H_{k+1,k}$, after applying the first $k-1$ plane rotations, has the form

$$(F_{k-1} \cdots F_1)\, H_{k+1,k} = \begin{pmatrix} x & x & \dots & x \\ & x & \dots & x \\ & & \ddots & \vdots \\ & & & r \\ & & & h \end{pmatrix}$$

where r is the (k,k)-entry of $\tilde{R}$ and $h = h_{k+1,k}$. The kth rotation is chosen to annihilate the nonzero entry in the last row:

$$c_k = r/\sqrt{r^2+h^2}, \quad s_k = -h/\sqrt{r^2+h^2}.$$

Note that r and c_k are nonzero since H_k is nonsingular.

We can write $H_k^{-1} = \tilde{R}^{-1}\tilde{F}$, and the $(k,1)$-entry of this is $1/r$ times the $(k,1)$-entry of $\tilde{F} = \tilde{F}_{k-1} \cdots \tilde{F}_1$, and this is just $s_1 \cdots s_{k-1}$. It follows that the nonzero entry of $\tilde{\nu}_k$ is $\beta(h_{k+1,k}/r)s_1 \cdots s_{k-1}$. Finally, using the fact that $|s_k/c_k| = |h/r| = |h_{k+1,k}/r|$, we obtain the second equality in (5.13).

From (5.13) it is clear that

$$\frac{\|\nu_k\|}{\|\nu_{k-1}\|} = |s_k|, \quad \frac{\|\tilde{\nu}_k\|}{\|\nu_k\|} = 1/|c_k|.$$

The result (5.14) follows upon replacing $|c_k|$ by $\sqrt{1-|s_k|^2}$. □

An immediate consequence of this lemma is the following relationship between the BiCG residual r_k^B and the quantity

$$z_k^Q \equiv \beta\xi_1 - T_{k+1,k}y_k^Q, \tag{5.15}$$

which is related to the residual r_k^Q in the QMR algorithm:

$$r_k^Q = V_{k+1}z_k^Q, \quad \|r_k^Q\| \le \sqrt{k+1}\|z_k^Q\|. \tag{5.16}$$

We will refer to z_k^Q as the QMR *quasi-residual*—the vector whose norm is actually minimized in the QMR algorithm.

TABLE 5.1

Relation between QMR quasi-residual norm reduction and ratio of BiCG residual norm to QMR quasi-residual norm.

$\|z_k^Q\|/\|z_{k-1}^Q\|$	$\|r_k^B\|/\|z_k^Q\|$
.5	1.2
.9	2.3
.99	7.1
.9999	70.7
.999999	707

THEOREM 5.4.1. *Assume that the Lanczos vectors at steps* 1 *through* k *are defined and that the tridiagonal matrix generated by the Lanczos algorithm at step* k *is nonsingular. Then the BiCG residual* r_k^B *and the QMR quasi-residual* z_k^Q *are related by*

$$\|r_k^B\| = \frac{\|z_k^Q\|}{\sqrt{1-(\|z_k^Q\|/\|z_{k-1}^Q\|)^2}}. \tag{5.17}$$

Proof. From (5.1), (5.5), and (5.6), it follows that the BiCG residual can be written in the form

$$\begin{aligned} r_k^B &= r_0 - AV_k y_k^B \\ &= r_0 - V_{k+1}T_{k+1,k}y_k^B \\ &= V_{k+1}(\beta\xi_1 - T_{k+1,k}T_k^{-1}\beta\xi_1). \end{aligned}$$

The quantity in parentheses has only one nonzero entry (in its $(k+1)$st position), and since $\|v_{k+1}\| = 1$, we have

$$\|r_k^B\| = \|\beta\xi_1 - T_{k+1,k}T_k^{-1}\beta\xi_1\|. \tag{5.18}$$

The desired result now follows from Lemma 5.4.1 and the definition (5.15) of z_k^Q. □

In most cases, the quasi-residual norms and the actual residual norms in the QMR algorithm are of the same order of magnitude. Inequality (5.16) shows that the latter can exceed the former by at most a factor of $\sqrt{k+1}$, and a bound in the other direction is given by

$$\|r_k^Q\| \geq \sigma_{min}(V_{k+1})\|z_k^Q\|,$$

where σ_{min} denotes the smallest singular value. While it is possible that $\sigma_{min}(V_{k+1})$ is very small (especially in finite precision arithmetic), it is unlikely that $\|r_k^Q\|$ would be much smaller than $\|z_k^Q\|$. The vector y_k^Q is chosen to satisfy the least squares problem (5.9), without regard to the matrix V_{k+1}.

Theorem 5.4.1 shows that if the QMR quasi-residual norm is reduced by a significant factor at step k, then the BiCG residual norm will be approximately

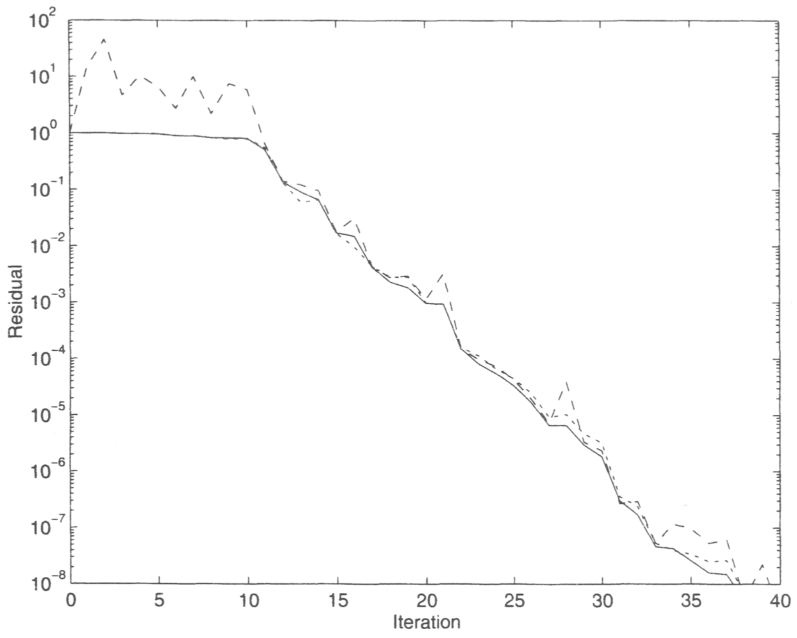

FIG. 5.1. *BiCG residual norms (dashed), QMR residual norms (dotted), and QMR quasi-residual norms (solid).*

equal to the QMR quasi-residual norm at step k, since the denominator in the right-hand side of (5.17) will be close to 1. If the QMR quasi-residual norm remains almost constant, however, then the denominator in the right-hand side of (5.17) will be close to 0, and the BiCG residual norm will be much larger. Table 5.1 shows the relation between the QMR quasi-residual norm reduction and the ratio of BiCG residual norm to QMR quasi-residual norm. Note that the QMR quasi-residual norm must be *very* flat before the BiCG residual norm is orders-of-magnitude larger.

Figure 5.1 shows a plot of the logarithms of the norms of the BiCG residuals (dashed line), the QMR residuals (dotted line), and the QMR quasi-residuals (solid line) versus iteration number for a simple example problem. The matrix A, a real 103-by-103 matrix, was taken to have 50 pairs of complex conjugate eigenvalues, randomly distributed in the rectangle $[1,2] \times [-i,i]$, and 3 additional real eigenvalues at 4, .5, and -1. A random matrix V was generated and A was set equal to VDV^{-1}, where D is a block-diagonal matrix with 3 1-by-1 blocks corresponding to the separated real eigenvalues of A and 50 2-by-2 blocks of the form

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}$$

corresponding to the pairs of eigenvalues $a \pm ib$.

In this example, the QMR residual and quasi-residual norm curves are barely distinguishable. As predicted by Theorem 5.4.1, peaks in the BiCG residual norm curve correspond to plateaus in the QMR convergence curve. At steps where the QMR quasi-residual norm is reduced by a large factor, the BiCG residual norm is reduced by an even greater amount so that it "catches up" with QMR.

Relation (5.17) implies roughly that the BiCG and QMR algorithms will either both converge well or both perform poorly for a given problem. While the QMR quasi-residual norm cannot increase during the iteration, it is no more useful to have a near constant residual norm than it is to have an increasing one. The analysis here assumes exact arithmetic, however. In finite precision arithmetic, one might expect that a very large intermediate iterate (corresponding to a very large residual norm) could lead to inaccuracy in the final approximation, and, indeed, such a result was established in [66]. This will be discussed further in section 7.3. A thorough study of the effect of rounding errors on the BiCG and QMR algorithms has not been carried out, however. Since the BiCG and QMR algorithms require essentially the same amount of work and storage per iteration and since the QMR quasi-residual norm is always less than or equal to the BiCG residual norm, it seems reasonable to choose QMR over BiCG, although the difference may not be great.

5.5. The Conjugate Gradient Squared Algorithm.

The BiCG and QMR algorithms require multiplication by both A and A^H at each step. This means extra work, and, additionally, it is sometimes much less convenient to multiply by A^H than it is to multiply by A. For example, there may be a special formula for the product of A with a given vector when A represents, say, a Jacobian, but a corresponding formula for the product of A^H with a given vector may not be available. In other cases, data may be stored on a parallel machine in such a way that multiplication by A is efficient but multiplication by A^H involves extra communication between processors. For these reasons it is desirable to have an iterative method that requires multiplication only by A and that generates good approximate solutions from the Krylov spaces of dimension equal to the number of matrix–vector multiplications. A method that attempts to do this is the conjugate gradient squared (CGS) method.

Returning to the BiCG algorithm of section 5.2, note that we can write

$$r_k = \varphi_k(A)r_0, \quad \hat{r}_k = \bar{\varphi}_k(A^H)\hat{r}_0,$$
$$p_k = \psi_k(A)r_0, \quad \hat{p}_k = \bar{\psi}_k(A^H)\hat{r}_0$$

for certain kth-degree polynomials φ_k and ψ_k. If the algorithm is converging well, then $\|\varphi_k(A)r_0\|$ is small and one might expect that $\|\varphi_k^2(A)r_0\|$ would be even smaller. If $\varphi_k^2(A)r_0$ could be computed with about the same amount of work as $\varphi_k(A)r_0$, then this would likely result in a faster converging algorithm. This is the idea of CGS.

Rewriting the BiCG recurrence in terms of these polynomials, we see that

$$\varphi_k(A)r_0 = \varphi_{k-1}(A)r_0 - a_{k-1}A\psi_{k-1}(A)r_0, \tag{5.19}$$

$$\psi_k(A)r_0 = \varphi_k(A)r_0 + b_k\psi_{k-1}(A)r_0, \tag{5.20}$$

where

$$a_{k-1} = \frac{\langle \varphi_{k-1}(A)r_0, \bar{\varphi}_{k-1}(A^H)\hat{r}_0\rangle}{\langle A\psi_{k-1}(A)r_0, \bar{\psi}_{k-1}(A^H)\hat{r}_0\rangle} = \frac{\langle \varphi_{k-1}^2(A)r_0, \hat{r}^0\rangle}{\langle A\psi_{k-1}^2(A)r_0, \hat{r}_0\rangle}, \tag{5.21}$$

$$b_k = \frac{\langle \varphi_k(A)r_0, \bar{\varphi}_k(A^H)\hat{r}_0\rangle}{\langle \varphi_{k-1}(A)r_0, \bar{\varphi}_{k-1}(A^H)\hat{r}_0\rangle} = \frac{\langle \varphi_k^2(A)r_0, \hat{r}_0\rangle}{\langle \varphi_{k-1}^2(A)r_0, \hat{r}_0\rangle}. \tag{5.22}$$

Note that the coefficients can be computed if we know $\hat{r}_0$ and $\varphi_j^2(A)r_0$ and $\psi_j^2(A)r_0$, $j = 1, 2, \ldots$.

From (5.19–5.20), it can be seen that the polynomials $\varphi_k(z)$ and $\psi_k(z)$ satisfy the recurrences

$$\varphi_k(z) = \varphi_{k-1}(z) - a_{k-1}z\psi_{k-1}(z), \quad \psi_k(z) = \varphi_k(z) + b_k\psi_{k-1}(z),$$

and squaring both sides gives

$$\varphi_k^2(z) = \varphi_{k-1}^2(z) - 2a_{k-1}z\varphi_{k-1}(z)\psi_{k-1}(z) + a_{k-1}^2z^2\psi_{k-1}^2(z),$$

$$\psi_k^2(z) = \varphi_k^2(z) + 2b_k\varphi_k(z)\psi_{k-1}(z) + b_k^2\psi_{k-1}^2(z).$$

Multiplying φ_k by the recurrence for ψ_k gives

$$\varphi_k(z)\psi_k(z) = \varphi_k^2(z) + b_k\varphi_k(z)\psi_{k-1}(z),$$

and multiplying the recurrence for φ_k by ψ_{k-1} gives

$$\begin{aligned} \varphi_k(z)\psi_{k-1}(z) &= \varphi_{k-1}(z)\psi_{k-1}(z) - a_{k-1}z\psi_{k-1}^2(z) \\ &= \varphi_{k-1}^2(z) + b_{k-1}\varphi_{k-1}(z)\psi_{k-2}(z) - a_{k-1}z\psi_{k-1}^2(z). \end{aligned}$$

Defining

$$\Phi_k \equiv \varphi_k^2, \quad \Theta_k \equiv \varphi_k\psi_{k-1}, \quad \Psi_{k-1} \equiv \psi_{k-1}^2,$$

these recurrences become

$$\begin{aligned} \Phi_k(z) &= \Phi_{k-1}(z) - 2a_{k-1}z(\Phi_{k-1}(z) + b_{k-1}\Theta_{k-1}(z)) + a_{k-1}^2z^2\Psi_{k-1}(z), \\ \Theta_k(z) &= \Phi_{k-1}(z) + b_{k-1}\Theta_{k-1}(z) - a_{k-1}z\Psi_{k-1}(z), \\ \Psi_k(z) &= \Phi_k(z) + 2b_k\Theta_k(z) + b_k^2\Psi_{k-1}. \end{aligned}$$

Let $r_k^S = \Phi_k(A)r_0$, $p_k^S = \Psi_k(A)r_0$, and $q_k^S = \Theta_k(A)r_0$. Then the following algorithm generates an approximate solution x_k^S with the required residual r_k^S.

Conjugate Gradient Squared Algorithm (CGS).

Given $x_0 \equiv x_0^S$, compute $r_0 \equiv r_0^S = b - Ax_0^S$, set $u_0^S = r_0^S$, $p_0^S = r_0^S$, $q_0^S = 0$, and $v_0^S = Ap_0^S$. Set an arbitrary vector $\hat{r}_0$. For $k = 1, 2, \ldots$

Compute $q_k^S = u_{k-1}^S - a_{k-1}v_{k-1}^S$, where $a_{k-1} = \frac{\langle r_{k-1}^S, \hat{r}_0\rangle}{\langle v_{k-1}^S, \hat{r}_0\rangle}$.

Set $x_k^S = x_{k-1}^S + a_{k-1}(u_{k-1}^S + q_k^S)$.
Then $r_k^S = r_{k-1}^S - a_{k-1}A(u_{k-1}^S + q_k^S)$.
Compute $u_k^S = r_k^S + b_k q_k^S$, where $b_k = \frac{\langle r_k^S, \hat{r}_0\rangle}{\langle r_{k-1}^S, \hat{r}_0\rangle}$.
Set $p_k^S = u_k^S + b_k(q_k^S + b_k p_{k-1}^S)$ and $v_k^S = Ap_k^S$.

The CGS method requires two matrix–vector multiplications at each step but no multiplications by the Hermitian transpose. For problems where the BiCG method converges well, CGS typically requires only about half as many steps and, therefore, half the work of BiCG (assuming that multiplication by A or A^H requires the same amount of work). When the norm of the BiCG residual increases at a step, however, that of the CGS residual usually increases by approximately the square of the increase of the BiCG residual norm. The CGS convergence curve may therefore show wild oscillations that can sometimes lead to numerical instabilities.

5.6. The BiCGSTAB Algorithm.

To avoid the large oscillations in the CGS convergence curve, one might try to produce a residual of the form

$$r_k = \chi_k(A)\varphi_k(A)r_0,$$

where φ_k is again the BiCG polynomial but χ_k is chosen to try and keep the residual norm small at each step while retaining the rapid overall convergence of CGS. For example, if $\chi_k(z)$ is of the form

$$\chi_k(z) = (1-\omega_k z)(1-\omega_{k-1}z)\cdots(1-\omega_1 z), \tag{5.23}$$

then the coefficients ω_j can be chosen at each step to minimize

$$\|r_j\| = \|(I-\omega_j A)\chi_{j-1}(A)\varphi_j(A)r_0\|.$$

This leads to the BiCGSTAB algorithm, which might be thought of as a combination of BiCG with Orthomin(1).

Again letting $\varphi_k(A)r_0$ denote the BiCG residual at step k and $\psi_k(A)r_0$ denote the BiCG direction vector at step k, recall that these polynomials satisfy recurrences (5.19–5.20). In the BiCGSTAB scheme we will need recurrences for

$$r_k = \chi_k(A)\varphi_k(A)r_0 \quad \text{and} \quad p_k = \chi_k(A)\psi_k(A)r_0.$$

It follows from (5.23) and (5.19–5.20) that

$$\begin{aligned} r_k &= (I-\omega_k A)\chi_{k-1}(A)[\varphi_{k-1}(A) - a_{k-1}A\psi_{k-1}(A)]r_0 \\ &= (I-\omega_k A)[r_{k-1} - a_{k-1}Ap_{k-1}], \end{aligned}$$

$$\begin{aligned} p_k &= \chi_k(A)[\varphi_k(A) + b_k\psi_{k-1}(A)]r_0 \\ &= r_k + b_k(I-\omega_k A)p_{k-1}. \end{aligned}$$

Finally, we need to express the BiCG coefficients a_{k-1} and b_k in terms of the new vectors. Using the biorthogonality properties of the BiCG polynomials—$\langle \varphi_k(A)r_0, A^{H^j}\hat{r}_0\rangle = \langle A\psi_k(A)r_0, A^{H^j}\hat{r}_0\rangle = 0$, $j = 0, 1, \ldots, k-1$ (see Exercise 5.3)—together with the recurrence relations (5.19–5.20), we derive the following expressions for inner products appearing in the coefficient formulas (5.21–5.22):

$$\langle \varphi_{k-1}(A)r_0, \bar{\varphi}_{k-1}(A^H)\hat{r}_0\rangle = (-1)^{k-1}a_{k-2}\cdots a_0\langle \varphi_{k-1}(A)r_0, A^{H^{k-1}}\hat{r}_0\rangle,$$

$$\langle A\psi_{k-1}(A)r_0, \bar{\psi}_{k-1}(A^H)\hat{r}_0\rangle = (-1)^{k-1}a_{k-2}\cdots a_0\langle A\psi_{k-1}(A)r_0, A^{H^{k-1}}\hat{r}_0\rangle.$$

It also follows from these same biorthogonality and recurrence relations that the BiCGSTAB vectors satisfy

$$\begin{aligned}\langle r_{k-1}, \hat{r}_0\rangle &= \langle \varphi_{k-1}(A)r_0, \bar{\chi}_{k-1}(A^H)\hat{r}_0\rangle \\ &= (-1)^{k-1}\omega_{k-1}\cdots\omega_1\langle \varphi_{k-1}(A)r_0, A^{H^{k-1}}\hat{r}_0\rangle,\end{aligned}$$

$$\begin{aligned}\langle Ap_{k-1}, \hat{r}_0\rangle &= \langle A\psi_{k-1}(A)r_0, \bar{\chi}_{k-1}(A^H)\hat{r}_0\rangle \\ &= (-1)^{k-1}\omega_{k-1}\cdots\omega_1\langle A\psi_{k-1}(A)r_0, A^{H^{k-1}}\hat{r}_0\rangle,\end{aligned}$$

and hence the coefficient formulas (5.21–5.22) can be replaced by

$$a_{k-1} = \frac{\langle r_{k-1}, \hat{r}_0\rangle}{\langle Ap_{k-1}, \hat{r}_0\rangle}, \quad b_k = \frac{a_{k-1}}{\omega_k}\frac{\langle r_k, \hat{r}_0\rangle}{\langle r_{k-1}, \hat{r}_0\rangle}.$$

This leads to the following algorithm.

Algorithm 6. BiCGSTAB.

```
Given $x_0$, compute $r_0 = b - Ax_0$ and set $p_0 = r_0$.
Choose $\hat{r}_0$ such that $\langle r_0, \hat{r}_0\rangle \neq 0$. For $k = 1, 2, \ldots,$

Compute $Ap_{k-1}$.

Set $x_{k-1/2} = x_{k-1} + a_{k-1}p_{k-1}$, where $a_{k-1} = \frac{\langle r_{k-1}, \hat{r}_0\rangle}{\langle Ap_{k-1}, \hat{r}_0\rangle}$.

Compute $r_{k-1/2} = r_{k-1} - a_{k-1}Ap_{k-1}$.

Compute $Ar_{k-1/2}$.

Set $x_k = x_{k-1/2} + \omega_k r_{k-1/2}$, where $\omega_k = \frac{\langle r_{k-1/2}, Ar_{k-1/2}\rangle}{\langle Ar_{k-1/2}, Ar_{k-1/2}\rangle}$.

Compute $r_k = r_{k-1/2} - \omega_k Ar_{k-1/2}$.

Compute $p_k = r_k + b_k(p_{k-1} - \omega_k Ap_{k-1})$, where $b_k = \frac{a_{k-1}}{\omega_k}\frac{\langle r_k, \hat{r}_0\rangle}{\langle r_{k-1}, \hat{r}_0\rangle}$.
```

5.7. Which Method Should I Use?

For Hermitian problems, the choice of an iterative method is fairly straightforward—use CG or MINRES for positive definite problems and MINRES for indefinite problems. One can also use a form of the CG algorithm for indefinite problems. The relation between residual norms for CG and MINRES is like that for BiCG and QMR, however, as shown in Exercise 5.1. For this reason, the MINRES method is usually preferred. By using a simpler iteration, such as the simple iteration method described in section 2.1 or the Chebyshev method [94] which has not been described, one can avoid the inner products required in the CG and MINRES algorithms. This gives some savings in the cost of an iteration, but the price in terms of number of iterations usually outweighs the savings. An exception might be the case in which one has such a good preconditioner that even simple iteration requires only one or two steps. For some problems, multigrid methods provide such preconditioners.

The choice of an iterative method for non-Hermitian problems is not so easy. If matrix–vector multiplication is extremely expensive (e.g., if A is dense and has no special properties to enable fast matrix–vector multiplication), then (full) GMRES is probably the method of choice because it requires the fewest matrix–vector multiplications to reduce the residual norm to a desired level. If matrix–vector multiplication is not so expensive or if storage becomes a problem for full GMRES, then one of the methods described in this chapter is probably a good choice. Because of relation (5.17), we generally recommend QMR over BiCG.

The choice between QMR, CGS, and BiCGSTAB is problem dependent. There are also transpose-free versions of QMR that have not been described here [53]. Another approach, to be discussed in section 7.1, is to symmetrize the problem. For example, instead of solving $Ax = b$, one could solve $A^H Ax = A^H b$ or $AA^H y = b$ (so that $x = A^H y$) using the CG method. Of course, one does not actually form the normal equations; it is necessary only to compute matrix–vector products with A and A^H. How this approach compares with the methods described in this chapter is also problem dependent. In [102], the GMRES (full or restarted), CGS, and CGNE (CG for $AA^H y = b$) iterations were considered. For each method an example was constructed for which that method was by far the best and another example was given for which that method was by far the worst. Thus none of these methods can be eliminated as definitely inferior to one of the others, and none can be recommended as *the* method of choice for non-Hermitian problems.

To give some indication of the performance of the methods, we show here plots of residual norm and error norm versus the number of matrix–vector multiplications and versus the number of floating point operations (additions, subtractions, multiplications, and divisions) *assuming* that multiplication by A or A^H requires $9n$ operations. This is the cost of applying a 5-diagonal matrix A to a vector and probably is a lower bound on the cost of matrix–vector multiplication in most practical applications. The methods considered are full

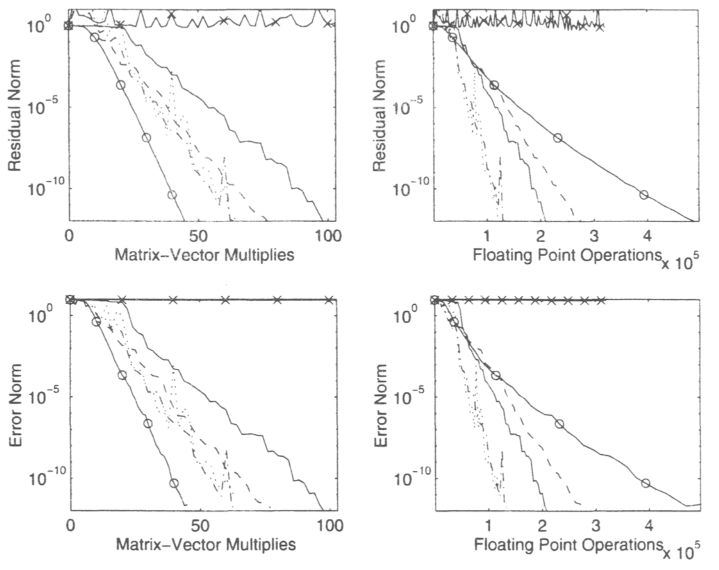

FIG. 5.2. *Performance of full GMRES (solid with o's), GMRES*(10) *(dashed), QMR (solid), CGS (dotted), BiCGSTAB (dash-dot), and CGNE (solid with x's).*

GMRES, GMRES(10) (that is, GMRES restarted after every 10 steps), QMR, CGS, BiCGSTAB, and CGNE.

The problem is the one described in section 5.4—a real 103-by-103 matrix A with random eigenvectors and with 50 pairs of complex conjugate eigenvalues randomly distributed in $[1,2] \times [-i,i]$ and 3 additional real eigenvalues at 4, .5, and -1. Results are shown in Figure 5.2. The full GMRES algorithm necessarily requires the fewest matrix-vector multiplications to achieve a given residual norm. In terms of floating point operations, however, when matrix-vector multiplication requires only $9n$ operations, full GMRES is the most expensive method. The QMR algorithm uses two matrix-vector multiplications per step (one with A and one with A^H) to generate an approximation whose residual lies in the same Krylov space as the GMRES residual. Hence QMR requires at least twice as many matrix-vector multiplications to reduce the residual norm to a given level, and for this problem it requires only slightly more than this. A transpose-free variant of QMR would likely be more competitive. Since the CGS and BiCGSTAB methods construct a residual at step k that comes from the Krylov space of dimension $2k$ (using two matrix-vector multiplications per step), these methods could conceivably require as few matrix-vector multiplications as GMRES. For this example they require a moderate number of additional matrix-vector multiplications, but these seem to be the most efficient in terms of floating point operations. The CGNE method proved *very* inefficient for this problem, hardly reducing the error at all over the first 52 steps (104 matrix-vector multiplications). The condition number of the matrix AA^H is 10^8, so this is not so surprising.

The results of this one test problem should *not* be construed as indicative of the relative performance of these algorithms for all or even most applications. In the Exercises, we give examples in which some of these methods perform far better or far worse than the others. It remains an open problem to characterize the classes of problems for which one method outperforms the others. For additional experimental results, see [128, 28].

Comments and Additional References.

The QMR algorithm was developed by Freund and Nachtigal [54]. Theorem 5.3.1 was given in [101].

Relation (5.13) in Lemma 5.4.1 has been established in a number of places (e.g., [22, 53, 54, 75, 111, 140]), but, surprisingly, the one-step leap to relation (5.14) and its consequence (5.17) seems not to have been taken explicitly until [29]. A similar relation between the GMRES and FOM residuals was observed in [22].

The CGS algorithm was developed by Sonneveld [124] and BiCGSTAB by van der Vorst [134].

An excellent survey article on iterative methods based on the nonsymmetric Lanczos algorithm, along with many references, is given in [76].

Exercises.

5.1. Use Lemma 5.4.1 to show that for Hermitian matrices A, the CG residual r_k^C is related to the MINRES residual r_k^M by

$$\|r_k^C\| = \frac{\|r_k^M\|}{\sqrt{1 - (\|r_k^M\|/\|r_{k-1}^M\|)^2}},$$

provided that the tridiagonal matrix T_k generated by the Lanczos algorithm is nonsingular.

5.2. Let r_k^Q denote the residual at step k of the QMR algorithm and let $T_{k+1,k}$ denote the $k+1$-by-k tridiagonal matrix generated by the Lanczos algorithm. Let T be any tridiagonal matrix whose upper left $k+1$-by-k block is $T_{k+1,k}$. Use the fact that the Arnoldi algorithm, applied to T with initial vector ξ_1, generates the same matrix $T_{k+1,k}$ at step k to show that

$$\|r_k^Q\| \leq \|V_{k+1}\| \cdot \|r_k^G(T)\| \leq \sqrt{k+1}\ \|r_k^G(T)\|,$$

where $r_k^G(T)$ is the residual at step k of the GMRES algorithm applied to the linear system $T\chi = \|r_0\|\xi_1$. If T is taken to be the tridiagonal matrix generated at step n of the Lanczos algorithm (assuming the algorithm does not break down or terminate before step n), then the eigenvalues of T are the same as those of A. Thus the convergence of QMR is like that of GMRES applied to a matrix with the same eigenvalues as A. This does not provide useful a priori information about the convergence rate of QMR, however, as it was noted in section 3.2 that eigenvalue information alone tells nothing about the behavior of GMRES. (This result was proved in [54] for the special case $T = T_{k+1}$, and it was proved in [28] for $T = T_n$.)

5.3. Prove the biconjugacy relations

$$\langle r_k, A^{H^j}\hat{r}_0\rangle = \langle Ap_k, A^{H^j}\hat{r}_0\rangle = 0, \quad j < k$$

for the BiCG algorithm.

5.4. The following examples are taken from [102]. They demonstrate that the performance of various iterative methods can differ dramatically for a given problem and that the best method for one problem may be the worst for another.

(a) *CGNE wins.* Suppose A is the unitary shift matrix

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ & & 1 & \dots & 0 \\ & & & \ddots & \\ & & & & 1 \\ 1 & 0 & \dots & \dots & 0 \end{pmatrix}$$

and b is the first unit vector ξ_1. How many iterations will the full GMRES method need to solve $Ax = b$, with a zero initial guess? What is a lower bound on the number of matrix–vector multiplications required by CGS? How many iterations are required if one applies CG to the normal equations $AA^H y = b$, $x = A^H y$?

(b) *CGNE loses.* Suppose A is the block diagonal matrix

$$\begin{pmatrix} M_1 & & & \\ & M_2 & & \\ & & \ddots & \\ & & & M_{n/2} \end{pmatrix}, \quad M_j = \begin{pmatrix} 1 & j-1 \\ 0 & 1 \end{pmatrix}, \quad 1 \le j \le n/2.$$

What is the degree of the minimal polynomial of A? How many steps will GMRES require to obtain the solution to a linear system $Ax = b$? How many matrix–vector multiplications will CGS require, assuming that $\hat{r}_0 = r_0$? The singular values of this matrix lie approximately in the range $[2/n, n/2]$. Would you expect CGNE to require few or many iterations if n is large?

(c) *CGS wins.* Suppose A is a Hermitian matrix with many eigenvalues distributed throughout an interval $[c, d]$ on the positive real axis. Which method would you expect to require the least amount of *work* to solve a linear system $Ax = b$—(full) GMRES, CGS, or CGNE? Explain your answer. (Of course, one would do better to solve a Hermitian problem using CG or MINRES, but perhaps it is not known that A is Hermitian.)

(d) *CGS loses.* Let A be the skew-symmetric matrix

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \times I_{n/2},$$

that is, an n-by-n block diagonal matrix with 2-by-2 blocks. Show that this matrix is normal and has eigenvalues $\pm\imath$ and singular value 1. How many steps are required to solve a linear system $Ax = b$ using CGNE? GMRES? Show, however, that for any real initial residual r_0, if $\hat{r}_0 = r_0$ then CGS breaks down with a division by 0 at the first step.

5.5. When the two-sided Lanczos algorithm is used in the solution of linear systems, the right starting vector is always the initial residual, $r_0/\|r_0\|$, but the left starting vector $\hat{r}_0$ is not specified. Consider an *arbitrary* 3-term recurrence:

$$\tilde{v}_{j+1} = Av_j - \alpha_j v_j - \beta_{j-1} v_{j-1},$$
$$v_{j+1} = \tilde{v}_{j+1}/\gamma_j, \quad \text{where } \gamma_j = \|\tilde{v}_{j+1}\|.$$

The γ's are chosen so that the vectors have norm 1, but the α's and β's can be anything. Show that if this recurrence is run for no more than $[(n+2)/2]$ steps, then there is a nonzero vector w_1 such that

$$(5.24) \qquad \langle v_j, A^{H^\ell} w_1 \rangle = 0 \quad \forall\, \ell < j-1, \quad j = 2, \ldots, \left[\frac{n+2}{2}\right];$$

i.e., assuming there is no exact breakdown with $\langle v_j, w_j \rangle = 0$, the arbitrary recurrence *is* the two-sided Lanczos algorithm for a certain left starting vector w_1. (Hint: The condition (5.24) is equivalent to $\langle w_1, A^\ell v_j \rangle = 0$ $\forall\, \ell < j-1$, $j = 2, \ldots, [(n+2)/2]$. Show that there are only $n-1$ linearly independent vectors to which w_1 must be orthogonal.)

This somewhat disturbing result suggests that some assumptions must be made about the left starting vector, if we are to have any hope of establishing good a priori error bounds for the Lanczos-based linear system solvers [67]. In practice, however, it is observed that the convergence behavior of these methods is about the same for most randomly chosen left starting vectors or for $\hat{r}_0 = r_0$, which is sometimes recommended.