Chapter 7

Miscellaneous Issues

7.1. Symmetrizing the Problem.

Because of the difficulties in solving non-Hermitian linear systems, one might consider converting a non-Hermitian problem to a Hermitian one by solving the *normal equations*. That is, one applies an iterative method to one of the linear systems

$$A^H A x = A^H b \quad \text{or} \quad A A^H y = b, \quad x = A^H y. \tag{7.1}$$

As usual, this can be accomplished without actually forming the matrices $A^H A$ or AA^H, and, in the latter case, one need not explicitly generate approximations to y but instead can carry along approximations $x_k \equiv A^H y_k$. For instance, if the CG method is used to solve either of the systems in (7.1), then the algorithms, sometimes called CGNR and CGNE, respectively, can be implemented as follows.

Algorithm 7. CG for the Normal Equations (CGNR and CGNE).

Given an initial guess x_0, compute $r_0 = b - Ax_0$.
Compute $A^H r_0$ and set $p_0 = A^H r_0$. For $k = 1, 2, \ldots$,

Compute Ap_{k-1}.

Set $x_k = x_{k-1} + a_{k-1} p_{k-1}$, where
$a_{k-1} = \frac{\langle A^H r_{k-1}, A^H r_{k-1} \rangle}{\langle Ap_{k-1}, Ap_{k-1} \rangle}$ for CGNR, $a_{k-1} = \frac{\langle r_{k-1}, r_{k-1} \rangle}{\langle p_{k-1}, p_{k-1} \rangle}$ for CGNE.

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

Compute $A^H r_k$.

Set $p_k = A^H r_k + b_{k-1} p_{k-1}$, where
$b_{k-1} = \frac{\langle A^H r_k, A^H r_k \rangle}{\langle A^H r_{k-1}, A^H r_{k-1} \rangle}$ for CGNR, $b_{k-1} = \frac{\langle r_k, r_k \rangle}{\langle r_{k-1}, r_{k-1} \rangle}$ for CGNE.

The CGNR algorithm minimizes the $A^H A$-norm of the error, which is the

2-norm of the residual $b - Ax_k$, over the affine space

$$x_k \in x_0 + \text{span}[A^H r_0, (A^H A)A^H r_0, \ldots, (A^H A)^{k-1} A^H r_0].$$

The CGNE algorithm minimizes the AA^H-norm of the error in y_k, which is the 2-norm of the error $x - x_k$, over the affine space

$$x_k \in x_0 + \text{span}[A^H r_0, A^H(AA^H)r_0, \ldots, A^H(AA^H)^{k-1} r_0].$$

Note that these two spaces are the same, and both involve powers of the symmetrized matrix $A^H A$ or AA^H.

Numerical analysts sometimes *cringe* at the thought of solving the normal equations for two reasons. First, since the condition number of $A^H A$ or AA^H is the *square* of the condition number of A, if there were an iterative method for solving $Ax = b$ whose convergence rate was governed by the condition number of A, then squaring this condition number would significantly degrade the convergence rate. Unfortunately, however, for a non-Hermitian matrix A, there is no iterative method whose convergence rate is governed by the condition number of A.

The other objection to solving the normal equations is that one cannot expect to achieve as high a level of accuracy when solving a linear system $Cy = d$ as when solving $Ax = b$, if the condition number of C is greater than that of A. This statement can be based on a simple perturbation argument. Since the entries of C and d probably cannot be represented exactly on the computer (or in the case of iterative methods, the product of C with a given vector cannot be computed exactly), the best approximation $\tilde{y}$ to y that one can hope to find numerically is one that satisfies a nearby system $(C + \delta C)\tilde{y} = d + \delta d$, where the size of δC and δd are determined by the machine precision. If $\tilde{y}$ is the solution of such a perturbed system, then it can be shown, for sufficiently small perturbations, that

$$\frac{\|y - \tilde{y}\|}{\|y\|} \leq \kappa(C) \left(\frac{\|\delta C\|}{\|C\|} + \frac{\|\delta d\|}{\|d\|} \right).$$

If C and d were the only data available for the problem, then the terms $\|\delta C\|/\|C\|$ and $\|\delta d\|/\|d\|$ on the right-hand side of this inequality could not be expected to be less than about ϵ, the machine precision. For the CGNR and CGNE methods, however, not only is $C = A^H A$ available, but the matrix A itself is available. (That is, one can apply A to a given vector.) As a result, it will be shown in section 7.4 that the achievable level of accuracy is about the same as that for the original linear system.

Thus, neither of the standard arguments against solving the normal equations is convincing. There are problems for which the CGNR and CGNE methods are best (Exercise 5.4a), and there are other problems for which one of the non-Hermitian matrix iterations far outperforms these two (Exercise 5.4b). In practice, the latter situation seems to be more common. There is

little theory characterizing problems for which the normal equations approach is or is not to be preferred to a non-Hermitian iterative method. (See Exercise 7.1, however.)

7.2. Error Estimation and Stopping Criteria.

When a linear system is "solved" by a computer using floating point arithmetic, one does not obtain the exact solution, whether a direct or an iterative method is employed. With iterative methods especially, it is important to have some idea of what constitutes an acceptably good approximate solution, so the iteration can be stopped when this level of accuracy is achieved. Often it is desired to have an approximate solution for which some standard norm of the error, say, the 2-norm or the ∞-norm, is less than some tolerance. One can compute the residual $b - Ax_k$, but one cannot compute the error $A^{-1}b - x_k$. Hence one might try to estimate the desired error norm using the residual or other quantities generated during the iteration.

The relative error norm is related to the relative residual norm by

$$\frac{1}{\kappa(A)} \frac{|||b - Ax_k|||}{|||b|||} \leq \frac{|||A^{-1}b - x_k|||}{|||A^{-1}b|||} \leq \kappa(A) \frac{|||b - Ax_k|||}{|||b|||}, \tag{7.2}$$

where $\kappa(A) = |||A||| \cdot |||A^{-1}|||$ and $|||\cdot|||$ represents any vector norm and its induced matrix norm. To see this, note that since $b - Ax_k = A(A^{-1}b - x_k)$, we have

$$|||b - Ax_k||| \leq |||A||| \cdot |||A^{-1}b - x_k|||, \tag{7.3}$$

$$|||A^{-1}b - x_k||| \leq |||A^{-1}||| \cdot |||b - Ax_k|||. \tag{7.4}$$

Since we also have $|||A^{-1}b||| \leq |||A^{-1}||| \cdot |||b|||$, combining this with (7.3) gives the first inequality in (7.2). Using the inequality $|||b||| \leq |||A||| \cdot |||A^{-1}b|||$ with (7.4) gives the second inequality in (7.2). To obtain upper and lower bounds on the desired error norm, one might therefore attempt to estimate the condition number of A in this norm.

It was noted at the end of Chapter 4 that the eigenvalues of the tridiagonal matrix T_k generated by the Lanczos algorithm (and, implicitly, by the CG and MINRES algorithms for Hermitian matrices) provide estimates of some of the eigenvalues of A. Hence, if A is Hermitian and the norm in (7.2) is the 2-norm, then the eigenvalues of T_k can be used to estimate $\kappa(A)$. It is easy to show (Exercise 7.2) that the ratio of largest to smallest eigenvalue of T_k gives a lower bound on the condition number of A, but in practice it is usually a very good estimate, even for moderate size values of k. Hence one might stop the iteration when

$$\kappa(T_k) \frac{\|b - Ax_k\|}{\|b\|} \leq \text{tol}, \tag{7.5}$$

where tol is the desired tolerance for the 2-norm of the relative error. This could cause the iteration to terminate too soon, since $\kappa(T_k) \leq \kappa(A)$, but, more

often, it results in extra iterations because the right-hand side of (7.2) is an overestimate of the actual error norm.

Unfortunately, for most other norms in (7.2), the condition number of A cannot be well approximated by the condition number (or any other simple function) of T_k. For non-Hermitian matrix iterations such as BiCG, QMR, and GMRES, the condition number of A cannot be approximated easily using the underlying non-Hermitian tridiagonal or upper Hessenberg matrix. (In the GMRES algorithm, one might consider approximating $\kappa(A)$ in the 2-norm by the ratio of largest to smallest singular value of H_k, since, at least for $k = n$, we have $\kappa(H_n) = \kappa(A)$. Unfortunately, however, for $k < n$, the singular values of H_k do not usually provide good estimates of the singular values of A.)

Since the right-hand side of (7.2) is an overestimate of the actual error norm, one might try to use quantities generated during the iteration in different ways. Sometimes an iteration is stopped when the difference between two consecutive iterates or between several consecutive iterates is less than some tolerance. For most iterative methods, however, this could lead to termination before the desired level of accuracy is achieved.

Consider the CG algorithm where A is Hermitian and positive definite and it is desired to reduce the A-norm of the error to a certain tolerance. The error $e_k \equiv x - x_k$ satisfies

$$e_k = e_{k-1} - a_{k-1}p_{k-1},$$

$$\langle e_k, Ae_k\rangle = \langle e_{k-1}, Ae_{k-1}\rangle - |a_{k-1}\langle r_{k-1}, p_{k-1}\rangle| \tag{7.6}$$

$$= \langle e_{k-d}, Ae_{k-d}\rangle - \sum_{j=1}^{d} |a_{k-j}\langle r_{k-j}, p_{k-j}\rangle|. \tag{7.7}$$

The A-norm of the difference $x_k - x_{k-1} = a_{k-1}p_{k-1}$ gives a *lower bound* on the error at step $k-1$:

$$\langle e_{k-1}, Ae_{k-1}\rangle \geq |a_{k-1}\langle r_{k-1}, p_{k-1}\rangle| = \|x_k - x_{k-1}\|_A^2.$$

To obtain an upper bound, one can use the fact that the A-norm of the error is reduced by at least the factor $(\kappa - 1)/(\kappa + 1)$ at every step, or, more generally, that the A-norm of the error is reduced by at least the factor

$$\gamma_d \equiv 2\left[\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^d + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^d\right]^{-1} \tag{7.8}$$

after every d steps. Here κ denotes the condition number of A in the 2-norm and is the ratio of largest to smallest eigenvalue of A. This error bound follows from the fact that the CG polynomial is optimal for minimizing the A-norm of the error and hence is at least as good as the product of the $(k-d)$th-degree CG polynomial with the dth-degree Chebyshev polynomial. (See Theorem 3.1.1.)

Using (7.8) with (7.7), we find

$$\langle e_{k-d}, Ae_{k-d}\rangle \leq \left(\frac{1}{1-\gamma_d^2}\right) \sum_{j=1}^{d} |a_{k-j}\langle r_{k-j}, p_{k-j}\rangle|. \tag{7.9}$$

One could again estimate κ using the ratio of largest to smallest eigenvalue of T_k and stop when the quantity in (7.9) is less than a given tolerance for some value of d.

Since (7.9) still involves an upper bound on the error which, for some problems, is not a very good estimate, different approaches have been considered. One of the more interesting ones involves looking at the quadratic form $r_k^H A^{-1} r_k$ as an integral and using different quadrature formulas to obtain upper and lower bounds for this integral [57]. The bounds obtained in this way appear to give very good estimates of the actual A-norm of the error. The subject of effective stopping criteria for iterative methods remains a topic of current research.

7.3. Attainable Accuracy.

Usually, the accuracy required from an iterative method is considerably less than it is capable of ultimately achieving. The important property of the method is the number of iterations or total work required to achieve a fairly modest level of accuracy. Occasionally, however, iterative methods are used for very ill-conditioned problems, and then it is important to know how the machine precision and the condition number of the matrix limit the attainable accuracy. Such analysis has been carried out for a number of iterative methods, and here we describe some results for a class of methods which includes the CG algorithm (Algorithm 2), the CGNR and CGNE algorithms (Algorithm 7), and some implementations of the MINRES, BiCG, and CGS algorithms (although not the ones recommended here). We will see in Part II of this book that the preconditioned versions of these algorithms also fall into this category.

The analysis applies to algorithms in which the residual vector r_k is updated rather than computed directly, using formulas of the form

$$x_k = x_{k-1} + a_{k-1}p_{k-1}, \quad r_k = r_{k-1} - a_{k-1}Ap_{k-1}. \tag{7.10}$$

Here p_{k-1} is some direction vector and a_{k-1} is some coefficient. It is assumed that the initial residual is computed directly as $r_0 = b - Ax_0$.

It can be shown that when formulas (7.10) are implemented in finite precision arithmetic, the difference between the true residual $b - Ax_k$ and the updated vector r_k satisfies

$$\frac{\|b - Ax_k - r_k\|}{\|A\|\,\|x\|} \leq \epsilon\, O(k) \left(1 + \max_{j \leq k} \frac{\|x_j\|}{\|x\|}\right), \tag{7.11}$$

where ϵ is the machine precision [66]. The growth in intermediate iterates, reflected on the right-hand side of (7.11), appears to play an important role in determining the size of the quantity on the left-hand side.

It is often observed numerically (but in most cases has *not* been proved) that the vectors r_k converge to zero as $k \to \infty$ or, at least, that their norms become many orders of magnitude smaller than the machine precision. In such cases, the right-hand side of (7.11) (without the $O(k)$ factor, which is an overestimate) gives a reasonable estimate of the best attainable actual residual:

$$\min_k \frac{\|b - Ax_k\|}{\|A\| \, \|x\|} \approx c \, \epsilon \, \max_k \frac{\|x_k\|}{\|x\|}, \tag{7.12}$$

where c is a moderate size constant.

The quantity on the left in (7.12) gives a measure of the *backward error* in x_k, since if this quantity is bounded by ζ, then x_k is the exact solution of a nearby problem $(A+\delta A)x_k = b$, where $\|\delta A\|/\|A\| \leq \zeta + O(\zeta^2)$. In general, the best one can hope for from a computed "solution" is that its backward error be about the machine precision ϵ, since errors of this order are made in simply representing the matrix A (either through its entries or through a procedure for computing the product of A with a given vector) on the computer.

Based on (7.12), one can expect a small backward error from iterative methods of the form (7.10), if the norms of the iterates x_k do not greatly exceed that of the true solution. For the CG algorithm for Hermitian positive definite linear systems, this is the case. It can be shown in exact arithmetic that the 2-norm of the error decreases monotonically in the CG algorithm [79]. From the inequality $\|x - x_k\| \leq \|x - x_0\|$, it follows that $\|x_k\| \leq 2\|x\| + \|x_0\|$. Assuming that $\|x_0\| \leq \|x\|$, the quantity $\max_k \|x_k\|/\|x\|$ in (7.12) is therefore bounded by about 3, and one can expect to eventually obtain an approximate solution whose backward error is a moderate multiple of ϵ. In finite precision arithmetic, the relation established in Chapter 4 between the error norms in finite precision arithmetic and exact arithmetic error norms for a larger problem can be used to show that a monotone reduction in the 2-norm of the error can be expected (to a close approximation) in finite precision arithmetic as well.

If the BiCG algorithm is implemented in the form (7.10), as in the algorithm stated in section 5.2, the norms of intermediate iterates may grow. Since no special error norm (that can be related easily to the 2-norm) is guaranteed to decrease in the BiCG algorithm, the norms of the iterates cannot be bounded a priori, and growth of the iterates will cause loss of accuracy in the final approximate solution, even assuming that the updated vectors r_k converge to zero.

An example is shown in Figure 7.1. Here A was taken to be a discretization of the convection–diffusion operator

$$-\triangle u + 40(xu_x + yu_y) - 100u$$

on the unit square with Dirichlet boundary conditions, using centered differences on a 32-by-32 mesh. The solution was taken to be $u(x,y) = x(x-1)^2y^2(y-1)^2$, and the initial guess was set to zero. The initial vector $\hat{r}_0$ was set equal to r_0.

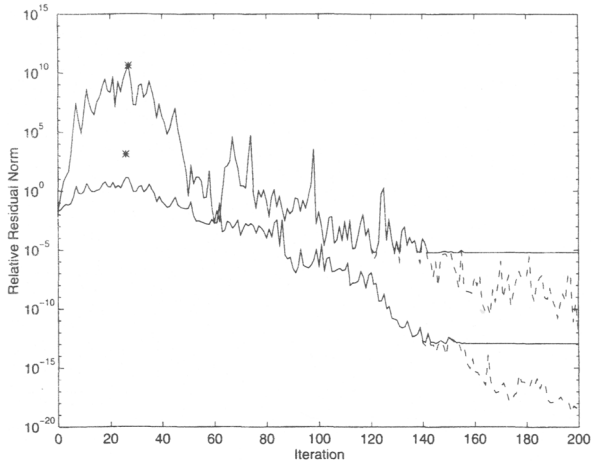

FIG. 7.1. *Actual residual norm (solid) and updated residual norm (dashed). Top curves are for CGS, bottom ones for BiCG. The asterisk shows the maximum ratio* $\|x_k\|/\|x\|$.

The lower solid line in Figure 7.1 represents the true BiCG residual norm $\|b - Ax_k\|/(\|A\|\|x\|)$, while the lower dashed line shows the updated residual norm, $\|r_k\|/(\|A\|\|x\|)$. The lower asterisk in the figure shows the maximum ratio $\|x_k\|/\|x\|$ at the step at which it occurred. The experiment was run on a machine with unit roundoff $\epsilon \approx 1.1e-16$, and the maximum ratio $\|x_k\|/\|x\|$ was approximately 10^3. As a result, instead of achieving a final residual norm of about ϵ, the final residual norm is about $1.e-13 \approx 10^3\epsilon$.

Also shown in Figure 7.1 (upper solid and dashed lines) are the results of running the CGS algorithm given in section 5.5 for this same problem. Again, there are no a priori bounds on the size of intermediate iterates, and in this case we had $\max_k \|x_k\|/\|x\| \approx 4 \cdot 10^{10}$. As a result, the final actual residual norm reaches the level $6.e-6$, which is roughly $4 \cdot 10^{10}\ \epsilon$.

Since the CGNE and CGNR algorithms are also of the form (7.10), the estimate (7.12) is applicable to them as well. Since CGNE minimizes the 2-norm of the error, it follows, as for CG, that $\|x_k\| \leq 2\|x\| + \|x_0\|$, so the backward error in the final approximation will be a moderate multiple of the machine precision. The CGNR method minimizes the 2-norm of the residual, but since it is equivalent to CG for the linear system $A^HAx = A^Hb$, it follows that the 2-norm of the error also decreases monotonically. Hence we again expect a final backward error of order ϵ.

An example for the CGNE method is shown in Figure 7.2. The matrix A was taken to be of the form $A = U\Sigma V^T$, where U and V are random orthogonal matrices and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$, with

$$\sigma_i = \kappa^{(i-1)/(n-1)}, \quad i = 1, \ldots, n, \quad \kappa = 10^4.$$

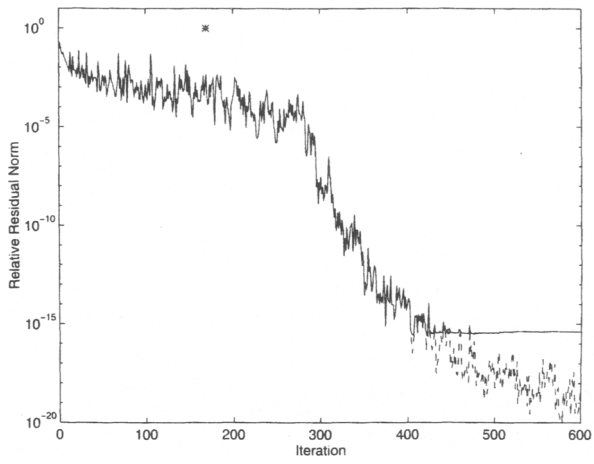

FIG. 7.2. *Actual residual norm (solid) and updated residual norm (dashed) for CGNE. The asterisk shows the maximum ratio* $\|x_k\|/\|x\|$.

For a problem of size $n = 40$, a random solution was set and a zero initial guess was used. The solid line in Figure 7.2 shows the actual residual norm $\|b - Ax_k\|/(\|A\|\|x\|)$, while the dashed line represents the updated residual norm $\|r_k\|/(\|A\|\|x\|)$. The maximum ratio $\|x_k\|/\|x\|$ is approximately 1, as indicated by the asterisk in Figure 7.2. Note that while rounding errors greatly affect the convergence rate of the method—in exact arithmetic, the exact solution would be obtained after 40 steps—the ultimately attainable accuracy is as great as one could reasonably expect—a backward error of size approximately 4ϵ. There is no loss of final accuracy due to the fact that we are (implicitly) solving the normal equations.

When a preconditioner is used with the above algorithms, the 2-norm of the error may not decrease monotonically, and then one must use other properties to establish bounds on the norms of the iterates.

It is sometimes asked whether one can accurately solve a very ill-conditioned linear system if a very good preconditioner is available. That is, suppose $\kappa(A)$ is very large but $\kappa(M^{-1}A)$ or $\kappa(M^{-1/2}AM^{-1/2})$, where M is a known preconditioning matrix, is not. We will see in Part II that the preconditioned CG algorithm, for instance, still uses formulas of the form (7.10). There is simply an additional formula, $Mz_k = r_k$, to determine a preconditioned residual z_k. The final residual norm is still given approximately by (7.12), and, unless the final residual vector is deficient in certain eigencomponents of A, this suggests an error satisfying

$$\min_k \frac{\|x - x_k\|}{\|x\|} \approx c \, \epsilon \, \kappa(A).$$

The presence of even an excellent preconditioner M (such as the LU factors

from direct Gaussian elimination) does not appear to improve this error bound for algorithms of the form (7.10).

For a discussion of the effect of rounding errors on the attainable accuracy with some different implementations, see, for example, [33, 70, 122].

7.4. Multiple Right-Hand Sides and Block Methods.

Frequently, it is desired to solve several linear systems with the same coefficient matrix but different right-hand sides. Sometimes the right-hand side vectors are known at the start and sometimes one linear system must be solved before the right-hand side for the next linear system can be computed (as, for example, in time-dependent partial differential equations). One might hope that information gained in the solution of one linear system could be used to facilitate the solution of subsequent problems with the same coefficient matrix.

We will consider only the case in which the right-hand sides are all available at the start. In that case, *block* versions of the previously described algorithms can be used. Suppose there are s right-hand sides. Then the linear systems can be written in the form

$$AX = B,$$

where X is the n-by-s matrix of solution vectors and B is the n-by-s matrix of right-hand sides.

Let A be Hermitian positive definite and consider the *block CG* algorithm. Instead of minimizing the A-norm of the error for each linear system over a single Krylov space, one can minimize

$$\mathrm{tr}[(X - X_k)^H A(X - X_k)]$$

over all X_k of the form

$$X_k \in X_0 + \mathrm{span}[R_0, AR_0, \ldots, A^{k-1}R_0].$$

That is, the approximation $x_k^{(\ell)}$ for the ℓth equation is equal to $x_0^{(\ell)}$ plus a linear combination of vectors from *all* of the Krylov spaces

$$\bigcup_{j=1}^{s} \mathrm{span}\left[r_0^{(j)}, Ar_0^{(j)}, \ldots, A^{k-1}r_0^{(j)}\right]$$

The following algorithm accomplishes this minimization.

Algorithm 8. Block Conjugate Gradient Method (Block CG)
(for Hermitian positive definite problems with multiple right-hand sides).

Given an initial guess X_0, compute $R_0 = B - AX_0$ and set $P_0 = R_0$.
For $k = 1, 2, \ldots,$

Compute AP_{k-1}.

Set $X_k = X_{k-1} + P_{k-1}a_{k-1}$, where $a_{k-1} = (P_{k-1}^H AP_{k-1})^{-1}(R_{k-1}^H R_{k-1})$.

Compute $R_k = R_{k-1} - AP_{k-1}a_{k-1}$.

Set $P_k = R_k + P_{k-1}b_{k-1}$, where $b_{k-1} = (R_{k-1}^H R_{k-1})^{-1}(R_k^H R_k)$.

It is left as an exercise to show that the following block orthogonality properties hold:

$$R_k^H R_j = 0, \quad j \neq k, \quad \text{and} \quad P_k^H AP_j = 0, \quad j \neq k.$$

As long as the matrices P_k and R_k retain full rank, the algorithm is well defined. If their columns become linearly dependent, then equations corresponding to dependent columns can be treated separately, and the algorithm will be continued with the remaining equations. A number of strategies have been developed for varying the block size. See, for example, [107, 130, 105].

A block algorithm requires somewhat more work than running s separate recurrences because s separate inner products and scalar divisions are replaced by the formation of an s-by-s matrix and the solution of s linear systems with this coefficient matrix. (Note, as usual, that it is not necessary to actually invert the matrices $P_{k-1}^H AP_{k-1}$ and $R_{k-1}^H R_{k-1}$ in the block CG algorithm, but instead one can solve s linear systems with right-hand sides given by the columns of $R_{k-1}^H R_{k-1}$ and $R_k^H R_k$, respectively. This can be accomplished by factoring the coefficient matrices using Cholesky decomposition and then backsolving with the triangular factors s times. The total work is $O(s^3)$.) If $s^3 < n$, then the extra work will be negligible.

How much improvement is obtained in the number of iterations required due to the fact that the error is minimized over a larger space? This depends on the right-hand side vectors. If all of the vectors in all of the s Krylov spaces are linearly independent, then at most n/s steps are required for the block algorithm, as compared to n for the nonblock version. Usually, however, the number of iterations required even for the nonblock iteration is significantly less than n/s. In this case, if the right-hand sides are unrelated random vectors, then the improvement in the number of iterations is usually modest. Special relations between right-hand side vectors, however, may lead to significant advantages for the block algorithm.

7.5. Computer Implementation.

If two iterative methods are both capable of generating a sufficiently accurate approximation to a system of linear equations, then we usually compare the two methods by counting operations—how many additions, subtractions, multiplications, and divisions will each method require? If the number of operations per iteration is about the same for the two methods, then we might just compare number of iterations. The method that requires fewer iterations is chosen. (One should be very careful about iteration count comparisons, however, to be sure that the algorithms being compared really do require the same amount of work per iteration!)

These are approximate measures of the relative computer time that will be required by the two algorithms, but they are only approximate. Computational time may also depend on *data locality* and potential for *parallelism*, that is, the ability to effectively use multiple processors simultaneously. These factors vary from one machine to another, so it is generally impossible to give a definitive answer to the question of which algorithm is faster.

Almost all of the algorithms discussed in this book perform vector *inner products* during the course of the iteration. If different pieces of the vectors are stored on different processors, then this requires some *global communication* to add together the inner products of the subvectors computed on the different processors. It has sometimes been thought that this would be a major bottleneck for distributed memory multiprocessors, but on today's supercomputers, this does not appear to be the case. It is a relatively small amount of data that must be passed between processors, and the bulk of the time for the iterative solver still lies in the matrix–vector multiplication and in the preconditioning step.

Sparse matrix–vector multiplication is often parallelized by assigning different rows or different blocks of the matrix to different processors, along with the corresponding pieces of the vectors on which they must operate. Many different distribution schemes are possible.

The most difficult part of an iterative method to parallelize is often the preconditioning step. This may require the solution of a sparse triangular system, which is a largely sequential operation—one solution component must be known before the next can be computed. For this reason, a number of more parallelizable preconditioners have been proposed. Examples include sparse approximate inverses (so the preconditioning step becomes just another sparse matrix–vector multiplication) and domain decomposition methods. Domain decomposition methods, to be discussed in Chapter 12, divide the physical domain of the problem into pieces and assign different pieces to different processors. The preconditioning step involves each processor solving a problem on its subdomain. In order to prevent the number of iterations from growing with the number of subdomains, however, some global communication is required. This is in the form of a coarse grid solve.

A number of parallel iterative method packages have been developed

for different machines. Some examples are described in [118, 82]. More information about the parallelization of iterative methods can be found in [117].

Exercises.

7.1. Let A be a matrix of the form $I - F$, where $F = -F^H$. Suppose the eigenvalues of A are contained in the line segment $[1 - \imath\gamma, 1 + \imath\gamma]$. It was shown by Freund and Ruscheweyh [55] that if a MINRES algorithm is applied to this matrix, then the residual at step k satisfies

$$\frac{\|r_k\|}{\|r_0\|} \leq \frac{2}{R^k + R^{k-2}}, \quad R = \frac{1 + \sqrt{1 + \gamma^2}}{\gamma}. \tag{7.13}$$

Moreover, if A contains eigenvalues throughout the interval, then this bound is sharp.

Determine a bound on the residual in the CGNR method. Will CGNR require more or fewer iterations than the MINRES method for this problem?

7.2. Use the fact that $T_k = Q_k^H A Q_k$ in the Hermitian Lanczos algorithm to show that the eigenvalues of T_k lie between the smallest and largest eigenvalues of A.

7.3. Prove the block orthogonality properties $R_k^H R_j = P_k^H A P_j = 0$, $j \neq k$, for the block CG algorithm.