Chapter 21

Krylov Subspace Methods

You should be familiar with

- Symmetric matrices and their properties
- Inner product
- Vector and matrix norms
- Matrix condition number
- Gradient of a function of n variables
- Cholesky decomposition
- Gram-Schmidt process
- Overdetermined least squares
- LU decomposition
- Poisson's equation and five-point finite difference approximation

Chapter 20 discussed the classical iterative methods, Jacobi, Gauss-Seidel, and successive overrelaxation (SOR) for solving linear systems. This chapter introduces more sophisticated methods for solving systems with a sparse coefficient matrix. After discussing the storage of sparse matrices, we introduce one of the most amazing algorithms ever developed in the field of numerical linear algebra, the conjugate gradient (CG) method. Many matrices used in the approximation of the solution to partial differential equations are symmetric positive definite. We saw an example of this situation when we discussed the heat equation in Chapter 12 and the Poisson equation in Chapter 20. The CG method is extremely effective for positive definite matrices. Rather than applying the same iteration scheme over and over, CG makes a decision at each step in order to obtain an optimal result. It uses matrix multiplication, which can be done very efficiently by taking advantage of the sparse structure of the matrices involved.

The CG method is an example of a Krylov subspace method, which is a class of algorithms that project the problem into a lower-dimensional subspace formed by powers of a matrix. We will discuss two additional Krylov subspace methods, minimum residual (MINRES) method that solves symmetric indefinite systems, and the general minimal residual (GMRES) method that applies to general sparse matrices.

In applications, the linear system is frequently ill-conditioned. It is possible using multiplication by an appropriate matrix to transform a system into one with a smaller condition number. This technique is termed preconditioning, and can yield very accurate results for a system with a poorly conditioned coefficient matrix. We will develop preconditioning techniques for CG and GMRES.

The chapter concludes with a discussion of the two-dimensional biharmonic equation, a fourth-order partial differential equation whose positive definite finite-difference matrix is ill-conditioned. Preconditioned CG gives good results for this somewhat difficult problem.

21.1 LARGE, SPARSE MATRICES

The primary use of iterative methods is for computing the solution to large, sparse systems and for finding a few eigenvalues of a large sparse matrix. Along with other problems, such systems occur in the numerical solution of partial differential equations. A good example is our discussion of a finite difference approach to the heat equation in Chapter 12. In that problem, it is necessary to solve a large tridiagonal system of equations, and we were able to use a modification of Gaussian elimination known as the Thomas algorithm. In many cases, matrices have four or more diagonals, are block structured, where nonzero elements exists in blocks throughout the matrix, or have little organized structure. The three types of large, sparse matrices are positive definite, symmetric indefinite, and nonsymmetric. The *sparsity pattern* of a matrix is a plot that shows the presence of nonzeros. Figure 21.1 shows the sparsity pattern of an actual problem for each type. An annotation specifies the computational area for which the matrix was used.

Numerical Linear Algebra with Applications. http://dx.doi.org/10.1016/B978-0-12-394435-1.00021-1
Copyright © 2015 Elsevier Inc. All rights reserved.

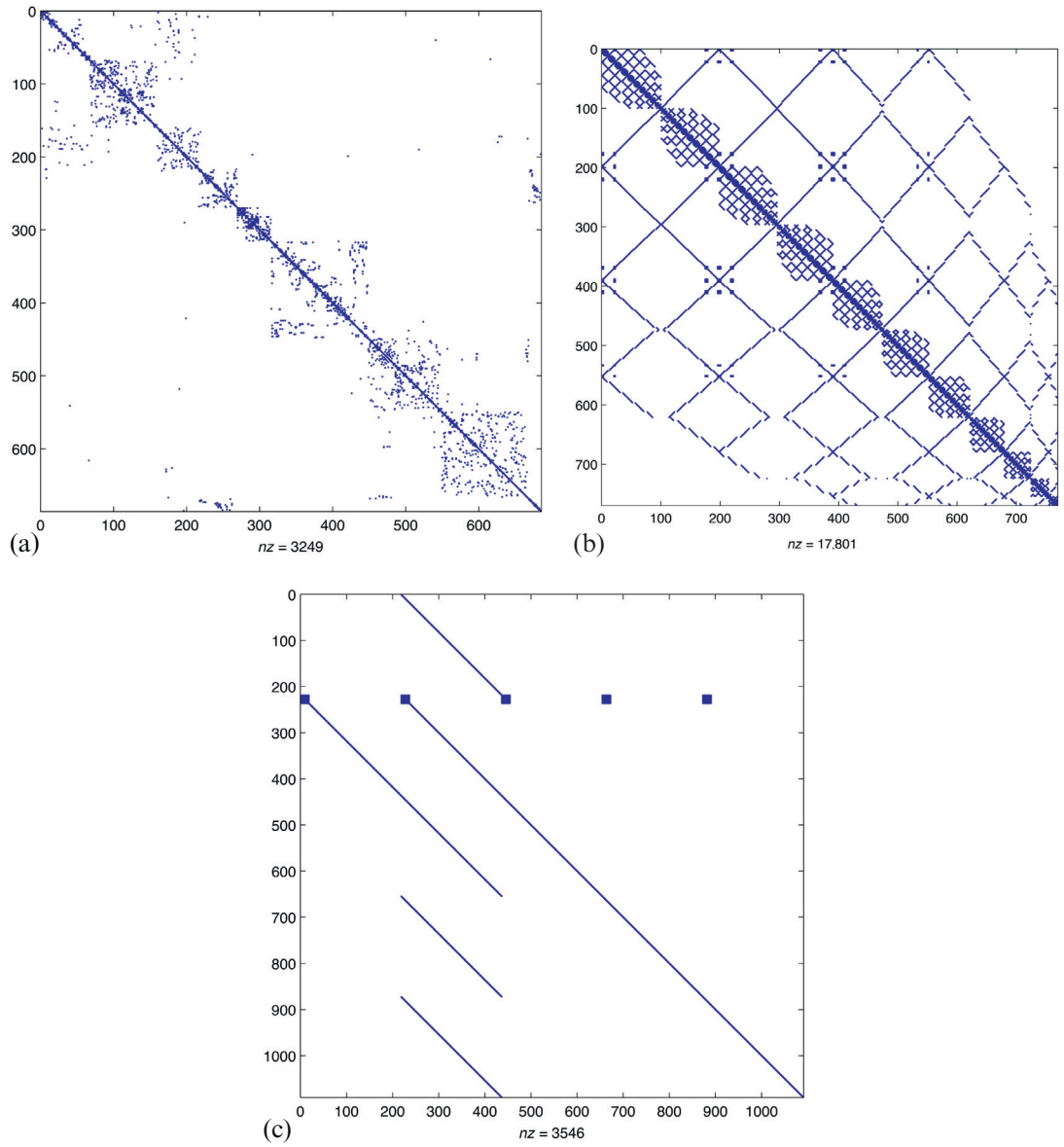


FIGURE 21.1 Examples of sparse matrices. (a) Positive definite: structural problem, (b) symmetric indefinite: quantum chemistry problem, and (c) nonsymmetric: computational fluid dynamics problem.

Algorithms for dealing with large, sparse matrices are an active area of research. The choice of a suitable method for handling a given problem depends on many factors, including the matrix size, structure, and what type of computer is used. Algorithms are different for vector computers and parallel computers. These problems can be very difficult, often requiring much experimentation before finding a suitable method of solution. The text by Saad [64] is an excellent reference for advanced material in this area.

Remark 21.1. The matrices in Figure 21.1 were obtained from the *University of Florida Sparse Matrix Collection*, maintained by Tim Davis, University of Florida, and Yifan Hu of AT&T Research [90]. It is located at http://www.cise.ufl.edu/research/sparse/matrices/. Matrices can be downloaded from the Web site or accessed by obtaining the UFgui Java interface and running the application on your system.

21.1.1 Storage of Sparse Matrices

There are a number of formats used to store sparse matrices. In each case, only the nonzero elements are recorded. We will look at the *compressed row storage (CRS) format* that stores the nonzero elements by rows. Assume we have a nonsymmetric

sparse matrix. Create three vectors: one for floating point numbers (AV) and the other two for integers (AI, AI). The AV vector stores the values of the nonzero elements of the matrix as they are traversed in a row-wise manner. The AJ vector stores the column indices of the elements in the AV vector; for instance, if $AV(j) = v$, then v is in column $AJ(j)$. The AI vector stores the locations in the AV vector that begin a row; that is, if $AI(i) = k$, then $AV(k)$ is the first nonzero element in row i. The remaining elements in row i are the elements in AV up to but not including $AI\,(i+1)$, so the number of nonzero elements in the ith row is equal to $AI\,(i+1) - AI\,(i)$. In order that this relationship will hold for the last row of the matrix, an additional entry, $nnz+1$ is added to the end of A, where nnz is the number of nonzero entries in the matrix. The vector AI must have an entry for every row. If all the elements in row k are zeros, then $AI\,(k)$ must have a value such as -1 to indicate row k has no nonzero elements.

The CRS format saves a significant amount of storage. Instead of storing n^2 elements, we need only storage locations for n elements in AI, nnz elements in both AV and AJ, and one additional entry $nnz+1$ in AI. If we assume that a machine has 8 byte floating point numbers and 4 byte integers, then the requirement for CRS storage is $8nnz + 4nnz + 4\,(n+1)$. For instance, without using sparse matrix storage a 1000×1000 matrix of double values requires $8 \times 10^6 = 8,000,000$ bytes of storage. If the matrix has only $nnz = 12,300$ nonzero elements, we need storage for only $8\,(12,300) + 4\,(12,300) + 4\,(1001) = 151,604$ bytes, which is $151,604/8,000,000$, or approximately 1.90% of the space required to store the entire matrix. If the matrix is symmetric, we only need to store the nonzero values on and above the diagonal, saving even more space.

The CRS storage format is very general. It makes no assumptions about the sparsity structure of the matrix. On the other hand, this scheme is not efficient for accessing matrices one element at a time. Inserting or removing a nonzero entry may require extensive data reorganization. However, element-by-element manipulation is rare when dealing with sparse matrices. As we will see, iterative algorithms for the solution of sparse linear systems avoid direct indexing and use matrix operations such as multiplication and addition for which there are algorithms that efficiently use the sparse matrix storage structure. There has been a significant amount of research in the area of sparse matrices, and the interested reader can consult Refs. [64–66].

Example 21.1. Consider the nonsymmetric matrix

$$A = \begin{bmatrix} 3 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 5 & 0 & 2 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 8 & 3 \end{bmatrix}.$$

The CRS format for this matrix is then specified by the arrays $\{AV, AJ, AI\}$ as follows:

AV	3	2	-1	1	5	2	6	7	12	3	1	8	3
AJ	1	5	3	6	2	4	5	1	5	4	3	6	7

AI	1	3	5	-1	8	10	11	14

Consider row 5. $AI\,(5) = 8$, $AJ\,(8) = 1$, and $AV\,(8) = 7$. The first element in row 5 is in column 1 and has value 7. There are $AI\,(7) - AI\,(6) = 2$ elements in row 5. The remaining element in row 5 is $AV\,(9) = 12$ in column $AJ\,(9) = 5$. ■

21.2 THE CG METHOD

The *CG* iteration is one of the most important methods in scientific computation, and is the algorithm of choice for solving positive definite systems. We used the SOR iteration for solving a Poisson equation in Section 20.5. The SOR iteration depends on a good choice of the relaxation parameter ω. A poor choice leads to slow convergence or divergence. The CG method does not have this problem. There are a number of approaches to developing CG. We take the approach of minimizing a quadratic function formed from the $n \times n$ matrix A. A *quadratic function* mapping $\mathbb{R}^n$ to $\mathbb{R}$ is a function of n variables $x_1, x_2, \ldots, x_n$ where each term involves the product of at most two variables. This approach allows us to incorporate illustrations that help with understanding this important algorithm.

21.2.1 The Method of Steepest Descent

To develop CG we start with the *method of steepest descent* for solving a positive definite system and then make modifications that improve the convergence rate, leading to CG. To solve the system $Ax = b$, where A is positive definite, we find the minimum of a quadratic function whose minimum is the solution to the system $Ax = b$. Let

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b, \tag{21.1}$$

where $A \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$. To minimize $\phi(x)$, set each partial derivative $\dfrac{\partial \phi}{\partial x_i} = 0$, $1 \le i \le n$, and solve for x. The *gradient* of ϕ is the vector $\nabla\phi = \left[\begin{array}{cccc} \dfrac{\partial\phi}{\partial x_1} & \dfrac{\partial\phi}{\partial x_2} & \cdots & \dfrac{\partial\phi}{\partial x_{n-1}} & \dfrac{\partial\phi}{\partial x_n}\end{array}\right]^T$, so x is a minimum if $\nabla\phi(x) = 0$. Some manipulations (Problem 21.2) will show that

$$\nabla\phi(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b.$$

Since A is symmetric,

$$\nabla\phi(x) = Ax - b, \tag{21.2}$$

and the minimum value of $\phi(x)$ occurs when $Ax = b$. Thus, finding the minimum value of $\phi(x)$ is equivalent to solving $Ax = b$. Since A is positive definite, it is nonsingular, and $x_{\min} = A^{-1}b$. Thus,

$$\nabla\phi(x_{\min}) = \frac{1}{2}x_{\min}^T AA^{-1}b - x_{\min}^T b = -\frac{1}{2}x_{\min}^T b.$$

Equation 21.1 also shows us the type of surface generated by $\phi(x)$. By the spectral theorem, there is an orthogonal matrix P such that $A = PDP^T$, where D is a diagonal matrix containing the eigenvalues of A. We have seen that an orthogonal matrix can be used to effect a change of coordinates (Section 7.1.2). Let $u = P^T(x - x_{\min})$, so $x = Pu + x_{\min}$. Substituting this into Equation 21.1, after some matrix algebra and noting that $D = P^T AP$ we obtain

$$\begin{aligned}\widetilde{\phi}(u) &= \frac{1}{2}(Pu)^T A(Pu) - \frac{1}{2}x_{\min}^T A\bar{x} = \frac{1}{2}u^T Du - \frac{1}{2}x_{\min}^T Ax_{\min}\\ &= \sum_{i=1}^{n}\lambda_i u_i^2 - \frac{1}{2}x_{\min}^T b\end{aligned}$$

Since A is positive definite, $\lambda_i > 0$, $1 \le i \le n$. For $n = 2$, $\widetilde{\phi}(u) = \lambda_1 u_1^2 + \lambda_2 u_2^2 - \frac{1}{2}x_{\min}^T b$ and is a paraboloid. In general, the orthogonal transformation $u = P^T(x - \bar{x})$ maintains lengths and angles, so $\phi(x)$ is a paraboloid in $\mathbb{R}^n$.

Example 21.2. If $Ax = b$, where $A = \left[\begin{array}{cc} 5 & 1\\ 1 & 4\end{array}\right]$ and $b = \left[\begin{array}{c} 1\\ 1\end{array}\right]$, the eigenvalues of the coefficient matrix are 3.3820 and 5.6180, so A is positive definite. Figure 21.2(a) is a graph of the paraboloid $\phi(x) = \frac{5}{2}x^2 + 2y^2 + xy - x - y$. The solution to the system is $\left[\begin{array}{cc} 0.1579 & 0.2105\end{array}\right]^T$, and we know the global minimum value is

$$-\frac{1}{2}\left[\begin{array}{cc} 0.1579 & 0.2105\end{array}\right]\left[\begin{array}{cc} 1 & 1\end{array}\right]^T = -0.1842. \quad \blacksquare$$

Before continuing to develop the method of steepest descent, we need to define some concepts. A *level set* of a real-valued function of n variables is a curve along which the function has a constant value. For a function of two variables, level sets are often called *contour lines*. They are formed when a plane parallel to the xy-plane passes through the surface and the curve formed is projected onto the xy-plane. Since $\phi(x)$ is a paraboloid, its contour lines are ellipses. Figure 21.2(b) is a graph that shows contour lines for our example function $\phi(x) = \frac{5}{2}x^2 + 2y^2 + xy - x - y$. These contour lines picture movement downward to the minimum of $\phi(x)$ located near the middle of the figure. Contour lines and the gradient are related. For a given point x, the gradient always points in the direction of greatest increase of $\phi(x)$, so $-\nabla\phi(x)$ points in the direction of greatest decrease. In addition, the gradient is always orthogonal to the contour line at the point x (Figure 21.2(b)).

In steepest descent, the minimum value $\phi(x_{\min})$ is found by starting at an initial point and descending toward $x_{\min}$ (the solution of $Ax = b$). Assume the current point is x_i, and we want to move from it to the point $x_{\min}$. Since the quadratic function ϕ decreases most rapidly in the direction of the negative gradient, Equation 21.2 specifies that $-\nabla\phi(x_i) = b - Ax_i$. Let

$$r_i = b - Ax_i$$

be the residual at point x_i. The residual itself is pointing in the direction of most rapid decrease from x_i, and is called a *direction vector*. If the residual is zero, we are done. If the residual is nonzero, move along a line specified by the direction vector, so let $x_{i+1} = x_i + \alpha_i r_i$, where α_i is a scalar. We must find α_i so that we minimize ϕ for each step. Let

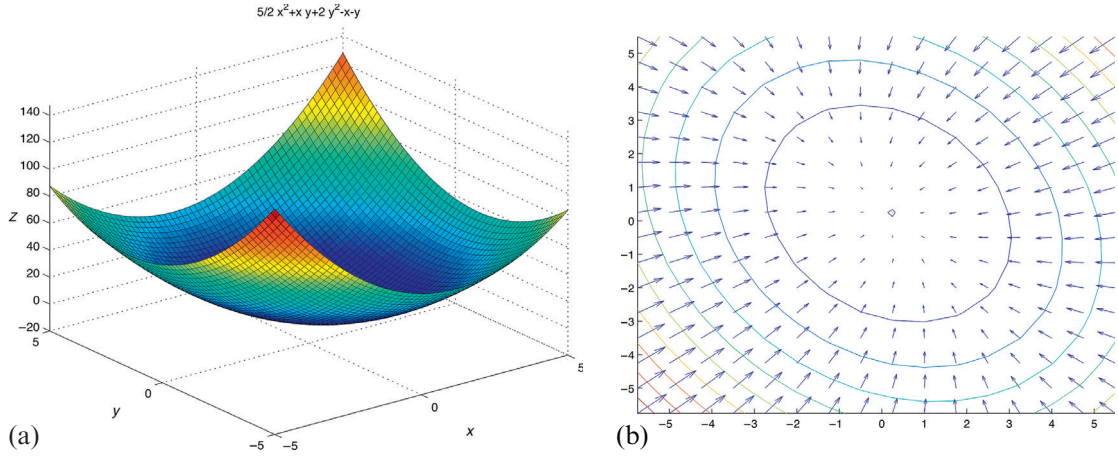


FIGURE 21.2 Steepest descent. (a) Quadratic function in steepest descent and (b) gradient and contour lines.

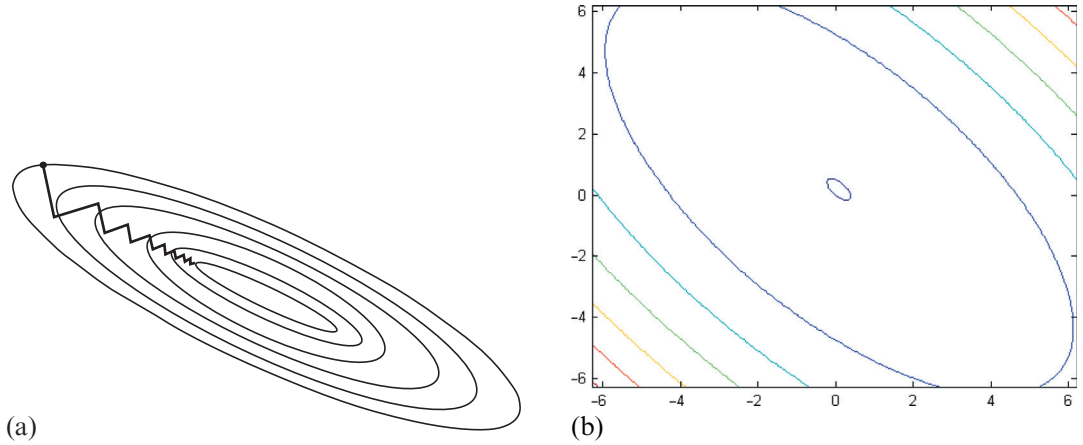


FIGURE 21.3 Steepest descent. (a) Deepest descent zigzag and (b) gradient contour lines.

$$f(\alpha_i) = \phi(x_i + \alpha_i r_i).$$

Using the symmetry of A and the commutativity of the inner product we have,

$$\begin{aligned} f(\alpha_i) &= \frac{1}{2}(x_i + \alpha_i r_i)^{\mathrm{T}} A(x_i + \alpha_i r_i) - (x_i + \alpha_i r_i)^{\mathrm{T}} b \\ &= \frac{1}{2} x_i^{\mathrm{T}} A x_i + \frac{\alpha_i^2}{2} r_i^{\mathrm{T}} A r_i + \alpha_i x_i^{\mathrm{T}} A r_i - x_i^{\mathrm{T}} b - \alpha_i r_i^{\mathrm{T}} b. \end{aligned}$$

The minimum occurs when $f'(\alpha_i) = 0$, and

$$\begin{aligned} f'(\alpha_i) &= \alpha_i r_i^{\mathrm{T}} A r_i + x_i^{\mathrm{T}} A r_i - r_i^{\mathrm{T}} b \\ &= \alpha_i r_i^{\mathrm{T}} A r_i + r_i^{\mathrm{T}} A x_i - r_i^{\mathrm{T}} b \\ &= \alpha_i r_i^{\mathrm{T}} A r_i + r_i^{\mathrm{T}} (A x_i - b) \\ &= \alpha_i r_i^{\mathrm{T}} A r_i - r_i^{\mathrm{T}} r_i. \end{aligned}$$

Thus, the minimum occurs when

$$\alpha_i = \frac{r_i^{\mathrm{T}} r_i}{r_i^{\mathrm{T}} A r_i}. \tag{21.3}$$

Step from x_i to $x_{i+1} = x_i + \alpha_i r_i$, and then use x_{i+1} and r_{i+1} to compute $x_{i+2} = x_{i+1} + \alpha_{i+1} r_{i+1}$, the next point downward toward the minimum.

Our discussion gives rise to the steepest descent algorithm.

NLALIB: The function `steepestDescent` implements Algorithm 21.1.

It can be shown that [2, pp. 625-627]

$$\left(\phi(x_i) + \frac{1}{2} b^{\mathrm{T}} A^{-1} b\right) \leq \left(1 - \frac{1}{\kappa(A)}\right)\left(\phi(x_{i-1}) + \frac{1}{2} b^{\mathrm{T}} A^{-1} b\right),$$

Algorithm 21.1 Steepest Descent

```
function STEEPESTDESCENT(A,b,x_0,tol,maxiter)
   % Solve Ax=b using the steepest descent method.
   % Input: Matrix A, right-hand side b, initial approximation x_0,
   % error tolerance tol, and maximum number of iterations maxiter.
   % Output: Approximate solution x, norm of the residual ||b - Ax||_2,
   % and the number of iterations required.
   % If the method does not converge, iter=-1.
   r_0 = b - Ax_0
   iter = 1
   i = 0
   while do (||r_i||_2 >= tol) and (iter <= maxiter)
      α_i = (r_i^T r_i) / (r_i^T A r_i)
      x_{i+1} = x_i + α_i r_i
      r_{i+1} = b - Ax_{i+1}
      i = i + 1
      iter = iter + 1
   end while
   iter = iter - 1
   if iter >= numiter then iter = -1
   end if
end function
```

which says that the algorithm converges no matter what initial value we choose. If $\kappa(A)$ is large, then $\left(1 - \frac{1}{\kappa(A)}\right)$ is close to 1, and convergence will be slow. We can see this geometrically. For any $n \times n$ matrix, successive search directions, r_i, are orthogonal, as we can verify by a calculation. Noting that $Ax_i = b - r_i$,

$$\begin{aligned} r_i^T r_{i+1} = \; & r_i^T (b - Ax_{i+1}) = r_i^T [b - A(x_i + \alpha_i r_i)] = \\ & r_i^T b - r_i^T (Ax_i + \alpha_i A r_i) = \\ & r_i^T b - r_i^T [(b - r_i) + \alpha_i A r_i] = \\ & r_i^T b - r_i^T b + r_i^T r_i - \alpha_i r_i^T A r_i = \\ & r_i^T r_i - \alpha_i r_i^T A r_i. \end{aligned}$$

From Equation 21.3

$$r_i^{\mathrm{T}} r_i - \alpha_i r_i^{\mathrm{T}} A r_i = 0,$$

so $r_i^{\mathrm{T}} r_{i+1} = 0$. If we are at x_i in the descent to the minimum, the negative of the gradient vector is orthogonal to the contour line $\phi(x_i) = k_i$. The next search direction r_{i+1} is orthogonal to r_i and orthogonal to the contour line $\phi(x_{i+1}) = k_{i+1}$. As illustrated in Figure 21.3(a), for $n = 2$ the approach to the minimum follows a zigzag pattern. Steepest descent always makes a turn of 90° as it moves toward the minimum, and it could very well be that a different turn is optimal. In $\mathbb{R}^2$, if the eigenvalues of A are $\lambda_1 = \lambda_2$, the contour lines are circles; otherwise, they are ellipses. The eigenvalues of a positive definite matrix are the same as its singular values (Problem 21.8), so the shape of the ellipses depends on the ratio of the largest to the smallest eigenvalue, which is the condition number. If the condition number of A is small, the ellipses are close to

circles, and the steepest descent algorithm steps from contour to contour, quickly moving toward the center. However, as the condition number gets larger, the ellipses become long and narrow. The paraboloid $\phi(x)$ has a steep, narrow canyon near the minimum, and the values of x_i move back and forth across the walls of the canyon, moving down very slowly, and requiring a great many iterations to converge to the minimum. This is illustrated in Figure 21.3(b) for the matrix $A = \begin{bmatrix} 5 & 3 \\ 3 & 4 \end{bmatrix}$. Note that the contour lines are more eccentric ellipses than those for the matrix $A = \begin{bmatrix} 5 & 1 \\ 1 & 4 \end{bmatrix}$ of Example 21.2.

Example 21.3. The MATLAB function `steepestDescent` in the software distribution returns the approximate solution and the number of iterations required. Apply the method to the matrices $A = \begin{bmatrix} 5 & 1 \\ 1 & 4 \end{bmatrix}$, $\kappa(A) = 1.6612$ and $B = \begin{bmatrix} 7 & 6.99 \\ 6.99 & 7 \end{bmatrix}$, $\kappa(B) = 1399$. In both cases, $b = \begin{bmatrix} 46.463 & 17.499 \end{bmatrix}^{\mathrm{T}}$ and $x_0 = \begin{bmatrix} 5 & 5 \end{bmatrix}^{\mathrm{T}}$. For matrix A, 21 iterations were required to attain an error tolerance of 1.0×10^{-12}; however, matrix B required 8462 iterations for the same error tolerance. ■

21.2.2 From Steepest Descent to CG

The method of steepest descent does a line search based on the gradient by computing $x_{i+1} = x_i + \alpha_i r_i$, where the r_i are the residual vectors. The CG method computes

$$x_{i+1} = x_i + \alpha_i p_i,$$

where the direction vectors $\{p_i\}$ are chosen so $\{x_i\}$ much more accurately and rapidly descends toward the minimum of $\phi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$. Repeat the calculations that lead to Equation 21.3, replacing r_i by p_i, to obtain

$$\alpha_i = \frac{p_i^{\mathrm{T}} r_i}{p_i^{\mathrm{T}} A p_i}. \tag{21.4}$$

Before showing how to choose $\{p_i\}$, we must introduce the A-norm.

Definition 21.1. If A is an $n \times n$ positive definite matrix and $x, y \in \mathbb{R}^n$, then $\langle x, y \rangle_A = x^{\mathrm{T}}Ay$ is an inner product, and if $\langle x, y \rangle_A = 0$, x and y are said to be *A-conjugate*. The corresponding norm, $\|x\|_A = \sqrt{x^{\mathrm{T}}Ax}$ is called the *A-norm* or the *energy norm*.

Remark 21.2. We leave the fact that $\langle \cdot, \cdot \rangle_A$ is an inner product to Problem 21.3. We use the term energy norm because the term $\frac{1}{2}x^{\mathrm{T}}Ax = \frac{1}{2}\|x\|_A^2$ represents physical energy in many problems.

The function $\phi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$ can be written using the A-norm as follows:

$$\phi(x) = \frac{1}{2}\|x\|_A^2 - \langle x, b \rangle,$$

where $\langle x, b \rangle$ is the Euclidean inner product. If $\bar{x}$ is the approximation to the minimum of $\phi(x)$ and $e = x - \bar{x}$ is the error, then completing the square gives (Problem 21.4)

$$\phi(x) = \frac{1}{2}e^{\mathrm{T}}Ae - \frac{1}{2}\|x\|_A^2 = \frac{1}{2}\|e\|_A^2 - \frac{1}{2}\|\bar{x}\|_A^2.$$

Thus, to minimize ϕ, we must minimize the A-norm of the error. The steepest descent algorithm minimizes the 2-norm of the error along a gradient line at each step. This is one-dimensional minimization. The CG method uses information from past steps so that it can minimize over higher-dimensional subspaces. As we move from x_{i-1} to x_i, the algorithm minimizes over an i-dimensional subspace. Another way of putting it is that instead of minimizing over a line we minimize over a plane ($n = 2$) or a hyperplane ($n > 2$). The approximations $x_{i+1} = x_i + \alpha_i p_i$ do not zigzag toward the minimum of $\phi(x)$, but follow a much better path. In the following figure, from point A the thin line represents the zigzag of steepest descent and requires four steps. The CG iteration requires only one step (Figure 21.4).

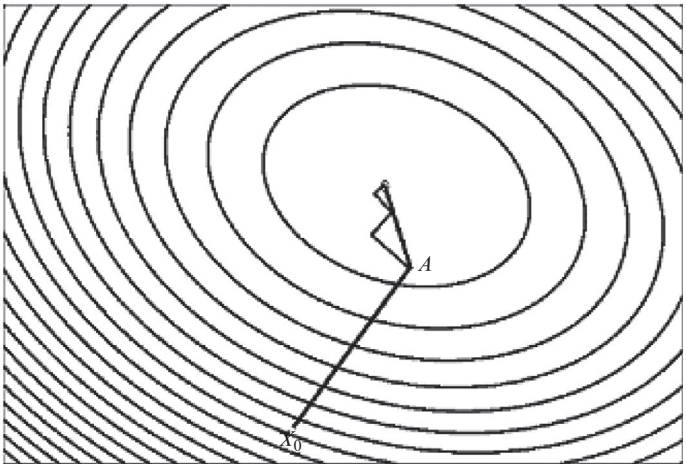


FIGURE 21.4 2-Norm and A-norm convergence.

For the steepest descent method, successive search directions r_i are orthogonal. Recall that the Gram-Schmidt process takes a set of linearly independent vectors and produces a set of mutually orthogonal unit vectors relative to the 2-norm that span the same subspace as the original set. To find $\{p_i\}$, we apply the Gram-Schmidt process to the residual vectors $\{r_i\}$ but use the energy inner product and norm. Start with $r_0 = b - Ax_0$ just like we did with steepest descent. The set of vectors $\{p_i\}$ must be A-conjugate ($p_i^{\mathrm{T}} A p_j = 0$, $i \neq j$), but it is not necessary that they be unit vectors. If we skip the normalization step of Gram-Schmidt at step i we have

$$p_i = r_i - \sum_{j=1}^{i-1} c_{j,i} p_j, \tag{21.5}$$

where

$$c_{ji} = \frac{r_i^{\mathrm{T}} A p_j}{p_j^{\mathrm{T}} A p_j}.$$

The purpose of Equation 21.5 is to determine the next search direction from the current residual and the previous search directions, thus doing far better than steepest descent that uses only the last search direction.

The computation of α_i defined in Equation 21.4 can be simplified by building some additional machinery. The following technical lemmas contains some formulas we will need.

Lemma 21.1. *Let x be the true solution to $Ax = b$, and the error at step i defined by $e_i = x - x_i$. Then,*

1. $r_{i+1} = r_i - \alpha_i A p_i$
2. $A e_{i+1} = r_{i+1}$
3. $e_{i+1} = e_i - \alpha_i p_i$.

Proof. Note that $r_{i+1} = b - Ax_{i+1} = b - A(x_i + \alpha_i p_i) = r_i - \alpha_i A p_i$, and we have property 1.
$A e_{i+1} = A(x - x_{i+1}) = A(x) - A x_{i+1} = b - A x_{i+1} = r_{i+1}$, which proves property 2.
For the proof of property 3, note that $e_{i+1} = x - x_{i+1} = x - (x_i + \alpha_i p_i) = e_i - \alpha_i p_i$. □

Using Lemma 21.1, we can develop a result that will further aid in simplifying the computation of α_i. It says that the error at step i+1 is A-conjuate to the previous direction vector p_i.

Lemma 21.2. $\langle e_{i+1}, p_i \rangle_A = 0$.

Proof. From property 1 of Lemma 21.1, $r_{i+1}^T p_i = r_i^T p_i - \alpha_i (A p_i)^T p_i = r_i^T p_i - \alpha_i \left(p_i^T A p_i\right) = 0$ by substituting the value of α_i. Now, multiply the equation in property 2 by p_i^{T} on the left to obtain

$$p_i^{\mathrm{T}} A e_{i+1} = \langle e_{i+1}, p_i \rangle_A = p_i^{\mathrm{T}} r_{i+1} = r_{i+1}^{\mathrm{T}} p_i = 0.$$ □

We are now in a position to simplify the values of α_i. The algorithm begins with the assignment $p_0 = r_0 = b - Ax_0$ and the computation of x_1. Lemma 21.2 says that $\langle e_1, p_0\rangle_A = 0$. In the next step, the Gram-Schmidt process creates p_1 so that $\langle p_1, p_0\rangle_A = 0$, determining x_2. Again, from Lemma 21.2, $\langle e_2, p_1\rangle_A = 0$. At this point we have $\langle e_1, p_0\rangle_A = 0$ and $\langle e_2, p_1\rangle_A = 0$. Noting that $\langle p_1, p_0\rangle = 0$, apply property 3 of Lemma 21.1 to obtain

$$\langle e_2, p_0\rangle_A = \langle e_1 - \alpha_1 p_1, p_0\rangle_A = \langle e_1, p_0\rangle_A - \alpha_1 \langle p_1, p_0\rangle_A = 0.$$

Thus, e_2 is A-orthogonal to both p_0 and p_1. Continuing in this fashion, we see that e_i is A-orthogonal to $\{p_0, p_1, \ldots, p_{i-1}\}$, or

$$\langle e_i, p_j\rangle_A = 0, \quad j < i. \tag{21.6}$$

Now,

$$\langle e_i, p_j\rangle_A = p_j^{\mathrm{T}} A e_i = p_j^{\mathrm{T}} (Ax - Ax_i) = p_j^{\mathrm{T}} (b - Ax_i) = \langle r_i, p_j\rangle$$

and from Equation 21.6 it follows that:

$$\langle r_i, p_j\rangle = 0, \quad j < i. \tag{21.7}$$

We can obtain two useful properties from Equation 21.7. Using Equation 21.5, take the inner product of p_j and r_i, $j < i$, and

$$r_i^{\mathrm{T}} p_j = r_i^{\mathrm{T}} r_j - \sum_{k=1}^{j-1} c_{kj} r_i^{\mathrm{T}} p_k = r_i^{\mathrm{T}} r_j = 0.$$

Thus,

$$r_i^{\mathrm{T}} r_j = 0, \quad j < i. \tag{21.8}$$

and all residuals are orthogonal to the previous ones. Similarly

$$\langle p_i, r_i\rangle = \|r_i\|_2^2 - \sum_{j=1}^{i-1} c_{ji} \langle r_i, p_j\rangle = \|r_i\|_2^2 \tag{21.9}$$

from Equation 21.7. Using Equation 21.9 in Equation 21.4 gives us the final value for α_i.

$$\alpha_i = \frac{\|r_i\|_2^2}{\langle Ap_i, p_i\rangle} \tag{21.10}$$

The evaluation of formula 21.5 seems expensive, and it appears that we must retain all the previous search directions. Here is where the CG algorithm is remarkable. We get the benefit of a combination of search directions in Equation 21.5 by computing only $c_{i-1,i}$, the coefficient of the last direction p_{i-1}! Take the inner product of r_i and the relationship in property 1 of Lemma 21.1 to obtain $r_i^{\mathrm{T}} r_{j+1} = r_i^{\mathrm{T}} r_j - \alpha_j r_i^{\mathrm{T}} A p_j$, and so

$$\alpha_j r_i^{\mathrm{T}} A p_j = r_i^{\mathrm{T}} r_j - r_i^{\mathrm{T}} r_{j+1}. \tag{21.11}$$

Equations 21.8 and 21.11 give us the results

$$r_i^{\mathrm{T}} A p_j = \begin{cases} \dfrac{1}{\alpha_i} r_i^{\mathrm{T}} r_i & j = i \\ -\dfrac{1}{\alpha_{i-1}} r_i^{\mathrm{T}} r_i & j = i-1 \\ 0 & j < i-1 \end{cases} \tag{21.12}$$

Recall that the Gram-Schmidt constants are $c_{ji} = \dfrac{r_i^{\mathrm{T}} A p_j}{p_j^{\mathrm{T}} A p_j}$. Using Equations 21.10 and 21.12,

$$c_{i-1,i} = \frac{r_i^{\mathrm{T}} A p_{i-1}}{p_{i-1}^{\mathrm{T}} A p_{i-1}} = -\frac{1}{\alpha_{i-1}} \frac{r_i^{\mathrm{T}} r_i}{p_{i-1}^{\mathrm{T}} A p_{i-1}} = -\frac{\langle Ap_{i-1}, p_{i-1}\rangle}{\|r_{i-1}\|_2^2} \frac{r_i^{\mathrm{T}} r_i}{\langle Ap_{i-1}, p_{i-1}\rangle} = -\frac{\|r_i\|_2^2}{\|r_{i-1}\|_2^2}.$$

From Equation 21.12, $c_{j,i} = 0, j < i-1$. Thus, all the coefficients $\{c_{ji}\}$ in Equation 21.5 are zero except $c_{i-1,i}$. By defining

$$\beta_i = \frac{\|r_i\|_2^2}{\|r_{i-1}\|_2^2},$$

we see that

$$p_i = r_i + \beta_i p_{i-1}. \tag{21.13}$$

We can now give the CG algorithm.

Algorithm 21.2 Conjugate Gradient

```
function cg(A,b,x1,tol,numiter)
   r_0 = b − Ax_1
   p_0 = r_0
   for i=1:numiter do
      α_{i−1} = (r_{i−1}^T r_{i−1}) / (p_{i−1}^T A p_{i−1})
      x_i = x_{i−1} + α_{i−1} p_{i−1}
      r_i = r_{i−1} − α_{i−1} A p_{i−1}
      if ‖r_i‖_2 < tol then
         iter=i
         return[x_i, iter]
      end if
      β_i = (r_i^T r_i) / (r_{i−1}^T r_{i−1})
      p_i = r_i + β_i p_{i−1}
   end for
   iter=-1 return[x_numiter, iter]
end function
```

NLALIB: The function `cg` implements Algorithm 21.2.

Example 21.4. Let $A = \begin{bmatrix} 5 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 6 \end{bmatrix}$, $b = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^{\mathrm{T}}$, $x_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$, trace two iterations of Algorithm 21.2, then apply the function `cg` to compute the solution. Note that only one more iteration is required to obtain a very accurate result:

$$r_0 = p_0 = b - Ax_0 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}^{\mathrm{T}}$$

$i = 1$:

$$\alpha_0 = (r_0^{\mathrm{T}} r_0) / (p_0^{\mathrm{T}} A p_0) = 0.1443 \quad x_1 = x_0 + \alpha_0 p_0 = \begin{bmatrix} 0.1443 & 0.2887 & 0.4330 \end{bmatrix}^{\mathrm{T}}$$

$$r_1 = r_0 - \alpha_0 A p_0 = \begin{bmatrix} -0.4433 & 0.2680 & -0.0309 \end{bmatrix} \quad \beta_1 = (r_1^{\mathrm{T}} r_1) / (r_0^{\mathrm{T}} r_0) = 0.0192$$

$$p_1 = r_1 + \beta_1 p_0 = \begin{bmatrix} -0.4241 & 0.3065 & 0.0268 \end{bmatrix}^{\mathrm{T}}$$

$i = 2$:

$$\alpha_1 = (r_1^{\mathrm{T}} r_1) / (p_1^{\mathrm{T}} A p_1) = 0.2659 \quad x_2 = x_1 + \alpha_1 p_1 = \begin{bmatrix} 0.0316 & 0.3702 & 0.4401 \end{bmatrix}^{\mathrm{T}}$$

```
>> [x r iter] = cg(A,b,x0,1.0e-15,10)
x =
    0.0374
    0.3832
    0.4299
```

```
r =
  1.0839e-016
iter =
  3
```

■

21.2.3 Convergence

We will not prove convergence of the CG method, but will state convergence theorems. For proofs, see Refs. [23, Chapter 8] and [27, Chapter 9].

Theorem 21.1. *If exact arithmetic is performed, the CG algorithm applied to an $n \times n$ positive definite system $Ax = b$ converges in n steps or less.*

CG converges after n iterations, so why should we care about convergence analysis? CG is commonly used for problems so large it is not feasible to run even n iterations. In practice, floating point errors accumulate and cause the residual to gradually lose accuracy and the search vectors to lose A-orthogonality, so it is not realistic to think about an exact algorithm.

Theorem 21.2 indicates how fast the CG method converges. For a proof, see Ref. [1, pp. 312-314]. The proof uses Chebyshev polynomials, which are defined and discussed in Appendix C.

Theorem 21.2. *Let the CG method be applied to a symmetric positive definite system $Ax = b$, where the condition number of A is κ. Then the A-norms of the errors satisfy*

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k.$$

The term $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ can be written as $1 - \frac{2}{\sqrt{\kappa}+1}$. If $\sqrt{\kappa}$ is reasonably small, the algorithm will converge quickly, but if $\sqrt{\kappa}$ is large, convergence slows down. Since κ depends on the largest and smallest eigenvalue of A, if the eigenvalues are clustered closely together, convergence will be good. If the eigenvalues of A are widely separated, CG convergence will be slower. Each iteration of the CG method requires $O\left(n^2\right)$ flops, so n iterations will cost $O\left(n^3\right)$ flops, which is the same as the Cholesky decomposition. However, practice has shown that convergence using floating point arithmetic often occurs in less than n iterations.

We conclude this section with a comparison of the convergence rates of steepest descent and CG. Of course, we expect CG to outperform steepest descent.

Example 21.5. The software distribution contains a sparse 300×300 symmetric positive definite matrix CGDES in the file `CGDES.mat`. It has a condition number of 30.0. Figure 21.5(a) is a density plot showing the location of its 858 nonzeros. If b is a random column vector, and $x_0 = \begin{bmatrix} 1 & 1 & \ldots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, Figure 21.5(b) is a graph of the log of the residual norms for the steepest descent and CG algorithms after $n = 5, 6, \ldots, 50$ iterations. Note the superiority of CG. After 50 iterations, CG has attained a residual of approximately 10^{-7}, but steepest descent is still moving very slowly downward toward a more accurate solution. ■

21.3 PRECONDITIONING

Since the convergence of an iterative method depends on the condition number of the coefficient matrix, it is often advantageous to use a *preconditioner matrix M that* transforms the system $Ax = b$ to one having the same solution but that can solved more rapidly and accurately. The idea is to choose M so that

- M is not too expensive to construct.
- M is easy to invert.
- M approximates A so that the product of A and M^{-1} "is close" to I ($\eta(I) = 1$).
- the preconditioned system is more easily solved with higher accuracy.

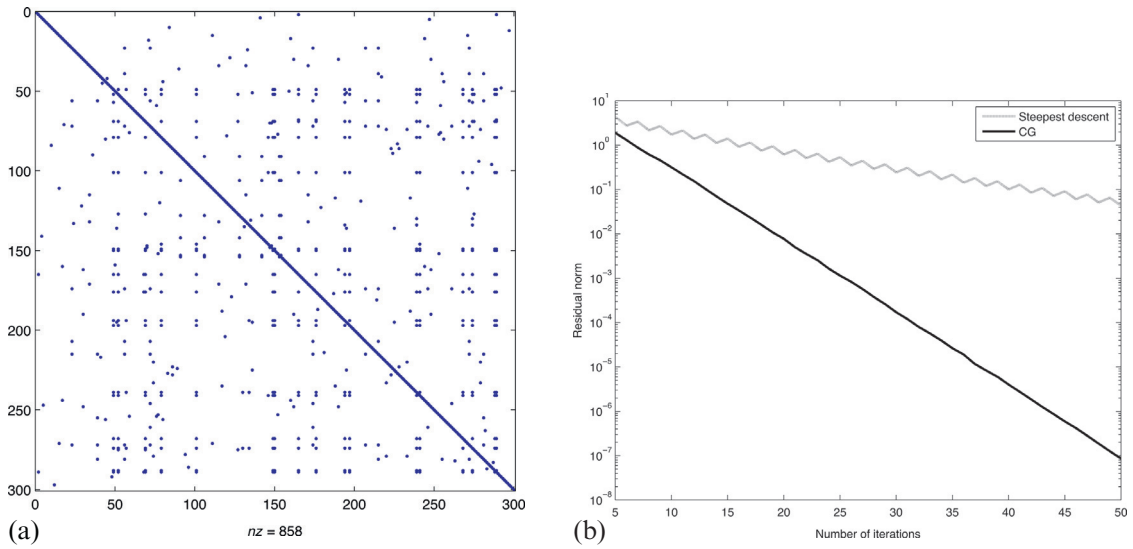


FIGURE 21.5 CG vs. steepest descent. (a) Density plot for symmetric positive definite sparse matrix CGDES and (b) residuals of CG and steepest descent.

If M is a nonsingular matrix, then the system

$$M^{-1}Ax = M^{-1}b \tag{21.14}$$

has the same solution as $Ax = b$. The solution to the system 21.14 depends on the coefficient matrix $M^{-1}A$ instead of A, and we hope that $\kappa\left(M^{-1}A\right)$ is much smaller than $\kappa\,(A)$.

The computation of $M^{-1}A$ is rarely done. The product may be a dense matrix, and we lose all the advantages of sparsity. Also, there is the expense and potential inaccuracy of computing $M^{-1}A$. We can form the product $M^{-1}Ax$ indirectly as follows:

a. Let $z = M^{-1}Ax$, and compute $w = Ax$.
b. To find z, solve $Mz = w$.

Compute the right-hand side of Equation 21.14 by letting $y = M^{-1}b$ and solving $My = b$. Such a matrix M is called a *left preconditioner*.

A matrix M can also be a *right preconditioner* by computing AM^{-1} in hope that its condition number is much smaller than that of A. In this case, form the equivalent system

$$\left(AM^{-1}\right)Mx = b,$$

and let $u = Mx$, so we need to solve

$$AM^{-1}u = b.$$

If M is of the factored form $M = M_L M_R$, then we can use a *split preconditioner*. First write the system as

$$M_L^{-1}A\left(M_R^{-1}M_R\right)x = M_L^{-1}b,$$

and then solve the system

$$M_L^{-1}AM_R^{-1}u = M_L^{-1}b,$$

where $u = M_R x$.

Remark 21.3. Note that

$$M^{-1}A = M^{-1}A\left(M^{-1}M\right) = M^{-1}\left(AM^{-1}\right)M$$
$$M_R^{-1}M_L^{-1}AM_R^{-1}M_R = (M_L M_R)^{-1}A = M^{-1}A,$$

and all the matrices for preconditioning are similar, so they have the same eigenvalues. If the eigenvalues of one of these matrices, say Mat, are close to 1 and $\|\text{Mat} - I\|_2$ is small, then the preconditioned system will converge quickly. If a preconditioner does not satisfy this criterion, its distribution of eigenvalues may be favorable to fast convergence.

21.4 PRECONDITIONING FOR CG

We will discuss two methods of finding a preconditioner for CG. In each case, we apply the CG method to the preconditioned system in the hope that the CG iteration will converge faster. Of course, in addition to wanting a system with a smaller condition number, M must be chosen so that the preconditioned system is positive definite. The two methods we will present are the incomplete Cholesky decomposition and the construction of an SSOR preconditioner.

21.4.1 Incomplete Cholesky Decomposition

For the moment, assume we have a positive definite left preconditioner matrix M. M has a Cholesky decomposition $M = R^{\mathrm{T}}R$. We will use this decomposition to obtain an equivalent system $\overline{A}\overline{x} = \overline{b}$, where $\overline{A}$ is positive definite so the CG method applies.

$$
\begin{aligned}
M^{-1}Ax &= M^{-1}b \\
\left(R^{\mathrm{T}}R\right)^{-1}Ax &= \left(R^{\mathrm{T}}R\right)^{-1}b \\
R^{-1}\left(R^{\mathrm{T}}\right)^{-1}Ax &= R^{-1}\left(R^{\mathrm{T}}\right)^{-1}b \\
\left(R^{\mathrm{T}}\right)^{-1}A\left(R^{-1}R\right)x &= \left(R^{\mathrm{T}}\right)^{-1}b \\
\left(\left(R^{\mathrm{T}}\right)^{-1}AR^{-1}\right)Rx &= \left(R^{\mathrm{T}}\right)^{-1}b
\end{aligned}
$$

System 21.15 is equivalent to the original system $Ax = b$.

$$
\overline{A}\overline{x} = \overline{b}, \quad \text{where } \overline{A} = \left(R^{-1}\right)^{\mathrm{T}}AR^{-1},\ \overline{x} = Rx,\ \overline{b} = \left(R^{-1}\right)^{\mathrm{T}}b \tag{21.15}
$$

The fact that $\overline{A}$ is symmetric is left to the exercises. Note that $R^{-1}x = 0$ only if $x = 0$. Thus, $\overline{A}$ is positive definite because

$$
x^{\mathrm{T}}\left(R^{-1}\right)^{\mathrm{T}}AR^{-1}x = \left(R^{-1}x\right)^{\mathrm{T}}A\left(R^{-1}x\right) > 0, \quad x \neq 0.
$$

Use the CG method to solve $\overline{A}\overline{x} = \overline{b}$. After obtaining $\overline{x}$, find x by solving $Rx = \overline{x}$.

The *incomplete Cholesky decomposition* is frequently used for preconditioning the CG method. As stated in the introduction, iterative methods are applied primarily to large, sparse systems. However, the Cholesky factor R used in system 21.15 is usually less sparse than M. Figure 21.6(a) shows the distribution of nonzero entries in a 685×685 positive definite sparse matrix, `POWERMAT.mat`, used in a power network problem, and Figure 21.6(b) shows the distribution in its Cholesky factor. Note the significant loss of zeros in the Cholesky factor.

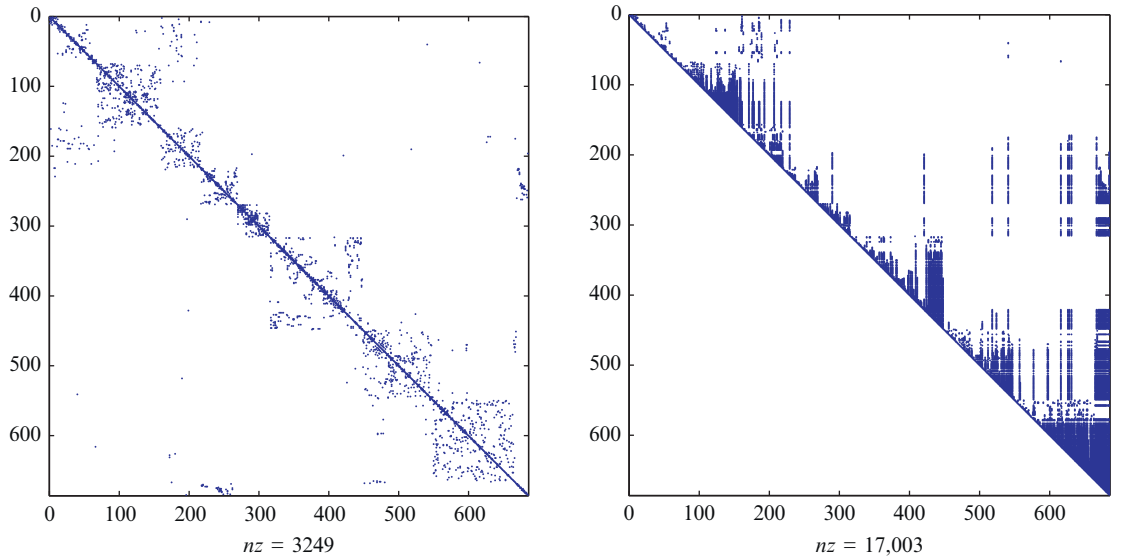
The incomplete Cholesky decomposition is a modification of the original Cholesky algorithm. If an element a_{ij} off the diagonal of A is zero, the corresponding element r_{ij} is set to zero. The factor returned, R, has the same distribution of nonzeros as A above the diagonal. Form $M = R^{\mathrm{T}}R$ from a modified Cholesky factor of A with the hope that the condition number of $M^{-1}A$ is considerably smaller than that of A. The function `icholesky` in the software distribution implements it with the calling sequence `R = icholesky(A)`. Implementation involves replacing the body of the inner for loop with

```
if A(i,j) == 0
   R(i,j) = 0;
else
    R(i,j) = (A(i,j) - sum(R(1:i-1,i).*R(1:i-1,j)))/R(i,i);
end
```

We can directly apply CG to system 21.15 by implementing the following statements.

Initialize

$$
\begin{aligned}
\overline{r_0} &= \overline{b} - \overline{A}\overline{x_1} \\
\overline{p_0} &= \overline{r_0}
\end{aligned}
$$



(a) Distribution of nonzero entries in POWERMAT (b) Distribution of nonzero entries in the Cholesky factor

FIGURE 21.6 Cholesky decomposition of a sparse symmetric positive definite matrix.

Loop

$$\begin{aligned}
\overline{\alpha_{i-1}} &= (\overline{r_{i-1}}^{\mathrm{T}}\overline{r_{i-1}})/(\overline{p_{i-1}}^{\mathrm{T}}\overline{A}\overline{p_{i-1}}) \\
\overline{x_i} &= \overline{x_{i-1}} + \overline{\alpha_{i-1}}\overline{p_{i-1}} \\
\overline{r_i} &= \overline{r_{i-1}} - \overline{\alpha_{i-1}}\overline{Ap_{i-1}} \\
\overline{\beta_i} &= (\overline{r_i}^{\mathrm{T}}\overline{r_i})/(\overline{r_{i-1}}^{\mathrm{T}}\overline{r_{i-1}}) \\
\overline{p_i} &= \overline{r_i} + \overline{\beta_i}\overline{p_{i-1}}
\end{aligned}$$

Upon completion, solve the upper triangular system $Rx = \bar{x}$. However, this is not an efficient implementation. Improvement can be made by determining relationships between the transformed variables and the original ones.

We can express the residual, $\overline{r_i}$, in terms of the original residual, r_i, by

$$\overline{r_i} = \left(R^{-1}\right)^{\mathrm{T}} r_i \tag{21.16}$$

using the steps

$$\begin{aligned}
\overline{r_i} &= \overline{b} - \overline{A}\overline{x_i} \\
&= \left(R^{-1}\right)^{\mathrm{T}} b - \left[\left(R^{-1}\right)^{\mathrm{T}} AR^{-1}\right] Rx_i \\
&= \left(R^{-1}\right)^{\mathrm{T}} b - \left(R^{-1}\right)^{\mathrm{T}} Ax_i \\
&= \left(R^{-1}\right)^{\mathrm{T}} (b - Ax_i) \\
&= \left(R^{-1}\right)^{\mathrm{T}} r_i.
\end{aligned}$$

Make a change of variable by letting $\overline{p_i} = Rp_i$. This leads to

$$\langle \overline{p_i}, \overline{p_j} \rangle_A = \langle p_i, p_j \rangle_A \tag{21.17}$$

as follows:

$$\begin{aligned}
\langle \overline{p_i}, \overline{p_j} \rangle_A &= \langle \overline{A}\overline{p_i}, \overline{p_j} \rangle \\
&= \left\langle \left(R^{-1}\right)^{\mathrm{T}} AR^{-1} Rp_i, Rp_j \right\rangle
\end{aligned}$$

$$
\begin{aligned}
&= \left\langle \left(R^{-1}\right)^{\mathrm{T}} A p_i, R p_j \right\rangle \\
&= p_i^{\mathrm{T}} A R^{-1} R p_j \\
&= p_i^{\mathrm{T}} A p_j \\
&= \langle p_i, p_j \rangle_A .
\end{aligned}
$$

By applying Equations 21.16 and 21.17, it follows that (Problem 21.11):

$$\overline{\alpha_{i-1}} = \frac{r_{i-1}^{\mathrm{T}} M^{-1} r_{i-1}}{p_{i-1}^{\mathrm{T}} A p_{i-1}} \tag{21.18}$$

$$\overline{\beta_i} = \frac{r_i^{\mathrm{T}} M^{-1} r_i}{r_{i-1}^{\mathrm{T}} M^{-1} r_{i-1}} \tag{21.19}$$

Applying the identities to the CG algorithm for the computation of $\bar{x}$, gives the preconditioned CG algorithm 21.3. Note that the algorithm requires the computation of the incomplete Cholesky factor R and then the solution of systems $Mz_i = r_i$, which can be done using the incomplete Cholesky factor (Section 13.3.3). It is never necessary to compute M or M^{-1}.

Algorithm 21.3 Preconditioned Conjugate Gradient

```
function PRECG(A,b,x1,tol,numiter)

  R=icholesky(A)
  r_1 = b − Ax_1
  z_1 = cholsolve(R, r_1)
  p_1 = z_1
  for i = 1:numiter do
    α_i = (z_i^T r_i) / (p_i^T A p_i)
    x_{i+1} = x_i + α_i p_i
    r_{i+1} = r_i − α_i A p_i
    if ‖r_{i+1}‖_2 < tol then
      iter=i
      return[x_{i+1}, iter]
    end if
    z_{i+1} = cholsolve(R, r_{i+1})
    β_i = (r_{i+1}^T z_{i+1}) / (r_i^T z_i)
    p_{i+1} = z_{i+1} + β_i p_i
  end for
  iter=-1 return[x_{numiter+1}, iter]
end function
```

For a particularly difficult matrix, it is possible that the incomplete Cholesky decomposition will break down due to cancelation error. One of the more sophisticated techniques used to improve the algorithm is the *drop tolerance*-based incomplete Cholesky decomposition. This method keeps the off-diagonal element r_{ij} computed by the Cholesky algorithm if a condition applies and retains the original value a_{ij} otherwise. For instance, in Ref. [67], the following criterion is suggested:

$$r_{ij} = \begin{cases} \frac{a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}}{r_{ii}} & a_{ij}^2 > \mathrm{tol}^2 a_{ii}\, b_{jj} \\ a_{ij} & \text{otherwise} \end{cases}$$

As the drop tolerance decreases, the incomplete Cholesky factor becomes more dense (Problem 21.25). Even with more advanced techniques, it still can be difficult to find an incomplete Cholesky preconditioner that works. The sophisticated MATLAB function `ichol` computes the incomplete Cholesky decomposition (see the documentation for `ichol`). The function `precg` in the software distribution uses `ichol` with selective drop tolerances.

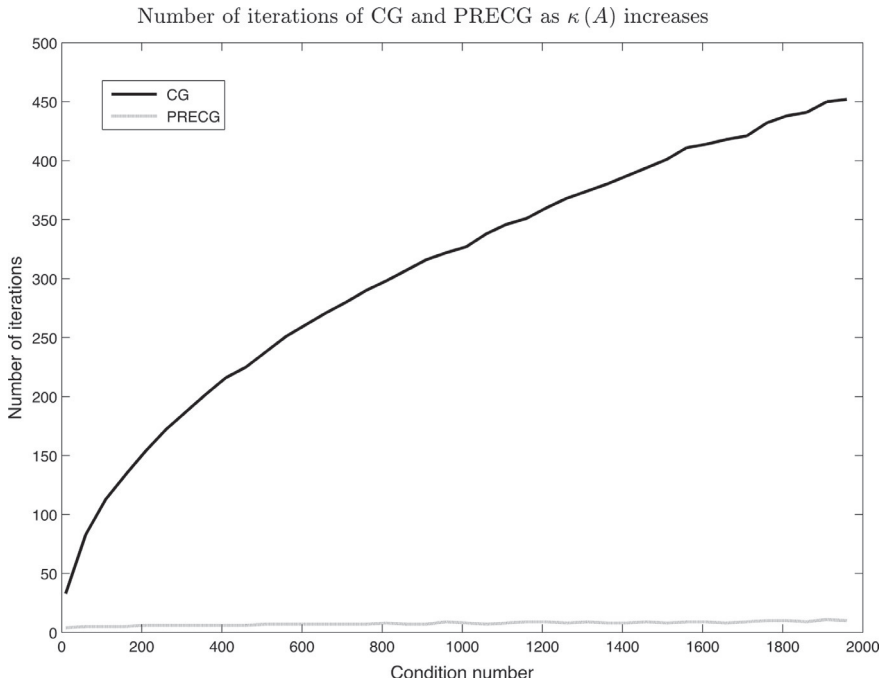


FIGURE 21.7 CG vs. PRECG.

Example 21.6. The MATLAB function call

```
R = sprandsym(n,n,density,rc,1)
```

generates a sparse random symmetric positive definite matrix of size $n \times n$, where `density` is the fraction of nonzeros, and the reciprocal of `rc` is the exact condition number. The number 1 as the last argument instructs the function to make the matrix positive definite. In the software distribution demos directory is a program `cg_vs_precg` that uses `sprandsym` to generate symmetric positive definite matrix of size 800×800 with density 0.05 and condition numbers varying from 10 to 200. The program applies CG and preconditioned CG to each problem and graphs (Figure 21.7) the number of iterations required to attain a minimum error tolerance of 1.0×10^{-6}. ■

21.4.2 SSOR Preconditioner

The SOR method can be used as a basis for building a CG preconditioner that normally one uses if incomplete Cholesky preconditioning is not effective or fails. Assume the diagonal elements of A are nonzero. The formula

$$(D + \omega L)\, x^{(k+1)} = ((1-\omega)\, D - \omega U)\, x^{(k)} + \omega b$$

specifies the SOR iteration in matrix form (see Equation 20.15). Using the equation $U = A - L - D$ along with some matrix algebra, it follows that (Problem 21.12):

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \left(\frac{D}{\omega} + L\right)^{-1} \left(b - Ax^{(k)}\right) \\ &= x^{(k)} + \left(\frac{D}{\omega} + L\right)^{-1} r^{(k)} \end{aligned} \tag{21.20}$$

The matrix $M = \left(\frac{D}{\omega} + L\right)$ can serve as a preconditioner. Assume that $M \approx A$. Then,

$$x^{(k+1)} \approx x^{(k)} + M^{-1}b - x^{(k)} = M^{-1}b.$$

Unfortunately, an SOR preconditioner is not symmetric and cannot be used for a symmetric positive definite matrix. As a result, we will use Equation 21.20 to construct a symmetric positive definite matrix termed the *SSOR preconditioning*

matrix. Starting with the approximation $x^{(k)}$, designate $x^{\left(k+\frac{1}{2}\right)}$ as the result of a forward SOR sweep. Now perform a backward sweep by reversing the iteration direction to obtain $x^{(k+1)}$. The reader can verify that in the backward sweep, the roles of U and L are swapped, so we have the two equations

$$\begin{aligned} x^{\left(k+\frac{1}{2}\right)} &= x^{(k)} + \left(\frac{D}{\omega} + L\right)^{-1} r^{(k)} \\ x^{(k+1)} &= x^{\left(k+\frac{1}{2}\right)} + \left(\frac{D}{\omega} + U\right)^{-1} r^{\left(k+\frac{1}{2}\right)}, \end{aligned} \tag{21.21}$$

where $r^{(k)}$, $r^{(k+1)}$ are residuals. Eliminating $x^{\left(k+\frac{1}{2}\right)}$ from Equation 21.21 gives the SSOR preconditioning matrix. The somewhat involved computations follow:

$$x^{\left(k+\frac{1}{2}\right)} - x^{(k)} = \left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} = \left(\tfrac{D}{\omega} + L\right)^{-1} \left(b - Ax^{(k)}\right)$$

$$r^{\left(k+\frac{1}{2}\right)} = b - Ax^{\left(k+\frac{1}{2}\right)} = b - Ax^{(k)} + Ax^{(k)} - Ax^{\left(k+\frac{1}{2}\right)} = r^{(k)} - A\left(x^{\left(k+\frac{1}{2}\right)} - x^{(k)}\right) =$$

$$\left(L + \tfrac{1}{\omega}D - A\right)\left(x^{\left(k+\frac{1}{2}\right)} - x^{(k)}\right) = \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)}$$

$$x^{(k+1)} - x^{\left(k+\frac{1}{2}\right)} = \left(\tfrac{D}{\omega} + U\right)^{-1} r^{\left(k+\frac{1}{2}\right)} = \left(\tfrac{D}{\omega} + U\right)^{-1} \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)}$$

$$x^{(k+1)} - x^{(k)} = \left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} + \left(\tfrac{D}{\omega} + U\right)^{-1} \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} =$$

$$\left(I + \left(\tfrac{D}{\omega} + U\right)^{-1} \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} =$$

$$\left(\left(\tfrac{D}{\omega} + U\right)^{-1} \left(\tfrac{D}{\omega} + U\right) + \left(\tfrac{D}{\omega} + U\right)^{-1} \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} =$$

$$\left(\tfrac{D}{\omega} + U\right)^{-1} \left(\left(\tfrac{D}{\omega} + U\right) + \left(\left(\tfrac{1}{\omega} - 1\right)D - U\right)\right)\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)} = \left(\tfrac{D}{\omega} + U\right)^{-1} \left(\tfrac{2}{\omega} - 1\right)D\left(\tfrac{D}{\omega} + L\right)^{-1} r^{(k)}$$

Thus,

$$M_{\text{SSOR}}^{-1} = \left(U + \frac{1}{\omega}D\right)^{-1} \left(\frac{2}{\omega} - 1\right) D \left(L + \frac{1}{\omega}D\right)^{-1}$$

and

$$M_{\text{SSOR}} = \frac{\omega}{2-\omega}\left(L + \frac{1}{\omega}D\right) D^{-1} \left(U + \frac{1}{\omega}D\right). \tag{21.22}$$

The reader should verify that M_{SSOR} is symmetric positive definite (Problem 21.9). Using $\omega = 1$ in Equation 21.22 gives

$$M_{\text{PGS}} = (D + L)\, D^{-1}\, (D + U)\,.$$

The matrix M_{PGS} corresponds to the Gauss-Seidel method. The value of ω is not as critical as the choice of ω for the SOR iteration, and $\omega = 1$ can be quite effective in many cases. Solve $M_{\text{PGS}}\, z = r$, or $(D + L)\, D^{-1}\, (D + U)\, z = r$ in two stages.

a. Solve $(D + L)\, y_1 = r$ for y_1.
b. Solve $(D + U)\, z = D y_1$ for z.

Remark 21.4. In MATLAB, compute z as follows:

```
z = (U+D)\(D*((L+D)\r));
```

The function `precg` in the book software distribution implements Algorithm 21.3 and adds the option of using the SSOR preconditioner M_{PGS}. Its calling format is

```
function [x residual iter] =
precg(A,b,x0,tol,maxiter,method,droptol)
```

where method = 'incomplete Cholesky' or 'SSOR' and droptol is the drop tolerance. The drop tolerance is only applicable when method is 'incomplete Cholesky'. If method is omitted, incomplete Cholesky is assumed with zero-fill. If method is specified and droptol is not, droptol defaults to 1.0e−4.

There is an extensive literature concerning preconditioning that includes the CG method as well as other iterations. We will present a preconditioner for the GMRES method in Section 21.7.1. For a general discussion of preconditioning, see Refs. [2, pp. 650-669], [64, pp. 283-351], and the book by Ref. [68].

Example 21.7. The matrix PRECGTEST in the software distribution is a 10000×10000 block pentadiagonal matrix having an approximate condition number of 8.4311×10^6. The following table shows four attempts to solve the system with $b = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, $x_0 = 0$, tol $= 1.0 \times 10^{-6}$, and maxiter = 2000. The incomplete Cholesky decomposition failed with zero-fill and also with a drop tolerance of 1.0×10^{-2} and 1.0×10^{-3}. Using a drop tolerance of 1.0×10^{-4} it succeeded very quickly. The SSOR preconditioned system required 1159 iterations and gave a slightly better result.

Method	droptol	Iterations	Residual	Time
Preconditioned Cholesky	Nofill	–	Fail	–
Preconditioned Cholesky	1.0×10^{-2}	–	Fail	–
Preconditioned Cholesky	1.0×10^{-3}	–	Fail	–
Preconditioned Cholesky	1.0×10^{-4}	59	9.7353×10^{-7}	0.324 s
SSOR	–	1159	9.4987×10^{-7}	2.594 s

■

21.5 KRYLOV SUBSPACES

If the coefficient matrix is large, sparse, and positive definite, the method of choice is normally CG. However, in applications it is often the case that the coefficient matrix is symmetric but not positive definite or may not even be symmetric. There are sophisticated iterations for these problems, most of which are based on the concept of a Krylov subspace. A Krylov subspace-based method does not access the elements of the matrix directly, but rather performs matrix-vector multiplication to obtain vectors that are projections into a lower-dimensional Krylov subspace, where a corresponding problem is solved. The solution is then converted into a solution of the original problem. These methods can give a good result after a relatively small number of iterations.

Definition 21.2. Assume $A \in \mathbb{R}^{n \times n}$, $u \in \mathbb{R}^n$, and k is an integer, the *Krylov sequence* is the set of vectors

$$u, Au, A^2u, \dots, A^{k-1}u$$

The *Krylov subspace* $\mathcal{K}_n(A, u)$ generated by A and u is

$$\text{span}\left\{ u, Au, A^2u, \dots, A^{k-1}u \right\}.$$

It is of dimension k if the vectors are linearly independent.

Although we approached the CG method using an optimization approach, CG is also a Krylov subspace method. Theorem 21.3 shows the connection.

Theorem 21.3. *Assume A is nonsingular. With an initial guess $x_0 = 0$, after i iterations of the CG method,*

$$\begin{aligned} \text{span}\{x_1, x_2, \dots, x_i\} &= \text{span}\{p_0, p_1, \dots, p_i\} = \text{span}\{r_0, r_2, \dots, r_i\} \\ &= \mathcal{K}_i(A, p_0) = \text{span}\left\{p_0, Ap_0, \dots, A^{i-1}p_0\right\} = \text{span}\left\{r_0, Ar_0, \dots, A^{i-1}r_0\right\} \end{aligned}$$

Proof. Let

$$S_i = \text{span}\{p_0, p_1, \dots, p_{i-1}\}.$$

Now, $x_i = x_{i-1} + \alpha_{i-1}p_{i-1}$, and so $x_1 = 0 + \alpha_0 p_0$, $x_2 = x_1 + \alpha_1 p_1 = \alpha_0 p_0 + \alpha_1 p_1$. In general, $x_i = \sum_{k=0}^{i-1} \alpha_k p_k$, and

$$\text{span}\{x_1, x_2, \dots, x_i\} = S_i.$$

We have $r_0 = p_0$ and from Equation 21.13 $r_i = p_i - \beta_i p_{i-1}$, so

$$\text{span}\{r_0, r_1, \dots, r_{i-1}\} = S_i.$$

From property 1 of Lemma 21.1

$$r_i = r_{i-1} - \alpha_{i-1} A p_{i-1}.$$

Since r_i and r_{i-1} are in S_i,

$$Ap_{i-1} \in S_i.$$

As a result,

$$S_i = \mathcal{K}_i(A, p_0) = \text{span}\left\{p_0, Ap_0, \ldots, A^{i-1}p_0\right\} = \mathcal{K}_i(A, r_0) = \text{span}\left\{r_0, Ar_0, \ldots, A^{i-1}r_0\right\}.$$

The p_i are A-orthogonal, the r_i are orthogonal, and so both sequences are linearly independent. Since A is nonsingular, the vectors in $\mathcal{K}_i(A, p_0)$ and $\mathcal{K}_i(A, r_0)$ are linearly independent. □

There are a number of highly successful iterative methods based on Krylov subspaces that work with the full range of matrix types. We will discuss the GMRES algorithm that applies to general matrices. We will also develop the MINRES method for symmetric nonpositive definite problems, since it can be effective and its derivation is very similar to the GMRES method. Other methods include the biconjugate gradient (Bi-CG) method, and the quasi-minimal residual (QMR) method for nonsymmetric matrices. The reader will find a presentation of these Krylov subspace methods in Refs. [2, pp. 639-647], [26, pp. 303-312], and [69, Chapters 7 and 9].

21.6 THE ARNOLDI METHOD

Our aim is to develop a method of solving a large, sparse system $Ax = b$, where $A^{n\times n}$ is a general nonsymmetric matrix. We approximate the solution by projecting the problem into a Krylov subspace $\mathcal{K}_m(A, r_0) = \left\{r_0, Ar_0, \ldots, A^{m-1}r_0\right\}$, where m is much smaller than n ($m \ll n$). Obtaining the solution to a related problem will be practical in this much smaller subspace $\mathbb{R}^m \subset \mathbb{R}^n$. We then convert the solution to the problem in $\mathbb{R}^m$ to the solution we want in $\mathbb{R}^n$ (Figure 21.8).

Our approach is to develop the *Arnoldi decomposition* of A and use it to solve a problem in $\mathbb{R}^m$ that, in turn, will be used to approximate the solution x to $Ax = b$ in $\mathbb{R}^n$. The decomposition creates an $n \times (m + 1)$ matrix Q_{m+1} with orthonormal columns and an $(m + 1) \times m$ upper Hessenberg matrix $\overline{H}_m$ such that

$$AQ_m = Q_{m+1}\overline{H}_m,$$

where $Q_m = Q_{m+1}(:, 1 : m)$. We build the decomposition by using a Krylov subspace

$$K_{m+1}(A, x_1) = \left\{x_1, Ax_1, \ldots, A^m x_1\right\},$$

where x_1 is an initial vector. Generally, the set of vectors $\left\{x_1, Ax_1, \ldots, A^{i-1}x_1\right\}$ is not a well-conditioned basis for $\mathcal{K}_i(A, x_1)$ since, as we showed in Chapter 18 when presenting the power method, the sequence approaches the dominant eigenvector of A. Thus the last few vectors in the sequence may be very close to pointing in the same direction. In order to fix this problem, we compute an orthonormal basis for $\mathcal{K}_i(A, x_1)$ using the modified Gram-Schmidt process. After i steps of the Gram-Schmidt process we have vectors $q_1, q_2, \ldots, q_i$ and do not retain the original set $\left\{x_1, Ax_1, \ldots, A^{i-1}x_1\right\}$. If we had $A^i x_1$, we could apply Gram-Schmidt to extend the orthonormal sequence to $q_1, q_2, \ldots, q_i, q_{i+1}$. We must find an alternative for the determination of q_{i+1}. We do not have A^i, but we do have A, so we compute Aq_i and apply Gram-Schmidt to extend the orthonormal sequence by 1. It can be shown [23, pp. 439-441] that even though we used Aq_i as the next vector rather

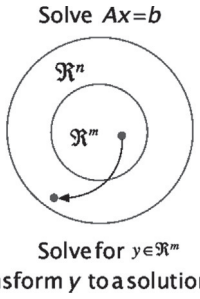


FIGURE 21.8 Arnoldi projection from $\mathbb{R}^n$ into $\mathbb{R}^m$, $m \ll n$.

than $A^i x_1$,

$$\text{span}\left\{x_1, Ax_1, \ldots, A^i x_1\right\} = \text{span}\left\{q_1, q_2, \ldots, q_{i+1}\right\}.$$

The first step is to normalize the initial vector x_1 by computing

$$q_1 = \frac{x_1}{\|x_1\|_2}.$$

For $1 \leq i \leq m$,

$$\overline{q_{i+1}} = Aq_i - \sum_{j=1}^{i} q_j h_{ji},$$

where h_{ji} is the Gram-Schmidt coefficient

$$h_{ji} = \langle q_j, Aq_i \rangle = q_j^T A q_i,\ j \leq i$$

Assign $h_{i+1,i} = \|\overline{q_{i+1}}\|_2$, and normalize $\overline{q_{i+1}}$ to obtain

$$q_{i+1} = \frac{\overline{q_{i+1}}}{h_{i+1,i}}.$$

The Gram-Schmidt coefficients are computed as follows:

$$\begin{aligned}
&h_{11},\ h_{21}\\
&h_{12},\ h_{22},\ h_{32}\\
&h_{13},\ h_{23},\ h_{33},\ h_{43}\\
&\vdots\\
&h_{1m},\ h_{2m},\ \ldots h_{m+1,m}
\end{aligned}$$

and form the $(m+1) \times m$ upper Hessenberg matrix

$$\overline{H_m} = \begin{bmatrix}
h_{11} & h_{12} & \ldots & \ldots & h_{1,m}\\
h_{21} & h_{22} & \ldots & \ldots & h_{2,m}\\
0 & h_{32} & \ddots & \ldots & \vdots\\
\vdots & \vdots & \ddots & h_{m-1,m-1} & \vdots\\
\vdots & \vdots & \ddots & h_{m,m-1} & h_{m,m}\\
0 & 0 & \ldots & 0 & h_{m+1,m}
\end{bmatrix}.$$

From the two equations

$$\begin{aligned}
\overline{q_{i+1}} &= Aq_i - \sum_{j=1}^{i} q_j h_{ji}\\
q_{i+1} &= \frac{\overline{q_{i+1}}}{h_{i+1,i}},
\end{aligned}$$

we have

$$Aq_i = \sum_{j=1}^{i} q_j h_{ji} + \overline{q_{i+1}} = \sum_{j=1}^{i} q_j h_{ji} + h_{i+1,i} q_{i+1} = \sum_{j=1}^{i+1} q_j h_{ji}. \tag{21.23}$$

Form the $n \times m$ matrix $Q_m = \begin{bmatrix} q_1 & q_2 & \ldots & q_m \end{bmatrix}$ and the $n \times (m+1)$ matrix $Q_{m+1} = \begin{bmatrix} q_1 & q_2 & \ldots & q_m & q_{m+1} \end{bmatrix}$ from the column vectors $\{q_i\}$. Matrix equation 21.24 follows directly from Equation 21.23:

$$AQ_m = Q_{m+1}\overline{H_m} \tag{21.24}$$

Figure 21.9 depicts the decomposition. Algorithm 21.4 specifies the Arnoldi process that generates an orthonormal basis $\{q_1, q_2, \ldots, q_{m+1}\}$ for the subspace spanned by $K_{m+1}(A, x_1)$ and builds the matrix $\overline{H_m}$.

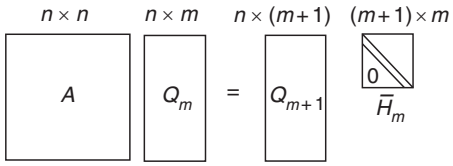


FIGURE 21.9 Arnoldi decomposition form 1.

Algorithm 21.4 Arnoldi Process

```
function ARNOLDI(A, x_1, m)
  % Input: n × n matrix A, n × 1 column vector x_1, and integer m
  % Output: m+1 orthogonal vectors q_1, q_2, ... , q_{m+1}
  % and an (m + 1) × m matrix H̄
  q_1 = x_1 / ‖x_1‖_2
  for k=1:m do
    w = A q_k
    for j=1:k do
      h_jk = q_j^T w
      w = w − h_jk q_j
    end for
    h_{k+1,k} = ‖w‖_2
    if h_{k+1,k} = 0 then
      return [Q = [q_i], H̄ = [h_ij]]
    end if
    q_{k+1} = w / h_{k+1,k}
  end for
  return [Q = [q_i], H̄ = [h_ij]]
end function
```

NLALIB: The function `arnoldi` implements Algorithm 21.4.

Efficiency

Since the Arnoldi method uses Gram-Schmidt, the approximate number of flops for the algorithm is $2m^2n$.

Notice that when $h_{k+1,k} = 0$, the Arnoldi algorithm stops. Clearly we cannot compute q_{k+1} due to division by zero, but there is a more to it than that, as indicated in Definition 21.3.

Definition 21.3. Let T be a linear transformation mapping a vector space V to V. A subspace $W \subseteq V$ is said to be an invariant subspace of T if for every $x \in W$, $Tx \in W$.

If $h_{i+1,i} = \|\overline{q_{i+1}}\|_2 = 0$, then the subspace span $\{q_1, q_2, \ldots, q_i\}$ is an invariant subspace of A, and further iterations produce nothing, and so the algorithm terminates. This means that a method for solving $Ax = b$ based on projecting onto the subspace $\mathcal{K}(A, j)$ will be exact. This is what is termed a "lucky breakdown" [64, pp. 154-156].

21.6.1 An Alternative Formulation of the Arnoldi Decomposition

Equation 21.24 can be written as (Problem 21.6)

$$AQ_m - Q_m H_m = h_{m+1,m} \begin{bmatrix} 0 & 0 & \ldots & 0 & q_{1m} \\ 0 & 0 & \ldots & 0 & q_{2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & q_{nm} \end{bmatrix},$$

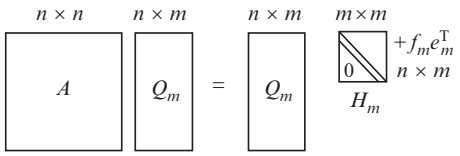


FIGURE 21.10 Arnoldi decomposition form 2.

or

$$AQ_m = Q_m H_m + h_{m+1,m} q_{m+1} e_m^{\mathrm{T}}, \tag{21.25}$$

where H_m is $\overline{H}_m$ with the last row removed, $f_m = h_{m+1,m} q_{m+1}$, and $e_m^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}$. Figure 21.10 depicts the decomposition. This form of the Arnoldi decomposition will be very useful in Chapter 22 when we discuss the computation of eigenvalues of sparse matrices.

21.7 GMRES

Assume A is a real $n \times n$ matrix, b is an $n \times 1$ vector, and we want to solve the system $Ax = b$. Assume that x_0 is an initial guess for the solution, and $r_0 = b - Ax_0$ is the corresponding residual. The *GMRES method* looks for a solution of the form $x_m = x_0 + Q_m y_m$, $y_m \in \mathbb{R}^m$ where the columns of Q_m are an n-dimensional orthogonal basis for the Krylov subspace $\mathcal{K}_m(A, r_0) = \left\{r_0, Ar_0, \dots, A^{m-1} r_0\right\}$. The vector y_m is chosen so the residual

$$\|r_m\|_2 = \|b - A(x_0 + Q_m y_m)\|_2 = \|r_0 - AQ_m y_m\|_2$$

has minimal norm over $\mathcal{K}_m(A, r_0)$. This is a least-squares problem. We must find a vector y_m that minimizes the residual specified in Equation 21.26,

$$\|r_m\|_2 = \|r_0 - AQ_m y_m\|_2 . \tag{21.26}$$

Let $\beta = \|r_0\|_2$. The first Arnoldi vector is $q_1 = \frac{r_0}{\|r_0\|}$, so $r_0 = \beta q_1$. By using Equations 21.24–21.26

$$\|r_m\|_2 = \left\|\beta q_1 - Q_{m+1} \overline{H_m} y_m\right\|_2 . \tag{21.27}$$

The Arnoldi vector q_1 is the first column of Q_{m+1}, so $q_1 = Q_{m+1} \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^{\mathrm{T}} = Q_{m+1} e_1$, and from Equation 21.27

$$\|r_m\|_2 = \left\|Q_{m+1}\left(\beta e_1 - \overline{H_m} y_m\right)\right\|_2 .$$

Since the columns of Q_{m+1} are orthonormal, we must minimize

$$\left\|\beta e_1 - \overline{H_m} y_m\right\|_2 .$$

This means that after solving the $(m+1) \times m$ least-squares problem

$$\overline{H_m} y_m = \beta e_1, \tag{21.28}$$

the approximate solution to $Ax = b$ is

$$x_m = x_0 + Q_m y_m .$$

Use the QR decomposition approach to solving overdetermined least-squares problems (Section 16.2.2). Since $\overline{H_m}$ in Equation 21.28 is an upper Hessenberg matrix, the system can be solved in $O\left(m^2\right)$ flops. In the practical implementation of GMRES, one estimate x_m is often not sufficient to obtain the error tolerance desired. Use x_m as an improved initial vector and repeat the process until satisfying the error tolerance. Algorithm 21.5 presents the GMRES method.

Algorithm 21.5 GMRES

```
function GMRESB(A,b,x_0,m,tol,maxiter)
   % Solve Ax = b using the GMRES method
   % Input: n × n matrix A, n × 1 vector b,
   % initial approximation x_0, integer m < n,
   % error tolerance tol, and the maximum number of iterations, maxiter.
   % Output: Approximate solution x_m, associated residual r,
   % and iter, the number of iterations required.
   % iter = −1 if the tolerance was not satisfied.

   iter = 1
   while iter ≤ maxiter do
      r = b − Ax_0
      [ Q_{m+1}  H̄_m ] = arnoldi(A, r, m)
      β = ‖r‖_2
      Solve the (m + 1) × m least-squares problem H̄_m y_m = β e_1
      using Givens rotations that take advantage of the upper
      Hessenberg structure of H̄_m
      x_m = x_0 + Q_m y_m
      r = ‖b − Ax_m‖_2
      if r < tol then
         return [x_m,r,iter]
      end if
      x_0 = x_m
      iter = iter + 1
   end while
   iter = −1
   return [x_m, r, iter]
end function
```

NLALIB: The function `gmresb` implements Algorithm 21.5. The function `hesslqsolve` in the software distribution implements the *QR* decomposition of an upper Hessenberg matrix. The implementation simply calls `givenshess` to perform the *QR* decomposition rather than `qr`. The name contains the trailing "*b*" because MATLAB supplies the function `gmres` that implements the GMRES method.

The choice of m is experimental. Try a small value and see if the desired residual norm can be attained. If not, increase m until obtaining convergence or finding that GMRES simply does not work. Of course, as you increase m, memory and computational effort increase. It may happen that the problem is not tractable using GMRES. In that case, there are other methods such as Bi-CG and QMR that may work [64, pp. 217-244].

Example 21.8. A *Toeplitz matrix* is a matrix in which each diagonal from left to right is constant. For instance,

$$\begin{bmatrix} 1 & 2 & 8 & -1 \\ 3 & 1 & 2 & 8 \\ 8 & 3 & 1 & 2 \\ 7 & 8 & 3 & 1 \end{bmatrix}$$

is a Toeplitz matrix. The following MATLAB statements create a 1000×1000 pentadiagonal sparse Toeplitz matrix with a small condition number. As you can see, `gmresb` works quite well with $m = 50$, tol $= 1.0 \times 10^{-14}$ and a maximum of 25 iterations. The method actually required 11 iterations and obtained a residual of 9.6893×10^{-15}.

```
>> P = gallery('toeppen',1000);
>> condest(P)

ans =
   23.0495

>> b = rand(1000,1);
```

```
>> x0 = ones(1000,1);
>> [x r iter] = gmresb(P,b,x0,50,1.0e-14,25);
>> r
r =
  9.6893e-015
>> iter
iter =
    11
```

■

Convergence

If A is positive definite, then GMRES converges for any $m \geq 1$ [64, p. 205]. There is no other simple result for convergence. A theorem that specifies some conditions under which convergence will occur can be found in Ref. [64, p. 206].

21.7.1 Preconditioned GMRES

In the same way that we used incomplete Cholesky decomposition to precondition A when A is positive definite, we can use the *incomplete LU decomposition* to precondition a general matrix. Compute factors L and U so that if element $a_{ij} \neq 0$ then the element at index (i, j) of $A - LU$ is zero. To do this, compute the entries of L and U at location (i, j) only if $a_{ij} \neq 0$. It is hoped that if $M = LU$, then $M^{-1}A$ will have a smaller condition number than A. Algorithm 21.6 describes the incomplete LU decomposition. Rather than using vectorization, it is convenient for the algorithm to use a triply nested loop. For more details see Ref. [64, pp. 287-296].

Algorithm 21.6 Incomplete *LU* Decomposition

```
function ILUB(A)
  % Compute an incomplete LU decomposition
  % Input: n × n matrix A
  % Output: lower triangular matrix L and upper triangular matrix U.
  for i=2:n do
    for j=1:i-1 do
      if a_ij ≠ 0 then
        a_ij = a_ij/a_jj
        for k = j+1:n do
          if a_ik ≠ 0 then
            a_ik = a_ik − a_ij a_jk
          end if
        end for
      end if
    end for
  end for
  U=upper triangular portion of A
  L=portion of A below the main diagonal

  for i=1:n do
    l_ii = 1
  end for
end function
```

NLALIB: The function `ilub` implements Algorithm 21.6.

Proceeding as we did with incomplete Cholesky, there results

$$(LU)^{-1} Ax = (LU)^{-1} b$$
$$\left(L^{-1}AU^{-1}\right)(Ux) = L^{-1}b$$

and

$$\overline{A}\overline{x} = \overline{b},$$

where

$$\begin{aligned} \overline{A} &= L^{-1}AU^{-1} \\ \overline{b} &= L^{-1}b \\ \overline{x} &= Ux \end{aligned} \tag{21.29}$$

The function `pregmres` in the software distribution approximates the solution to $Ax = b$ using Equation 21.29.

Remark 21.5. Algorithm 21.6 will fail if there is a zero on the diagonal of U. In this case, it is necessary to use Gaussian elimination with partial pivoting. We will not discuss this, but the interested reader will find a presentation in Ref. [64, pp. 287-320]. The software distribution contains a function `mpregmres` that computes the incomplete LU decomposition with partial pivoting by using the MATLAB function `ilu`. It returns a decomposition such that $P\overline{A} = LU$, so $\overline{A} = P^{\mathrm{T}}LU$. It is recommended that, in practice, `mpregmres` be used rather than pregmres.

Example 21.9. The 903×903 nonsymmetric matrix, `DK01R`, in Figure 21.11 was used to solve a computational fluid dynamics problem. `DK01R` was obtained from the University of Florida Sparse Matrix Collection. A right-hand side, `b_DK01R`, and an approximate solution, `x_DK01R`, were supplied with the matrix. The approximate condition number of the matrix is 2.78×10^8, so it is ill-conditioned. Using $x_0 = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^{\mathrm{T}}$, $m = 300$, and tol $= 1.0 \times 10^{-15}$, niter $= 20$, the solution was obtained using `gmresb` and `mpregmres`. Table 21.1 gives the results of comparing the solutions from `mpregmres` and `gmresb` to `x_DK01R`.

TABLE 21.1 Comparing gmresb and mpregmres

	iter	r	Time	$\|x\_DK01R - x\|_2$
Solution supplied	–	6.29×10^{-16}	–	–
gmresb	−1(failure)	5.39×10^{-10}	6.63	9.93×10^{-11}
mpregmres	1	1.04×10^{-15}	0.91	5.20×10^{-17}

In a second experiment, the function `gmresb` required 13.56 s and 41 iterations to attain a residual of 8.10×10^{-16}. Clearly, preconditioning GMRES is superior to normal GMRES for this problem. ■

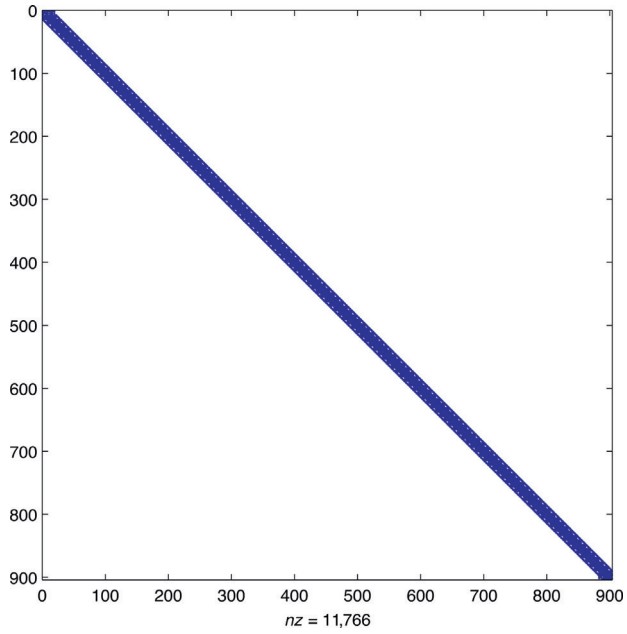


FIGURE 21.11 Large nonsymmetric matrix.

21.8 THE SYMMETRIC LANCZOS METHOD

The *Lanczos method* is the Arnoldi method applied to a symmetric matrix. For reasons that will become evident, replace the matrix name H by T. Like the Arnoldi decomposition, the Lanczos decomposition can be written in two ways

$$\begin{aligned} AT_m &= Q_{m+1}\overline{T_m} \\ AQ_m &= Q_m T_m + t_{m+1,m} q_{m+1} e_m^{\mathrm{T}} \end{aligned}$$

Figure 21.12 depicts the two formulations. Now, noting that the columns of Q_m are orthogonal to q_{m+1},

$$Q_m^{\mathrm{T}} A Q_m = Q_m^{\mathrm{T}} Q_m T_m + t_{m+1,m} \left(Q_m^{\mathrm{T}} q_{m+1} e_m^{\mathrm{T}} \right) = T_m + 0 = T_m,$$

and

$$T_m^{\mathrm{T}} = \left(Q_m^{\mathrm{T}} A Q_m \right)^{\mathrm{T}} = Q_m^{\mathrm{T}} A^{\mathrm{T}} Q_m = Q_m^{\mathrm{T}} A Q_m = T_m.$$

Thus, T_m is symmetric, and a symmetric upper Hessenberg matrix must be tridiagonal, so T_m and $\overline{T_m}$ have the form

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \alpha_{m-1} & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_m \end{bmatrix}, \quad \overline{T_m} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \alpha_{m-1} & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_m \\ 0 & 0 & \dots & 0 & t_{m+1,} \end{bmatrix}$$

The interior loop of the Arnoldi iteration uses the array entries h_{jm}, $1 \le j \le m$. From the tridiagonal structure of T_m, the only entries in column k from rows 1 through k are β_{k-1} in row $k-1$ and α_k in row k. Trace the inner Arnoldi loop for $j = k-1$ and $j = k$. You will determine that the Arnoldi inner loop can be replaced by two statements:

$$\begin{aligned} \alpha_k &= q_k^{\mathrm{T}} A q_k \\ w &= w - \beta_{k-1} q_{k-1} - \alpha_k q_k, \end{aligned}$$

so the computation of $\overline{T_m}$ is much faster than for the Arnoldi process. Algorithm 21.7 defines the Lanczos process.

NLALIB: The function `lanczos` implements Algorithm 21.7.

21.8.1 Loss of Orthogonality with the Lanczos Process

Using the Lanczos process is not as simple as it may seem. Poor convergence can result from a bad choice of the starting vector, so a random vector is a good choice. Using exact arithmetic, the vectors $q_1, q_2, \dots q_m, q_{m+1}$ are mutually

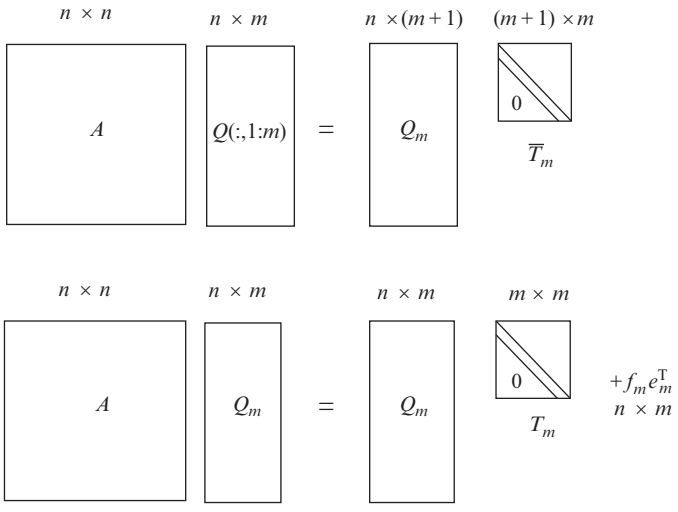


FIGURE 21.12 Lanczos decomposition.

orthogonal. When implemented in finite precision arithmetic, the Lanczos algorithm does not behave as expected. Roundoff error can cause lack or orthogonality among the Lanczos vectors with serious consequences. We will outline why this happens in Chapter 22 when we discuss computing a few eigenvalues of a large sparse symmetric matrix. One approach to fix this problem is to reorthogonalize as each new Lanczos vector is computed; in other words, we orthogonalize twice.

Algorithm 21.7 Lanczos Method

```
function LANCZOS(A, x0, m)
   q0 = 0
   β0 = 0
   q1 = x0/‖x0‖2

   for k=1:m do
      w = Aqk
      αk = qkTw
      w = w − βk−1qk−1 − αkqk
      βk = ‖w‖2
      qk+1 = w/βk
   end for
   T(1:m,1:m)=diag(β(1:m-1),-1)+diag(α)+diag(β(1:m-1),1);
   T(m+1,m)=β(m);
   return[ Q, T ]
end function
```

This is termed *full reorthogonalization* and is time-consuming but, as we will see in Example 21.10, can be of great benefit. Do the reorthogonalization by following

$$\begin{aligned} \alpha_k &= q_k^{\mathrm{T}} A q_k \\ w &= w - \beta_{k-1} q_{k-1} - \alpha_k q_k \end{aligned}$$

with

```
for i=1:k-1 do
   h = qiTw
   w = w − qih
end for
```

In the software distribution, the function lanczos will perform full reorthogonalization by adding a fourth parameter of 1. If speed is critical, try the algorithm without full reorthogonalization and add it if the results are not satisfactory. It is possible to perform selective reorthogonalization. For a discussion of these issues, see Refs. [1, pp. 366-383] and [2, pp. 565-566].

Example 21.10. Ref. [70, pp. 498-499], a numerical example using a matrix introduced by Strakoš [71] demonstrates lack of orthogonality of the Lanczos vectors. Recall that the eigenvalues of a diagonal matrix are its diagonal elements. Define a diagonal matrix by the eigenvalues

$$\lambda_i = \lambda_1 + \left(\frac{i-1}{n-1}\right)(\lambda_n - \lambda_1)\rho^{n-i}, 1 \le i \le n.$$

The parameter ρ controls the distribution of the eigenvalues within the interval $[\ \lambda_1\ \ \lambda_n\ ]$. Create the matrix with $n = 30$, $\lambda_1 = 0.1$, $\lambda_n = 100$, and $\rho = 0.8$, that has well-separated large eigenvalues. Execute $m = n$ steps of the Lanczos process and compute $q_i^{\mathrm{T}} q_j$, $1 \le i, j \le n$. Using exact arithmetic, the Lanczos vectors comprising Q should form an orthonormal set, so $\left\|q_i^{\mathrm{T}} q_j\right\|_2 = 0$, $i \ne j$ and $\left\|q_i^{\mathrm{T}} q_i\right\|_2 = 1$. Figure 21.13(a) is a plot of $\left\|q_i^{\mathrm{T}} q_j\right\|_2$ over the grid $1 \le i, j \le n$ without reorthogonalization, and Figure 21.13(b) plots the same data with reorthogonalization. Note the large difference between the two plots. Clearly, Figure 21.3(a) shows that the Lanczos vectors lost orthogonality. As further evidence, if we name the variable Q in the first plot $Q1$ and name Q in the second plot $Q2$, we have

```
norm(Q1)=1.41843
norm(Q2)=1.00000
```

Problem 21.36 asks you to reproduce these results. ■

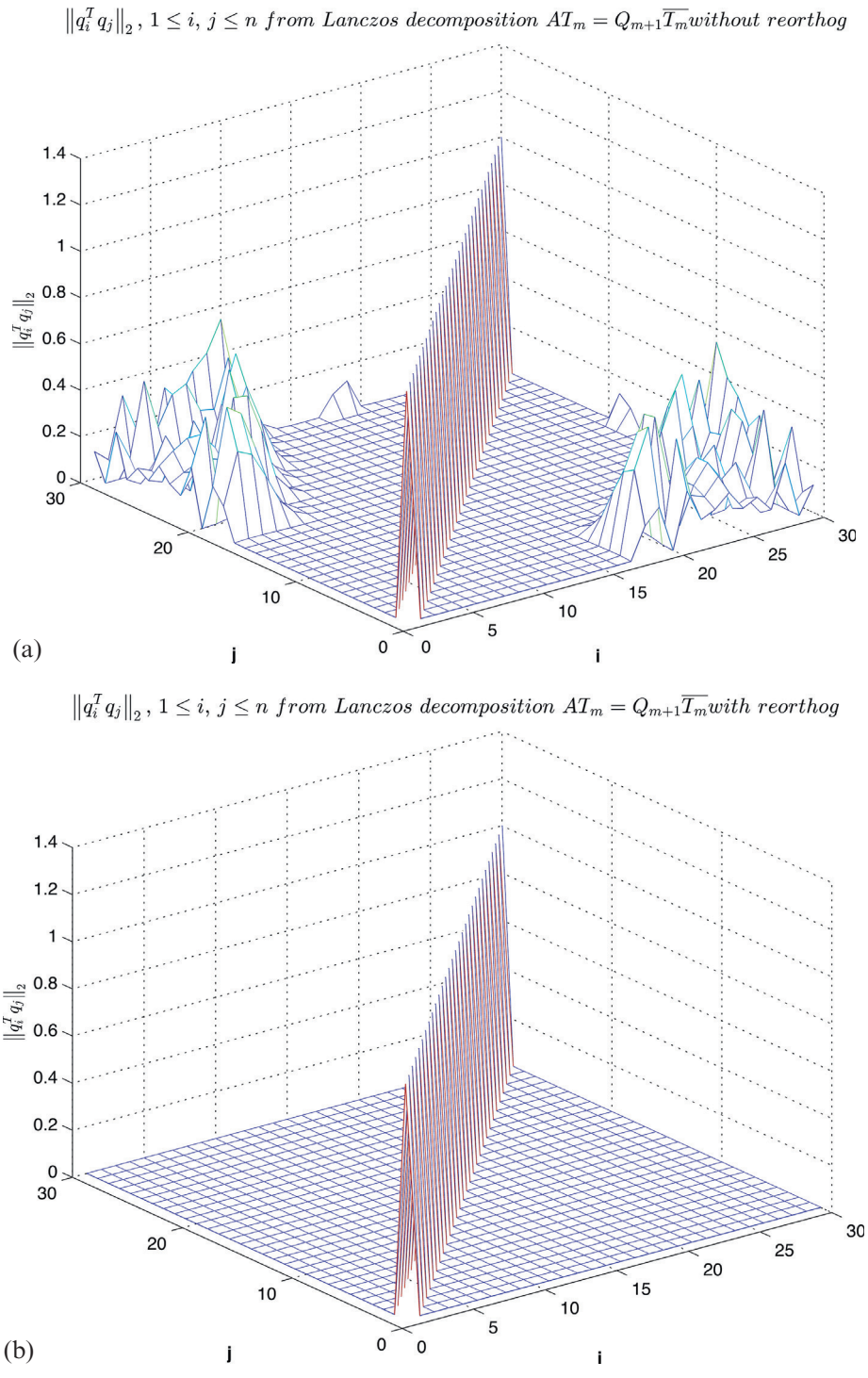


FIGURE 21.13 Lanczos process with and without reorthogonalization. (a) Lanczos without reorthogonalization and (b) Lanczos with reorthogonalization.

21.9 THE MINRES METHOD

If a symmetric matrix is indefinite, the CG method does not apply. The minimum residual method (MINRES) is designed to apply in this case. In the same fashion as we developed the GMRES algorithm using the Arnoldi iteration, Algorithm 21.8 implements the MINRES method using the Lanczos iteration. In the resulting least-squares problem, the coefficient matrix is tridiagonal, and we compute the QR decomposition using Givens rotations.

Algorithm 21.8 MINRES

```
function MINRESB(A,b,x0,m,tol,maxiter)
   % Solve Ax = b using the MINRES method
   % Input: n × n matrix A, n × 1 vector b,
   % initial approximation x0, integer m < n,
   % error tolerance tol, and the maximum number of iterations, maxiter.
   % Output: Approximate solution xm, associated residual r,
   % and iter, the number of iterations required.
   % iter = −1 if the tolerance was not satisfied.

   iter = 1
   while iter ≤ maxiter do
      r = b − Ax0
      [ Qm+1  T̄m ] = lanczos(A, r, m)
      β = ‖r‖2
      Solve the (m + 1) × m least-squares problem T̄m ym = β e1
      using Givens rotations that take advantage of the tridiagonal
      structure of T̄m
      xm = x0 + Qm ym
      r = ‖b − Axm‖2
      if r < tol then
         return [xm,r,iter]
      end if
      x0 = xm
      iter = iter + 1
   end while
   iter = −1
   return [xm, r, iter]
end function
```

NLALIB: The function `minresb` implements Algorithm 21.8. The Lanczos process uses full reorthogonalization.

MINRES does well when a symmetric matrix is well conditioned. The tridiagonal structure of T_k makes MINRES vulnerable to rounding errors [69, pp. 84-86], [72]. It has been shown that the rounding errors propagate to the approximate solution as the square of $\kappa(A)$. For GMRES, the errors propagate as a function of the $\kappa(A)$. Thus, if A is badly conditioned, try `mpregmres`.

Example 21.11. The MINRES method was applied to three systems whose matrices are shown in Figure 21.14. In each case, $x_0 = 0$, and b was a matrix with random integer values. Matrix (a) has a small condition number. Using $m = 50$ and tol $= 1.0 \times 10^{-6}$, one iteration gave a residual of 3.5×10^{-10}. Matrix (b) has a condition number of approximately 772, but with the same parameters, MINRES yielded a residual of 2.5×10^{-8} in three iterations. Matrix (c) is another story. It has an approximate condition number of 2.3×10^4 and so is ill-conditioned. Using the parameters $m = 1000$ and tol $= 1.0 \times 10^{-6}$, MINRES gave a residual of 8.79×10^{-7} using 42 iterations. On the author's system, this required approximately 4 min, 13 s of computation. In this situation, it is appropriate to try preconditioned GMRES. Using $m = 50$ and tol $= 1.0 \times 10^{-6}$, `mpregmres` produced a residual of 2.75×10^{-12} in one iteration requiring approximately 1.8 s of computation. ■

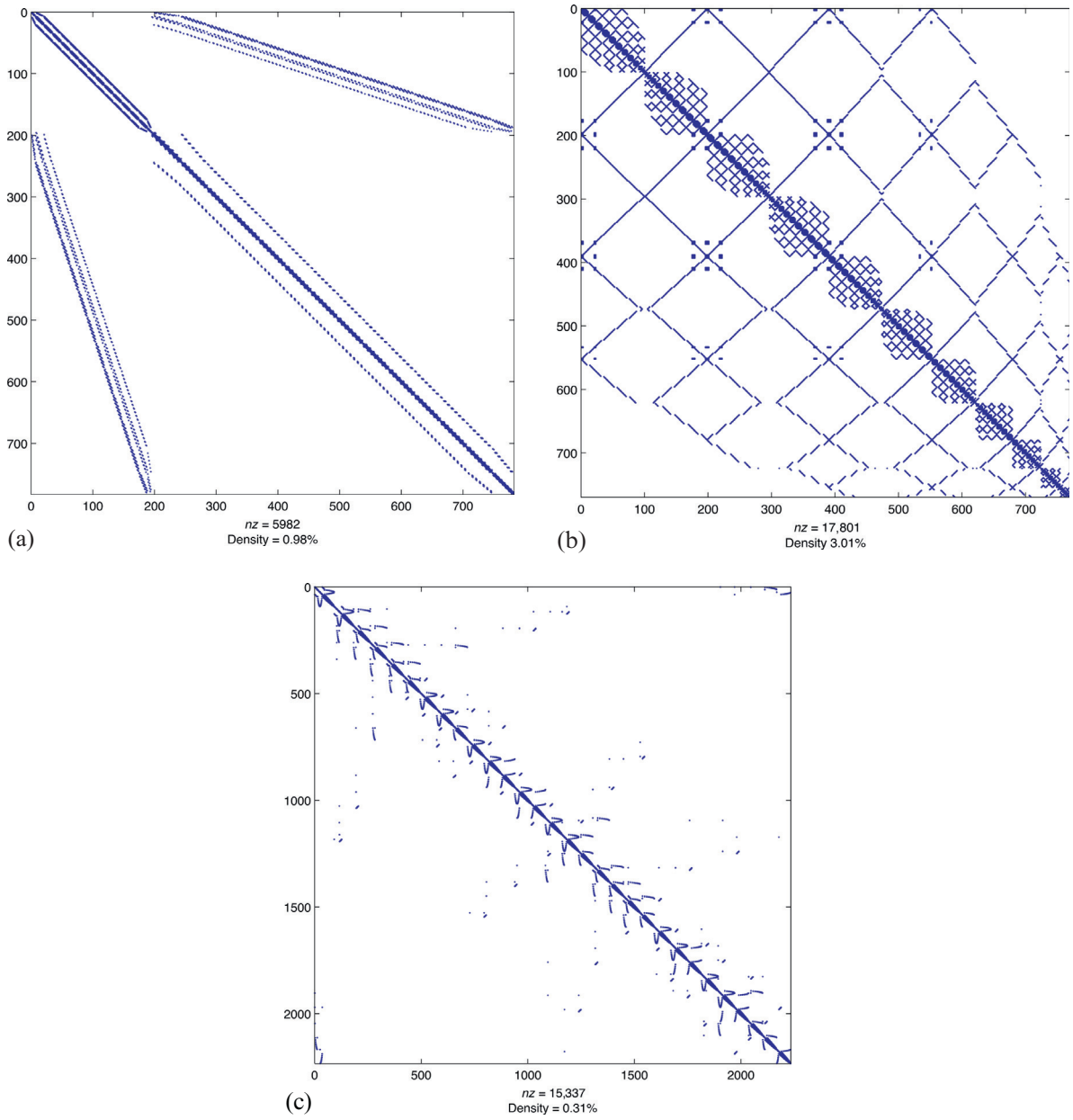


FIGURE 21.14 Large sparse symmetric matrices.

Convergence

Like GMRES, there is no simple set of properties that guarantee convergence. A theorem that specifies some conditions under which convergence will occur can be found in Ref. [73, pp. 50-51].

Remark 21.6. If A is positive definite, one normally uses CG or preconditioned CG. If A is symmetric indefinite and ill-conditioned, it is not safe to use a symmetric preconditioner K with MINRES if $K^{-1}A$ is not symmetric. Finding a preconditioner for a symmetric indefinite matrix is difficult, and in this case the use of GMRES is recommended.

21.10 COMPARISON OF ITERATIVE METHODS

We have developed three iterative methods for large, sparse matrices, CG, GMRES, and MINRES. In the case of CG and GMRES, algorithms using preconditioning were presented. Given the tools we have developed, the following decision tree suggests (Figure 21.15) an approach for choosing an iterative algorithm. Note that sometimes sparse matrix problems will

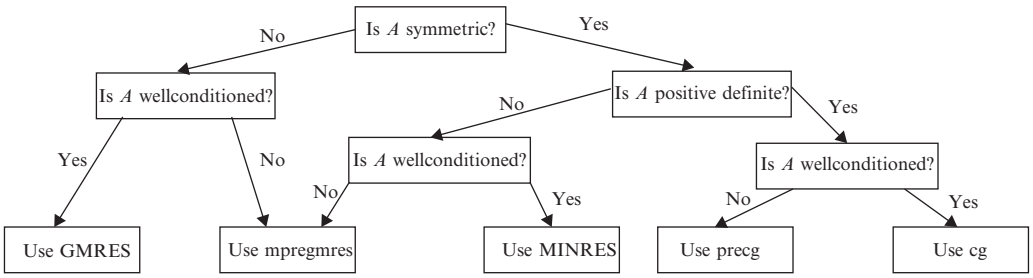


FIGURE 21.15 Iterative method decision tree.

defy proper solution using these techniques, and either other methods should be tried or the user must develop a custom preconditioner.

21.11 POISSON'S EQUATION REVISITED

In Section 20.5, we discussed the numerical solution of the two-dimensional Poisson equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y)$$
$$u(x, y) = g(x, y) \quad \text{on } \partial R, \quad \text{where } R = [0, 1] \times [0, 1]$$

using finite difference equations

$$-u_{i-1,j} - u_{i+1,j} + 4u_{ij} - u_{i,j-1} - u_{i,j+1} = h^2 f(x_i, y_j), \quad 1 \le i, j \le n-1. \tag{21.30}$$

The resulting system was solved using the SOR iteration. In this section, we will solve the system of equations using preconditioned CG. There are $(n-1)^2$ unknown values inside the grid, so we must solve a system $Au = b$ of dimension $(n-1)^2 \times (n-1)^2$. We need to determine the structure of A and b. The matrix A is sparse, since each unknown is connected only to its four closest neighbors. We let $n = 4$, explicitly draw the grid (Figure 21.16) and, from that, determine the form of A and b. The form for larger values of n follows the same pattern.

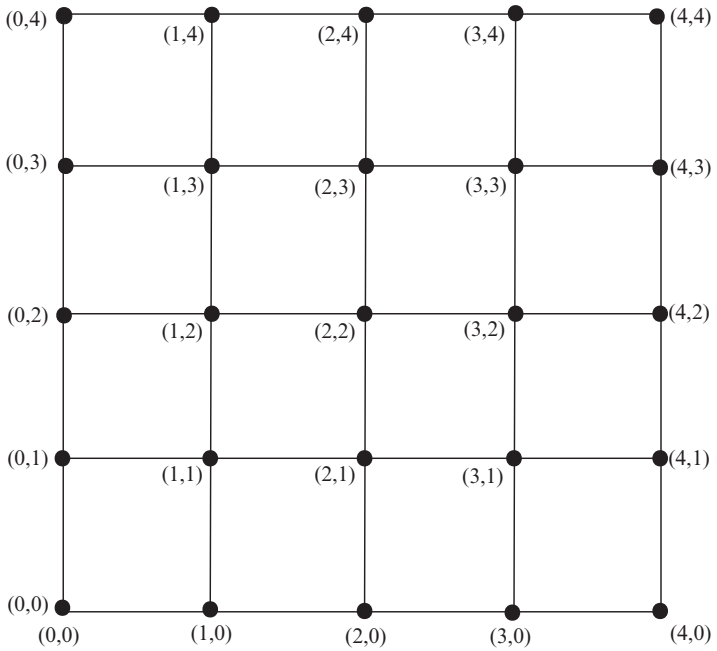


FIGURE 21.16 Poisson's equation grid for $n = 4$.

We adopt the notation $f(x_i, y_j) = f_{ij}$. Cycle through the grid by rows, applying Equation 21.30, and obtain nine equations in the nine unknowns $\begin{bmatrix} u_{11} & u_{21} & u_{31} & \dots & u_{13} & u_{23} & u_{33} \end{bmatrix}^{\mathrm{T}}$:

$$\begin{aligned}
4u_{11} - u_{21} - u_{12} &= h^2 f_{11} + g(0, y_1) + g(x_1, 0) \\
-u_{11} + 4u_{2,1} - u_{31} - u_{22} &= h^2 f_{21} + g(x_2, 0) \\
-u_{2,1} + 4u_{3,1} - u_{3,2} &= h^2 f_{31} + g(x_3, 0) + g(x_4, y_1) \\
-u_{11} + 4u_{12} - u_{22} - u_{1,3} &= h^2 f_{12} + g(0, y_2) \\
-u_{21} - u_{12} + 4u_{22} - u_{32} - u_{2,3} &= h^2 f_{22} \\
-u_{31} - u_{22} + 4u_{32} - u_{33} &= h^2 f_{32} + g(x_4, y_2) \\
-u_{12} + 4u_{13} - u_{23} &= h^2 f_{13} + g(0, y_3) + g(x_1, y_4) \\
-u_{22} - u_{13} + 4u_{23} - u_{33} &= h^2 f_{23} + g(x_2, y_4) \\
-u_{32} - u_{23} + 4u_{33} &= h^2 f_{33} + g(x_4, y_3) + g(x_3, y_4)
\end{aligned}$$

In matrix form, the set of equations is

$$\begin{bmatrix}
4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & -4 & 0 & 0 & -1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\
0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\
0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4
\end{bmatrix}
\begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \\ u_{12} \\ u_{22} \\ u_{32} \\ u_{13} \\ u_{23} \\ u_{33} \end{bmatrix}
=
\begin{bmatrix}
h^2 f_{11} + g(0, y_1) + g(x_1, 0) \\
h^2 f_{21} + g(x_2, 0) \\
h^2 f_{31} + g(x_3, 0) + g(x_4, y_1) \\
h^2 f_{12} + g(0, y_2) \\
h^2 f_{22} \\
h^2 f_{32} + g(x_4, y_2) \\
h^2 f_{13} + g(0, y_3) + g(x_1, y_4) \\
h^2 f_{23} + g(x_2, y_4) \\
h^2 f_{33} + g(x_4, y_3) + g(x_3, y_4)
\end{bmatrix}$$

The coefficient matrix has a definite pattern. It is called a *block tridiagonal matrix*. Let

$$T = \begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}.$$

If I is the 3×3 identity matrix, we can write the coefficient matrix using block matrix notation as

$$\begin{bmatrix} T & -I & 0 \\ -I & T & -I \\ 0 & -I & T \end{bmatrix}.$$

In general, if $h = \frac{1}{n}$, the $(n-1)^2 \times (n-1)^2$ matrix for the numerical solution of Poisson's equation has the form

$$\begin{bmatrix}
T & -I & & & & \\
-I & T & -I & & & \\
 & -I & T & -I & & \\
 & & \ddots & & & \\
 & & -I & T & -I \\
 & & & -I & T
\end{bmatrix}, \tag{21.31}$$

where I is the $(n-1) \times (n-1)$ identity matrix and T is the $(n-1) \times (n-1)$ tridiagonal matrix

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

This matrix can grow very large. For instance, if $n = 50$, the problem requires solving $49^2 = 2401$ equations. If we double n, there are approximately four times as many equations to solve. Standard Gaussian elimination will destroy the structure of the matrix, so iterative methods are normally used for this positive definite, ill-conditioned matrix. Problem 21.31 presents an interesting situation involving the Poisson equation.

21.12 THE BIHARMONIC EQUATION

The *biharmonic equation* is a fourth-order partial differential equation that is important in applied mechanics. It has applications in the theory of elasticity, mechanics of elastic plates, and the slow flow of viscous fluids [74]. The two-dimensional equation takes the form

$$\frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} = f(x, y), \tag{21.32}$$

with specified boundary conditions on a bounded domain. For our purposes, we assume the domain is the square $R = [0, 1] \times [0, 1]$ and that

$$u(x, y) = 0 \quad \text{and} \quad \frac{\partial u}{\partial n} = 0 \quad \text{on } \partial R.$$

The notation $\frac{\partial u}{\partial n}$ refers to the *normal derivative*, or $\langle \nabla u, n \rangle$, where n is a unit vector orthogonal to the surface. In our case, this becomes

$$\frac{\partial u}{\partial n} = \begin{cases} \frac{\partial u}{\partial x} = 0, & x = 0, \ x = 1 \\ \frac{\partial u}{\partial y} = 0, & y = 0, \ y = 1 \end{cases}.$$

The square can be partitioned into an $(n + 1) \times (n + 1)$ grid with equal steps, $h = \frac{1}{n}$, in the x and y directions. Use a 13-point central difference formula

$$\begin{aligned} &[\ 20u_{ij} - 8\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right) \\ &+2\left(u_{i-1,j} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1}\right) \\ &+\left(u_{i-2,j} + u_{i+2,j} + u_{i,j-2} + u_{i,j+2}\right)\ ]\ /h^4 \end{aligned}$$

to approximate $\frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4}$, and approximate the normal derivative on each side of the square by reflecting the first interior set of grid points across the boundary (Figure 21.17).

This results in an ill-conditioned positive definite $(n-1)^2 \times (n-1)^2$ *block pentadiagonal* coefficient matrix of the form

$$A = \frac{1}{h^4}\begin{bmatrix} \ddots & \ddots & \ddots & & & & \\ & \ddots & \ddots & \ddots & & & \\ \ddots & & \ddots & \ddots & \ddots & & \\ & D & C & B & C & D & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & & & \ddots & \end{bmatrix}.$$

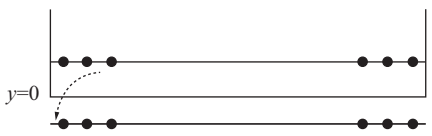


FIGURE 21.17 Estimating the normal derivative.

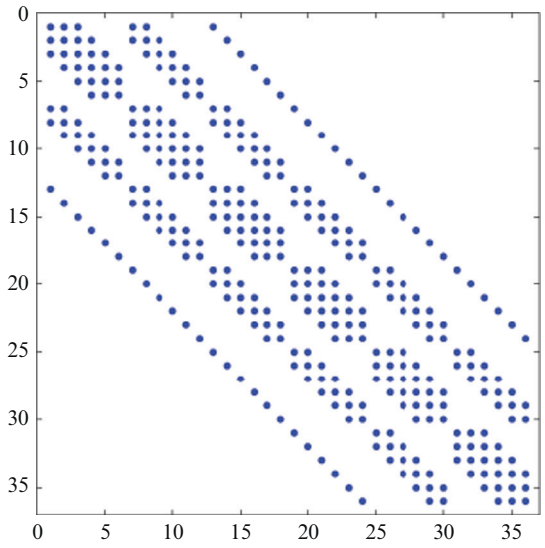


FIGURE 21.18 36×36 biharmonic matrix density plot.

Each block is of size $(n-1) \times (n-1)$. Block D is the identity matrix, C has the pattern $\begin{bmatrix} 2 & -8 & 2 \end{bmatrix}$, and B has the pattern $\begin{bmatrix} 1 & -8 & 21 & -8 & 1 \end{bmatrix}$. For $n = 7$, the blocks are

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} -8 & 2 & 0 & 0 & 0 & 0 \\ 2 & -8 & 2 & 0 & 0 & 0 \\ 0 & 2 & -8 & 2 & 0 & 0 \\ 0 & 0 & 2 & -8 & 2 & 0 \\ 0 & 0 & 0 & 2 & -8 & 2 \\ 0 & 0 & 0 & 0 & 2 & -8 \end{bmatrix},$$

$$B = \begin{bmatrix} 21 & -8 & 1 & 0 & 0 & 0 \\ -8 & 20 & -8 & 1 & 0 & 0 \\ 1 & -8 & 20 & -8 & 1 & 0 \\ 0 & 1 & -8 & 20 & -8 & 1 \\ 0 & 0 & 1 & -8 & 20 & -8 \\ 0 & 0 & 0 & 1 & -8 & 21 \end{bmatrix}$$

In the first and last occurrence of B, each diagonal element is incremented by 1. You can see the sparsity pattern in Figure 21.18, a density plot of the 36×36 biharmonic matrix.

In the software distribution, the functions `biharmonic_op` and `biharmonic` build the biharmonic coefficient matrix and use the preconditioned CG method to approximate the solution, respectively. Ref. [75, p. 385] provides an exact solution, $u(x, y) = 2350x^4 (x-1)^2 y^4 (y-1)^2$ to Equation 21.32 when

$$\begin{aligned} f(x, y) &= 56,400\left(1 - 10x + 15x^2\right)(1-y)^2 y^4 + 18,800x^2\left(6 - 20x + 15x^2\right) y^2\left(6 - 20y + 15y^2\right) \\ &\quad +56,400\,(1-x)^2 x^4\left(1 - 10y + 15y^2\right) \end{aligned}$$

Figure 21.19(a) and (b) shows a surface plot of the approximate and exact solutions, respectively. MATLAB code for this problem is in the directory "Text Examples/Chapter 21/biharmonicprob.m".

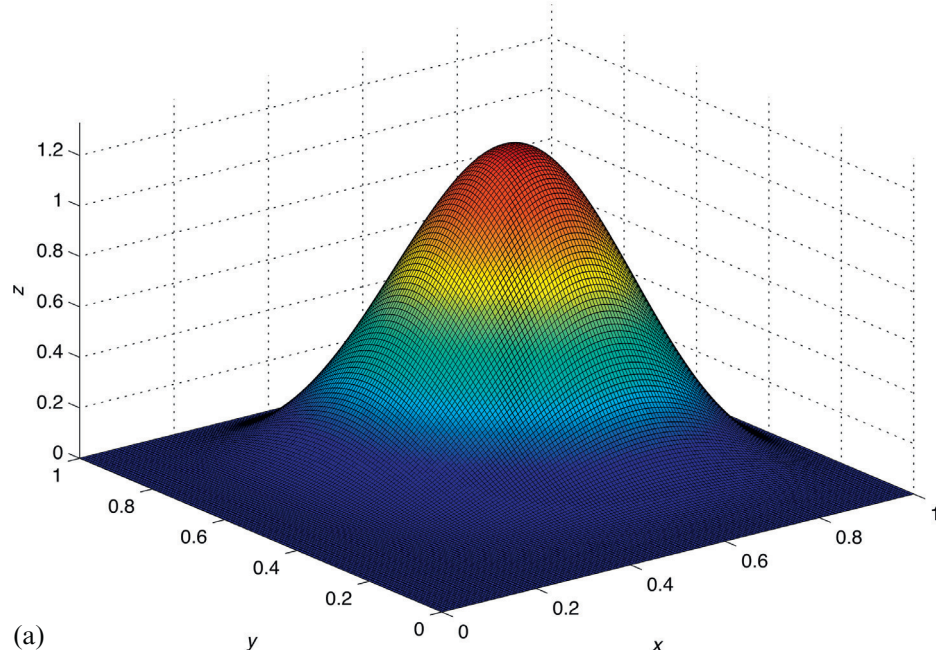
For more information about the biharmonic equation, see Refs. [29, 74].

21.13 CHAPTER SUMMARY

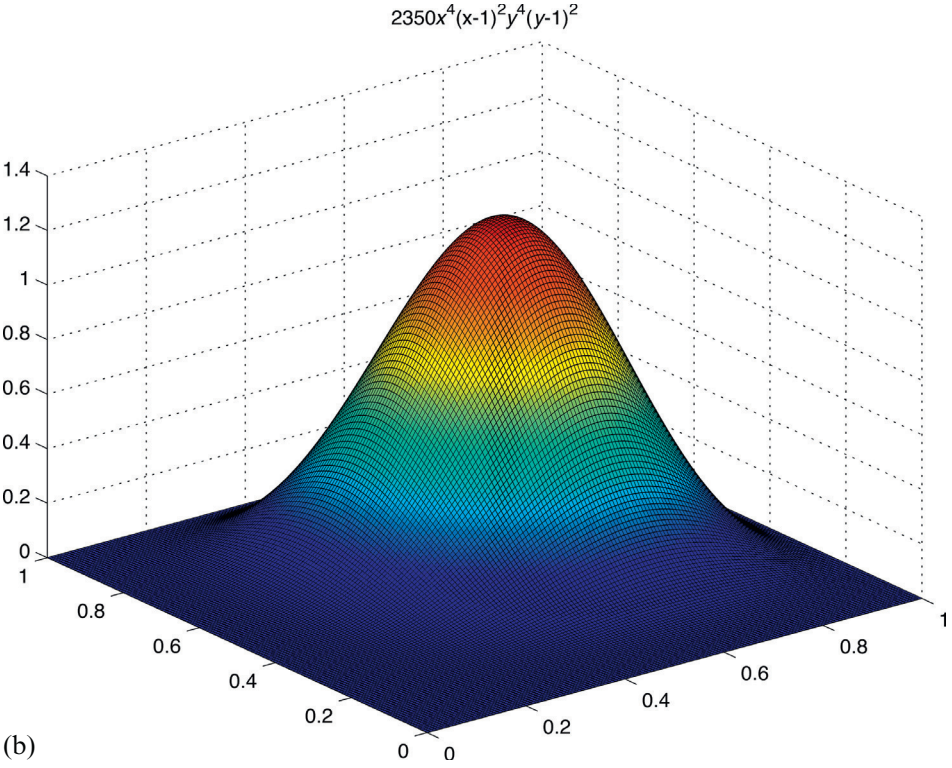
Large, Sparse Matrices

The primary use of iterative methods is for computing the solution to large, sparse systems. Along with other problems, such systems occur in the numerical solution of partial differential equations. There are a number of formats used to store

$$\frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} = f\,(x,\, y)$$
$$u = 0,\ \frac{\partial u}{\partial n} = 0 \text{ on } \partial R = \{0 \le x,\, y \le 1\}$$



(a)



(b)

FIGURE 21.19 The biharmonic equation. (a) Biharmonic equation numerical solution and (b) biharmonic equation true solution.

sparse matrices. In each case, only the nonzero elements are recorded. We look at the *CRS format* that stores the nonzero elements by rows. The algorithm uses three vectors that specify the row, column, and value of each nonzero matrix entry.

The CG Method

The CG method approximates the solution to $Ax = b$, where A is symmetric positive definite. The CG method is an extremely clever improvement of the method of steepest descent. The minimum of the quadratic function $\phi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$ is the solution to $Ax = b$. The negative of the gradient, $-\nabla\phi$, points in the direction of greatest decrease of ϕ. In addition, the gradient is always orthogonal to the contour line at each point. The method of steepest descent takes a step toward the solution by moving in the direction of the gradient line using $x_{i+1} = x_i + \alpha_i r_i$, where α_i is a scalar chosen to minimize ϕ for each step, and r_i is the residual for the current step. Successive residuals are orthogonal, which creates a "zigzag" toward the minimum of $\phi(x)$. The CG method uses an iteration of the same form, $x_{i+1} = x_i + p_i r_i$, but determines the next search direction from the current residual and the previous search directions. Successive approximations are orthogonal relative to the A norm, $\|x\|_A = \sqrt{x^{\mathrm{T}}Ax}$, do not "zigzag," and move much faster to the minimum of $\phi(x)$ than steepest descent.

Preconditioning

Iterative methods can converge very slowly if the coefficient matrix is ill-conditioned. Preconditioning involves finding a matrix, M, that approximates A. In this case, $M^{-1}A$ will have a lower condition number than A, and we can solve the equivalent system $M^{-1}Ax = M^{-1}b$, which is termed left-preconditioning. Other approaches are right- and split-preconditioning. Determination of a good matrix M can be difficult.

Preconditioning for CG

Any preconditioner for CG must be symmetric positive definite. We discuss two methods for preconditioning, the incomplete Cholesky decomposition and the SSOR preconditioner. The Cholesky approach can suffer from roundoff error problems, requiring the use of a drop tolerance, which decreases the sparsity of M. The SSOR preconditioner is inexpensive to compute and does not suffer from roundoff error to the same extent. However, incomplete Cholesky normally results in faster convergence.

Krylov Subspace Methods

The Krylov subspace $\mathcal{K}_m$ generated by A and u is span $\left\{ u \;\; Au \;\; A^2u \;\; \ldots \;\; A^{m-1}u \right\}$. It is of dimension m if the vectors are linearly independent. The Krylov subspace methods project the solution to the $n \times n$ problem, $Ax = b$, into a Krylov subspace $\mathcal{K}_m = \text{span}\left\{ r \;\; Ar \;\; A^2r \;\; \ldots \;\; A^{m-1}r \right\}$, where r is the residual and $m < n$.

Two Krylov subspace methods are discussed, the GMRES for the solution of any system and the MINRES for the solution of a symmetric indefinite system. The Arnoldi and Lanczos factorizations are required for the development of GMRES and MINRES, respectively. The solution to a least-squares problem in the Krylov subspace yields the solution to the n x n problem.; If the symmetric matrix is positive definite, CG or preconditioned CG is normally used. In the case of GMRES, the incomplete LU decomposition preconditioner is developed.

Comparison of Iterative Methods

Solving a large, sparse system can be very difficult, and there are many methods available. In the book, we have presented some of the most frequently used methods. Figure 21.15 is a diagram of possible paths that can be used for solving a particular problem.

The Biharmonic Equation

The biharmonic equation is a fourth-order partial differential equation that is important in applied mechanics. A 13-point central finite difference approximation results in a block pentadiagonal matrix that is symmetric positive definite and ill-conditioned. Preconditioned CG is effective in computing a solution to the biharmonic equation. A problem with a known solution is solved for the boundary conditions $u(x, y) = 0$ and $\frac{\partial u}{\partial n} = 0$ on ∂R.

21.14 PROBLEMS

21.1 Fill-in the table by describing the characteristics of each iterative method. Briefly note when convergence is guaranteed and its speed relative to the other methods.

Method	Properties
Jacobi	
Gauss-Seidel	
SOR	
CG	
GMRES	
MINRES	

21.2 If $A \in \mathbb{R}^{n\times n}$, $x \in \mathbb{R}^n$, and $\phi(x) = \frac{1}{2}x^{\mathrm{T}}Ax - x^{\mathrm{T}}b$, show that the gradient of ϕ,

$$\nabla\phi = \begin{bmatrix} \frac{\partial\phi}{\partial x_1} & \frac{\partial\phi}{\partial x_2} & \cdots & \frac{\partial\phi}{\partial x_{n-1}} & \frac{\partial\phi}{\partial x_n} \end{bmatrix}^{\mathrm{T}},$$

is

$$\nabla\phi(x) = \frac{1}{2}A^{\mathrm{T}}x + \frac{1}{2}Ax - b.$$

21.3

a. Show that $\langle x, y\rangle_A = x^{\mathrm{T}}Ay$ is an inner product, where A is a positive definite matrix.

b. If A is symmetric, there is an orthonormal basis, v_i, $1 \le i \le n$, of eigenvectors. Show that any two distinct basis vectors are A-orthogonal.

21.4 The technique of completing the square is another way to show that the solution, $\bar{x}$, to $Ax = b$ minimizes ϕ (Equation 21.1).

a. Show that for arbitrary vectors x, y, and symmetric matrix A, $x^{\mathrm{T}}Ay = (Ay)^{\mathrm{T}}x = y^{\mathrm{T}}Ax$.

b. From part (a), we know that $\bar{x}^{\mathrm{T}}Ax = x^{\mathrm{T}}A\bar{x}$, where $\bar{x}$ is the solution to $Ax = b$. By completing the square show that

$$\phi(x) = \frac{1}{2}(x - \bar{x})^{\mathrm{T}}A(x - \bar{x}) - \frac{1}{2}\bar{x}^{\mathrm{T}}A\bar{x}.$$

c. Assume A is positive definite. Using the result of part (b), argue that the minimum for $\phi(x)$ occurs when $x = \bar{x}$.

21.5 Show that if $A = R^{\mathrm{T}}R$ is the Cholesky decomposition of the positive definite matrix A, then $\overline{A} = \left(R^{-1}\right)^{\mathrm{T}}AR^{-1}$ is symmetric.

21.6 Verify Equation 21.25, $AQ_m = Q_mH_m + h_{m+1.m}q_{m+1}e_m^{\mathrm{T}}$, for the Arnoldi iteration.

21.7 The diagonal preconditioner, also called the *Jacobi preconditioner*, is one of the simplest means for reducing the condition number of the coefficient matrix. The method is particularly effective for a diagonally dominant matrix or a matrix with widely different diagonal elements but, of course, the method does not always help.

a. Let M be the diagonal of A and use M^{-1} as the preconditioner. Develop a formula for the method that does not involve computing M^{-1}.

b. Let $A = \begin{bmatrix} 20 & 20.01 \\ 9.99 & 10 \end{bmatrix}$. Compute the condition number of the coefficient matrix before and after using diagonal preconditioning.

c. What happens if all elements of the diagonal are constant?

21.8 Show that the eigenvalues of a positive definite matrix are the same as its singular values.

21.9 Show that the SSOR preconditioning matrix $M_{\mathrm{SSOR}}(\omega)$ (Equation 21.22) is symmetric positive definite.

21.10 SSOR can be used as a preconditioner for GMRES. If we choose $\omega = 1$, the matrix $M_{\mathrm{PGS}} = (D + L)D^{-1}(D + U)$ serves as a preconditioner. Using the development of pregmres as a guide, develop equations for the preconditioned matrix, $\overline{A}$, the right-hand side, $\overline{b}$, and the solution, x. HINT: The matrix expression

$$D(D + L)^{-1}A(D + U)^{-1}$$

can be evaluated without computing an inverse using the matrix operators \ and / as follows:

$$(I + L/D) \setminus (A/(D + U)).$$

The definition of the operators is:

$$x = A\backslash b$$

solves

$$Ax = b,$$

and

$$x = b/A$$

solves

$$xA = b.$$

In each case, x and b can be matrices. The book has not discussed the solution to a system of the form $xA = b$, since this type of system is rarely seen. A look at the 2×2 or 3×3 case for A and b will reveal a method of solution.

21.11 Verify Equations 21.18 and 21.19.

21.12 Develop Equation 21.20.

21.13 Two bases $V = \left\{ v_1 \ \ldots \ v_m \right\}$ and $W = \left\{ w_1 \ \ldots \ w_m \right\}$ are *biorthogonal* if

$$\langle v_i, w_j \rangle = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}.$$

Show that if the $m \times m$ matrices P and Q have columns formed from the vectors in V and W, respectively, then $P^{\mathrm{T}}Q = I$. Biorthogonality is a primary component in the Bi-CG iteration for general sparse matrices. See Problem 21.34.

21.14.1 MATLAB Problems

21.14 Load the positive definite 100×100 sparse matrix ACG from the software distribution. Let b be a random 100×1 vector, $x_0 = 0$, and use `cg` to approximate the solution to $ACGx = b$. Using the same tolerance and maximum number of iterations, approximate the solution using `precg`. Compute the residual for each iteration.

21.15 When a nonsymmetric matrix is well conditioned and it is feasible to compute A^{T}, then CG can be applied to the normal equations.

- **a.** Write a MATLAB `function [x r iter] = normalsolve(A,b,x0,tol,numiter)` that uses the normal equation approach to solving $Ax = b$.
- **b.** The MATLAB command `R = sprandn(m,n,density,rc)` generates a random sparse matrix of size $m \times n$, where `density` is the fraction of nonzeros and the reciprocal of `rc` is the approximate condition number. Generate b randomly, and set `x0 = ones(size,1)`. Using an error tolerance of 1.0×10^{-10} and maximum number of iterations set to 500, compute the residual and number of iterations required by `normalsolve` for each sparse system.
  - **i.** R1 = sprandn(1000,1000,0.03,0.10)
  - **ii.** R2 = sprandn(1500,1500,0.05,0.01)
  - **iii.** R3 = sprandn(2000,2000,0.2,0.001)
- **c.** Add the normal equation approach to the appropriate locations in Figure 21.15.

21.16 The MATLAB function `sprandsym` generates a random symmetric matrix and will additionally make the matrix positive definite. The statement

```
R = sprandsym(n,density,rc,1);
```

generates a sparse random symmetric positive definite matrix of size $n \times n$, where `density` is the fraction of nonzeros, and the reciprocal of `rc` is the exact condition number. Using `spransym`, generate a sparse symmetric positive definite matrix of dimension 500×500, with 3% zeros, and condition number of 50. Execute CG and preconditioned CG using random b, random x_0, tol $= 1.0 \times 10^{-12}$, and numiter $= 100$. For preconditioned CG, use incomplete Cholesky with droptol $= 1.0 \times 10^{-4}$. Compare the time required and the resulting residual.

For the remaining problems, unless stated otherwise, use $x_0 = 0$ and let b be a random $n \times 1$ vector.

21.17 Load the 1138×1138 matrix HB1138 used in the solution of a power network problem.

- **a.** Verify it is positive definite, and approximate its condition number.
- **b.** Apply CG with tol $= 1.0 \times 10^{-10}$ and maxiter $= \{500, 1000, 3000, 5000, 10000\}$.
- **c.** Apply preconditioned CG with 500 iterations.
- **d.** Apply SSOR preconditioning. You will need to experiment with the maximum number of iterations required.

21.18 Using $m = 3$, compute the Arnoldi decomposition for the matrix

$$A = \begin{bmatrix} 1 & 3 & -1 & 7 & 2 \\ 5 & -8 & 25 & 3 & 12 \\ 0 & -1 & 0 & 3 & 7 \\ 8 & -3 & 23 & 6 & 9 \\ 56 & 13 & 8 & -9 & 1 \end{bmatrix}.$$

21.19 Using $m = 3$, compute the Lanczos decomposition for the symmetric matrix

$$A = \begin{bmatrix} 1 & 5 & 3 & -1 & 6 \\ 5 & 1 & 7 & -8 & 2 \\ 3 & 7 & 12 & -1 & 3 \\ -1 & -8 & -1 & 9 & 4 \\ 6 & 2 & 3 & 4 & 1 \end{bmatrix}.$$

21.20 Load the 2500×2500 matrix SYM2500.

a. Verify it is symmetric but not positive definite, and approximate its condition number.

b. Apply `minresb` with tol $= 1.0 \times 10^{-6}$, niter = 10, and $m = \{50, 100, 150, 200, 250\}$. Time each execution, and comment on the result

21.21 Load the 1024×1024 matrix DIMACS10 used in an undirected graph problem.

a. Verify it is symmetric but not positive definite, and approximate its condition number.

b. Apply `minresb` with tol $= 1.0 \times 10^{-6}$, maxiter = 50, and $m = \{50, 100, 250, 350\}$. Using $m = 50$ with all the other parameters the same, estimate the solution using mpregmres. Time each execution, and comment on the results.

21.22 Load the 1080×1080 matrix SHERMAN2 used in a computational fluid dynamics problem.

a. Verify it is not symmetric, and approximate its condition number.

b. Apply `gmresb` with tol $= 1.0 \times 10^{-6}$, maxiter = 10, and $m = \{500, 800\}$. Time each execution.

c. Apply `mpregmres` with tol $= 1.0 \times 10^{-6}$, niter = 10, and $m = 500$. Time the execution, and comment on the result and those of part (b).

21.23 The software distribution contains a MATLAB sparse matrix file for each of three matrices:

a. ACG.mat **b.** Si.mat **c.** west0479.mat

Solve each system as best you can without preconditioning. Then solve each system using preconditioning.

21.24

a. Using the statement `A = sprandsym(2000,0.05,.01)`, create a 2000×2000 symmetric indefinite matrix with a condition number of 100. Solve the system using `minresb`.

b. Using `A = sprandsym(2000,0.05,.001)`, create a symmetric indefinite matrix with a condition number of 1000. Solve the system using both `minresb` and `mpregmres`. Use $m = 500$ and tol $= 1.0 \times 10^{-6}$.

21.25 As the drop tolerance decreases when using Cholesky preconditioning, the matrix R becomes more dense.

a. Using `spy`, graph the complete Cholesky factor for the matrix ROOF in the software distribution.

b. Computing an incomplete Cholesky factorization of ROOF is particularly difficult. Try `icholesky` and then `ichol` with no drop tolerance.

c. Using a drop tolerance will make `ichol` successful. Using `spy`, demonstrate the increase in density using the MATLAB function `ichol` with drop tolerances of 1.0×10^{-4} and 1.0×10^{-5}.

d. What happens when you try drop tolerances of 1.0×10^{-2} and 1.0×10^{-3}?

21.26 The function `spdiags` extracts and creates sparse band and diagonal matrices and is a generalization of the MATLAB function `diag`. We will only use one of its various forms. To create a pentadiagonal matrix, create vectors that specify the diagonals. We will call these diagonals d_1, d_2, d_3, d_4, d_5, where d_1, d_2 are the subdiagonals and d_4, d_5 are the superdiagonals. For an $n \times n$ matrix, each vector must have length n. To create the matrix, execute

```
>> A = spdiags([d1 d2 d3 d4 d5], -2:2, n, n);
```

The parameter `-2:2` specifies that the diagonals are located at offsets -2, -1, 0, 1, and 2 from the main diagonal.

a. Create a $10,000 \times 10,000$ pentadiagonal matrix with sub- and superdiagonals $\begin{bmatrix} -1 & -1 & \dots & -1 & -1 \end{bmatrix}^{\mathrm{T}}$ and diagonal $\begin{bmatrix} 4 & 4 & \dots & 4 & 4 \end{bmatrix}^{\mathrm{T}}$. Show the matrix is positive definite, with an approximate condition number of 2.0006×10^7.

b. Using $x_0 = 0$ and a random b, apply `cg` using tol $= 1.0 \times 10^{-14}$ and maxiter = [10 100 1000 10000 20000]. In each case, output the residual.

c. Use `precg` with the same tolerance and niter = 100. Output the residual.

21.27 The biharmonic matrix is ill-conditioned. Verify this for matrices with $n =$ [10 50 100 500] over the interval $0 \le x, y \le 1$.

21.28 We will investigate the one-dimensional version of the biharmonic equation:

$$\frac{\mathrm{d}^4 u}{\mathrm{d}x^4} = f(x). \tag{21.33}$$

Assuming the boundary conditions

$$u(0) = u(1) = \frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(0) = \frac{d^2 u}{dx^2} = 0, \quad 0 \le x \le 1$$

and using a five-point central finite-difference approximation over $0 \le x \le 1$ with uniform step size $h = 1/n$, gives rise to the following pentadiagonal $(n-1) \times (n-1)$ matrix [29]:

$$A = \frac{1}{h^4}\begin{bmatrix} 5 & -4 & 1 & & & 0 \\ -4 & 6 & -4 & 1 & \ddots & \\ 1 & -4 & 6 & -4 & 1 & \ddots \\ \vdots & \ddots & & & & \ddots \\ \dots & 1 & -4 & 6 & -4 & 1 \\ \vdots & & & & & \ddots \\ 0 & & \cdots & & 1 & -4 \quad 5 \end{bmatrix}$$

A is symmetric positive definite and ill-conditioned, so the preconditioned CG method should be used.

a. Write a function `[x, y, u, residual, iter] = biharmonic1D(n, f, tol,maxiter)` that uses preconditioned CG with n subintervals, specified tolerance and maximum number of iterations to approximate a solution to Equation 21.33 and graph it.

b. Consider the specific problem

$$\frac{\mathrm{d}^4 u}{\mathrm{d}x^4} = x, \quad 0 \le x \le 1,$$

$$u(0) = u(1) = \frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(0) = \frac{\mathrm{d}^2 u}{\mathrm{d}x^2}(1) = 0,$$

whose exact solution is $u(x) = \frac{1}{120}x^5 - \frac{1}{36}x^3 + \frac{7}{360}x$. Use $h = \frac{1}{100}$, that requires solving a 99×99 pentadiagonal system having 489 diagonal entries, so the density of nonzero entries is 4.99%.

21.29

a. The Jacobi diagonal preconditioner (Problem 21.7) performs best when the coefficient matrix is strictly diagonally dominant and has widely varying diagonal entries. Write a function `jacobipregmres` that implements Jacobi preconditioning for the GMRES method. Return with a message if the preconditioned matrix does not have a condition number that is at most 80% of the condition number for A.

b. Build the 1000×1000 matrix, T, with seven diagonals:

$$a_i = -1,\ b_i = 2,\ c_i = 4,\ d_i = 20 + 100(i-1),\ e_i = 4, f_i = 2,\ g_i = -1,\ 1 \le i \le 1000.$$

Compute the condition number of T. Using rhs $= \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, tol $= 1.0 \times 10^{-6}$, and $m = 100$, compute the solution using both `gmresb` and `jacobipregmres`. Display the residuals.

21.30 Example 21.7 applies `precg` to the matrix `PRECGTEST` in the software distribution. Modify `precg` and create the function `precgomega` with the following calling sequence:

```
% If method = 'incomplete Cholesky', parm is the drop tolerance.
% If method = 'SSOR', parm is omega.
[x, residual, iter] = precgomega(A,b,x0,tol,maxiter,method,parm);
```

Rather than assuming $\omega = 1$ in Equation 21.22, maintain the equation as a function of ω. Apply `precgomega` with the SSOR preconditioner to PRECGTEST using different values of ω and see if you can improve upon the result in Example 21.7.

21.31 The ill-conditioned Poisson matrix 21.11 developed for the five-point central finite difference approximation to the solution of Poisson's equation is positive definite, so the preconditioned CG method applies. To use CG, we will need to construct the block tridiagonal matrix. Fortunately, MATLAB builds the sparse $n^2 \times n^2$ matrix with the command

```
P = gallery('poisson', n);
```

a. Modify the Poisson equation solver `sorpoisson` referenced in Section 20.5 to use the preconditioned CG method, where $\partial R = \{0 \le x, y \le 1\}$. The function declaration should be `function [x y u] = cgpoisson(n,f,g,numiter)`.

b. Test `cgpoisson` using the following problem with $n = 75$, maxiter $= 100$, and tol $= 1.0 \times 10^{-10}$.

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = xy, u\,(x,\, y) = 0 \quad \text{on } \partial R = [0,\, 1] \times [0, 1\,]$$

c. The following Poisson problem describes the electrostatic potential field induced by charges in space, where u is a potential field and ρ is a charge density function.

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 4\pi\rho$$
$$u\,(x,\, y) = g\,(x,\, y) \quad \text{on } \partial R.$$

Assume that the electrostatic potential fields are induced by approximately 15 randomly placed point charges with strength 1 ($\rho = 1$). The edges are grounded, so $u\,(x,\, y) = 0$ on $\partial R = \{0 \le x, y \le 1\}$. Note that the right-hand side function $4\pi\rho$ affects only the points inside the boundary. Solve the problem using $n = 100$, and draw a surface plot and a contour plot of the result (Figure 21.20 shows sample plots). One way of coding the right-hand side is

```
function r = rho(~,~)
%r = rho(x,y) generates a point charge at random points.
%
%   generate a point charge of strength 1 at approximately
%   3/2000 points (x,y) in the plane. this causes a point charge
%   of approximately 1/15 of the points in the plot of the voltage
%   produced by 'potentialsolve.m'.

p = randi([1 2000],1,1);
if p == 500 || p == 250 || p == 750
    r = 4 * pi;
else
    r = 0;
end
```

21.32

a. Using the results of Problem 21.10, develop `function [x, r, iter] = ssorpregmres(A,b,x0,m,tol, maxiter)` that uses SSOR as a preconditioner for GMRES.

b. Ref. [64, p. 97] discusses the nonsymmetric matrix ORSIIR_1 with dimension 1030×1030 that arises from a reservoir engineering problem. Let $x_0 = 0$, $b = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}$, $m = 150$, and tol $= 1.0 \times 10^{-6}$. Solve the system ORSII_1$x = b$ using the functions `gmresb`, `mpregmres`, and `ssorpregmres`.

c. The positive definite matrix ROOF presented in Problem 21.25 required experimenting with a drop tolerance when preconditioning using the incomplete Cholesky decomposition. Using $b = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, $x_0 = 0$, solve ROOF$x = b$ using ssorpregmres with tol $= 1.0 \times 10^{-6}$, $m = 100$, and maxiter $= 15$.

21.33 MATLAB provides various solvers for large, sparse problems.

a. The MATLAB function `gmres` implements the GMRES algorithm. Look it up using the help system and apply it to the nonsymmetric matrix `DK01R` from the software distribution, using the right-hand side `b_DK01R`. You will need to precondition the matrix using a drop tolerance with the statement

```
[L,U] = ilu(DK01R,struct('type','ilutp','droptol',droptol));
```

b. The MATLAB function `minres` implements the MINRES algorithm. Use it with the matrix `bcsstm10` from the software distribution with $b = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, $x_0 = 0$. Use sufficiently many iterations so it succeeds with an error tolerance of 1.0×10^{-6}.

c. The MATLAB function `symmlq` is a general sparse symmetric matrix solver based upon solving a symmetric tridiagonal system. Use it with the matrix `bcsstm10` from the software distribution with $b = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^{\mathrm{T}}$, $x_0 = 0$. Use sufficiently many iterations so it succeeds with an error tolerance of 1.0×10^{-6}.

21.34 MATLAB provides functions we have not discussed in the book that implement sparse system solvers for general matrices. In this problem, you will investigate two such algorithms.

a. The bi-CG method is a CG-type method. The bi-CG method generates two sets of residual sequences, $\{r_i\}$ and $\{\overline{r_i}\}$, which are biorthogonal (Problem 2.13). Two sets of direction vectors, $\{p_i\}$ and $\{\overline{p_i}\}$ are computed from the

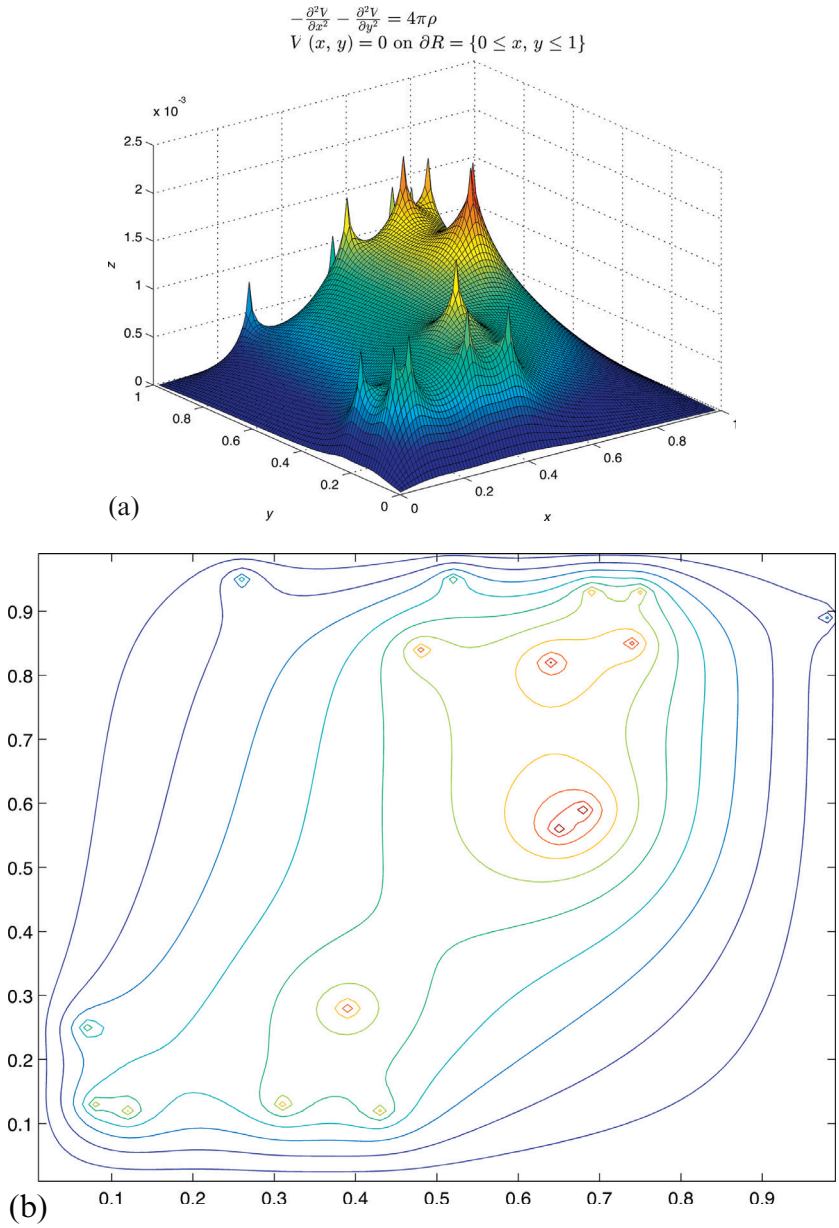


FIGURE 21.20 (a) Electrostatic potential fields induced by approximately 15 randomly placed point charges (b) contour plot of randomly placed point charges.

residuals. The function `bicg` implements the algorithm. Try it with the matrix TOLS1090 from the software distribution. Preconditioning is necessary.

b. The QMR method applies to general matrices and functions by working with a nonsymmetric tridiagonal matrix. Repeat part (a) using `qmr`.

21.35 Let $R = [0,\ 1] \times [0,\ 1]$, and graph the approximate solution to the biharmonic equation

$$\begin{aligned} \frac{\partial^4 u}{\partial x^4} + 2\frac{\partial^4 u}{\partial x^2 \partial y^2} + \frac{\partial^4 u}{\partial y^4} &= \sin(\pi x)\sin(\pi y), \\ u(x, y) &= 0 \quad \text{on } \partial R \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial R \end{aligned}$$

21.36 Reproduce Figures 21.13(a), (b), and console output for Example 21.10.