Numerical Analysis

SECOND EDITION

Timothy Sauer
George Mason University

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River Amsterdam Cape Town
Dubai London Madrid Milan Munich Paris Montréal Toronto Delhi Mexico City São Paulo
Sydney Hong Kong Seoul Singapore Taipei Tokyo

2.5 ITERATIVE METHODS

Gaussian elimination is a finite sequence of $O(n^3)$ floating point operations that result in a solution. For that reason, Gaussian elimination is called a **direct** method for solving systems of linear equations. Direct methods, in theory, give the exact solution within a finite number of steps. (Of course, when carried out by a computer using limited precision, the resulting solution will be only approximate. As we saw earlier, the loss of precision is quantified by the condition number.) Direct methods stand in contrast to the root-finding methods described in Chapter 1, which are iterative in form.

So-called **iterative** methods also can be applied to solving systems of linear equations. Similar to Fixed-Point Iteration, the methods begin with an initial guess and refine the guess at each step, converging to the solution vector.

2.5.1 Jacobi Method

The Jacobi Method is a form of fixed-point iteration for a system of equations. In FPI the first step is to rewrite the equations, solving for the unknown. The first step of the Jacobi Method is to do this in the following standardized way: Solve the ith equation for the ith unknown. Then, iterate as in Fixed-Point Iteration, starting with an initial guess.

▶ EXAMPLE 2.19 Apply the Jacobi Method to the system $3u + v = 5, u + 2v = 5$.

Begin by solving the first equation for u and the second equation for v. We will use the initial guess $(u_0, v_0) = (0, 0)$. We have

$$\begin{aligned} u &= \frac{5 - v}{3} \\ v &= \frac{5 - u}{2}. \end{aligned} \tag{2.35}$$

The two equations are iterated:

$$\begin{aligned} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix} \\ \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/2}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix} \\ \begin{bmatrix} u_3 \\ v_3 \end{bmatrix} &= \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-5/6}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}. \end{aligned} \tag{2.36}$$

Further steps of Jacobi show convergence toward the solution, which is $[1, 2]$. ◀

Now suppose that the equations are given in the reverse order.

▶ EXAMPLE 2.20 Apply the Jacobi Method to the system $u + 2v = 5, 3u + v = 5$.

Solve the first equation for the first variable u and the second equation for v. We begin with

$$\begin{aligned} u &= 5 - 2v \\ v &= 5 - 3u. \end{aligned} \tag{2.37}$$

The two equations are iterated as before, but the results are quite different:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$
$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$$
$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}. \tag{2.38}$$

In this case the Jacobi Method fails, as the iteration diverges. ◀

Since the Jacobi Method does not always succeed, it is helpful to know conditions under which it does work. One important condition is given in the following definition:

DEFINITION 2.9 The $n \times n$ matrix $A = (a_{ij})$ is **strictly diagonally dominant** if, for each $1 \leq i \leq n$, $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$. In other words, each main diagonal entry dominates its row in the sense that it is greater in magnitude than the sum of magnitudes of the remainder of the entries in its row. ❒

THEOREM 2.10 If the $n \times n$ matrix A is strictly diagonally dominant, then (1) A is a nonsingular matrix, and (2) for every vector b and every starting guess, the Jacobi Method applied to $Ax = b$ converges to the (unique) solution. ■

Theorem 2.10 says that, if A is strictly diagonally dominant, then the Jacobi Method applied to the equation $Ax = b$ converges to a solution for each starting guess. The proof of this fact is given in Section 2.5.3. In Example 2.19, the coefficient matrix is at first

$$A = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix},$$

which is strictly diagonally dominant because $3 > 1$ and $2 > 1$. Convergence is guaranteed in this case. On the other hand, in Example 2.20, Jacobi is applied to the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix},$$

which is not diagonally dominant, and no such guarantee exists. Note that strict diagonal dominance is only a sufficient condition. The Jacobi Method may still converge in its absence.

► EXAMPLE 2.21 Determine whether the matrices

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 2 & -5 & 2 \\ 1 & 6 & 8 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 3 & 2 & 6 \\ 1 & 8 & 1 \\ 9 & 2 & -2 \end{bmatrix}$$

are strictly diagonally dominant.

The matrix A is diagonally dominant because $|3| > |1| + |-1|, |-5| > |2| + |2|$, and $|8| > |1| + |6|$. B is not, because, for example, $|3| > |2| + |6|$ is not true. However, if the first and third rows of B are exchanged, then B is strictly diagonally dominant and Jacobi is guaranteed to converge. ◀

The Jacobi Method is a form of fixed-point iteration. Let D denote the main diagonal of A, L denote the lower triangle of A (entries below the main diagonal), and U denote the upper triangle (entries above the main diagonal). Then $A = L + D + U$, and the equation to be solved is $Lx + Dx + Ux = b$. Note that this use of L and U differs from the use in the LU factorization, since all diagonal entries of this L and U are zero. The system of equations $Ax = b$ can be rearranged in a fixed-point iteration of form:

$$\begin{aligned} Ax &= b \\ (D + L + U)x &= b \\ Dx &= b - (L + U)x \\ x &= D^{-1}(b - (L + U)x). \end{aligned} \tag{2.39}$$

Since D is a diagonal matrix, its inverse is the matrix of reciprocals of the diagonal entries of A. The Jacobi Method is just the fixed-point iteration of (2.39):

Jacobi Method

$$\begin{aligned} x_0 &= \text{ initial vector} \\ x_{k+1} &= D^{-1}(b - (L + U)x_k) \quad \textbf{for} \quad k = 0, 1, 2, \ldots. \end{aligned} \tag{2.40}$$

For Example 2.19,

$$\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix},$$

the fixed-point iteration (2.40) with $x_k = \begin{bmatrix} u_k \\ v_k \end{bmatrix}$ is

$$\begin{aligned} \begin{bmatrix} u_{k+1} \\ v_{k+1} \end{bmatrix} &= D^{-1}(b - (L + U)x_k) \\ &= \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_k \\ v_k \end{bmatrix} \right) \\ &= \begin{bmatrix} (5 - v_k)/3 \\ (5 - u_k)/2 \end{bmatrix}, \end{aligned}$$

which agrees with our original version.

2.5.2 Gauss–Seidel Method and SOR

Closely related to the Jacobi Method is an iteration called the **Gauss–Seidel** Method. The only difference between Gauss–Seidel and Jacobi is that in the former, the most recently updated values of the unknowns are used at each step, even if the updating occurs in the current step. Returning to Example 2.19, we see that Gauss–Seidel looks like this:

$$\begin{aligned} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-5/3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{3} \end{bmatrix} \\ \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-5/3}{3} \\ \frac{5-10/9}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{35}{18} \end{bmatrix} \\ \begin{bmatrix} u_3 \\ v_3 \end{bmatrix} &= \begin{bmatrix} \frac{5-v_2}{3} \\ \frac{5-u_3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-35/18}{3} \\ \frac{5-55/54}{2} \end{bmatrix} = \begin{bmatrix} \frac{55}{54} \\ \frac{215}{108} \end{bmatrix}. \end{aligned} \tag{2.41}$$

Note the difference between Gauss–Seidel and Jacobi: The definition of v_1 uses u_1, not u_0. We see the approach to the solution $[1, 2]$ as with the Jacobi Method, but somewhat more accurately at the same number of steps. Gauss–Seidel often converges faster than Jacobi if the method is convergent. Theorem 2.11 verifies that the Gauss–Seidel Method, like Jacobi, converges to the solution as long as the coefficient matrix is strictly diagonally dominant.

Gauss–Seidel can be written in matrix form and identified as a fixed-point iteration where we isolate the equation $(L + D + U)x = b$ as

$$(L + D)x_{k+1} = -Ux_k + b.$$

Note that the usage of newly determined entries of x_{k+1} is accommodated by including the lower triangle of A into the left-hand side. Rearranging the equation gives the Gauss–Seidel Method.

Gauss–Seidel Method

$$x_0 = \text{initial vector}$$
$$x_{k+1} = D^{-1}(b - Ux_k - Lx_{k+1}) \quad \textbf{for } k = 0, 1, 2, \ldots.$$

▶ EXAMPLE 2.22 Apply the Gauss–Seidel Method to the system

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}.$$

The Gauss–Seidel iteration is

$$u_{k+1} = \frac{4 - v_k + w_k}{3}$$
$$v_{k+1} = \frac{1 - 2u_{k+1} - w_k}{4}$$
$$w_{k+1} = \frac{1 + u_{k+1} - 2v_{k+1}}{5}.$$

Starting with $x_0 = [u_0, v_0, w_0] = [0, 0, 0]$, we calculate

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} \frac{4-0-0}{3} = \frac{4}{3} \\ \frac{1-8/3-0}{4} = -\frac{5}{12} \\ \frac{1+4/3+5/6}{5} = \frac{19}{30} \end{bmatrix} \approx \begin{bmatrix} 1.3333 \\ -0.4167 \\ 0.6333 \end{bmatrix}$$

and

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{101}{60} \\ -\frac{3}{4} \\ \frac{251}{300} \end{bmatrix} \approx \begin{bmatrix} 1.6833 \\ -0.7500 \\ 0.8367 \end{bmatrix}.$$

The system is strictly diagonally dominant, and therefore the iteration will converge to the solution $[2, -1, 1]$. ◀

The method called **Successive Over-Relaxation (SOR)** takes the Gauss–Seidel direction toward the solution and "overshoots" to try to speed convergence. Let ω be a real

number, and define each component of the new guess x_{k+1} as a weighted average of ω times the Gauss–Seidel formula and $1-\omega$ times the current guess x_k. The number ω is called the **relaxation parameter**, and $\omega > 1$ is referred to as **over-relaxation**.

▶ EXAMPLE 2.23 Apply SOR with $\omega = 1.25$ to the system of Example 2.22.

Successive Over-Relaxation yields

$$u_{k+1} = (1-\omega)u_k + \omega\frac{4 - v_k + w_k}{3}$$
$$v_{k+1} = (1-\omega)v_k + \omega\frac{1 - 2u_{k+1} - w_k}{4}$$
$$w_{k+1} = (1-\omega)w_k + \omega\frac{1 + u_{k+1} - 2v_{k+1}}{5}.$$

Starting with $[u_0, v_0, w_0] = [0, 0, 0]$, we calculate

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} \approx \begin{bmatrix} 1.6667 \\ -0.7292 \\ 1.0312 \end{bmatrix}$$

and

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} \approx \begin{bmatrix} 1.9835 \\ -1.0672 \\ 1.0216 \end{bmatrix}.$$

In this example, the SOR iteration converges faster than Jacobi and Gauss–Seidel to the solution $[2, -1, 1]$. ◀

Just as with Jacobi and Gauss–Seidel, an alternative derivation of SOR follows from treating the system as a fixed-point problem. The problem $Ax = b$ can be written $(L + D + U)x = b$, and, upon multiplication by ω and rearranging,

$$\begin{aligned} (\omega L + \omega D + \omega U)x &= \omega b \\ (\omega L + D)x &= \omega b - \omega U x + (1-\omega)Dx \\ x &= (\omega L + D)^{-1}[(1-\omega)Dx - \omega U x] + \omega(D + \omega L)^{-1}b. \end{aligned}$$

Successive Over-Relaxation (SOR)

$$x_0 = \text{initial vector}$$
$$x_{k+1} = (\omega L + D)^{-1}[(1-\omega)Dx_k - \omega U x_k] + \omega(D + \omega L)^{-1}b \quad \textbf{for } k = 0, 1, 2, \ldots.$$

SOR with $\omega = 1$ is exactly Gauss–Seidel. The parameter ω can also be allowed to be less than 1, in a method called Successive Under-Relaxation.

▶ EXAMPLE 2.24 Compare Jacobi, Gauss–Seidel, and SOR on the system of six equations in six unknowns:

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}. \tag{2.42}$$

The solution is $x = [1, 1, 1, 1, 1, 1]$. The approximate solution vectors x_6, after running six steps of each of the three methods, are shown in the following table:

Jacobi	Gauss–Seidel	SOR
0.9879	0.9950	0.9989
0.9846	0.9946	0.9993
0.9674	0.9969	1.0004
0.9674	0.9996	1.0009
0.9846	1.0016	1.0009
0.9879	1.0013	1.0004

The parameter ω for Successive Over-Relaxation was set at 1.1. SOR appears to be superior for this problem. ◀

Figure 2.3 compares the infinity norm error in Example 2.24 after six iterations for various ω. Although there is no general theory describing the best choice of ω, clearly there is a best choice in this case. See Ortega [1972] for discussion of the optimal ω in some common special cases.

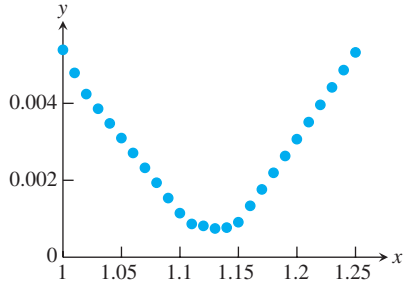

Figure 2.3 Infinity norm error after six steps of SOR in Example 2.24, as a function of over-relaxation parameter ω. Gauss–Seidel corresponds to $\omega = 1$. Minimum error occurs for $\omega \approx 1.13$

2.5.3 Convergence of iterative methods

In this section we prove that the Jacobi and Gauss–Seidel Methods converge for strictly diagonally dominant matrices. This is the content of Theorems 2.10 and 2.11.

The Jacobi Method is written as

$$x_{k+1} = -D^{-1}(L + U)x_k + D^{-1}b. \tag{2.43}$$

Theorem A.7 of Appendix A governs convergence of such an iteration. According to this theorem, we need to know that the spectral radius $\rho(D^{-1}(L + U)) < 1$ in order to guarantee convergence of the Jacobi Method. This is exactly what strict diagonal dominance implies, as shown next.

Proof of Theorem 2.10. Let $R = L + U$ denote the nondiagonal part of the matrix. To check $\rho(D^{-1}R) < 1$, let λ be an eigenvalue of $D^{-1}R$ with corresponding eigenvector v. Choose this v so that $||v||_\infty = 1$, so that for some $1 \le m \le n$, the component $v_m = 1$ and all other components are no larger than 1. (This can be achieved by starting with any eigenvector and dividing by the largest component. Any constant multiple of an eigenvector is again an eigenvector with the same eigenvalue.) The definition of eigenvalue means that $D^{-1}Rv = \lambda v$, or $Rv = \lambda Dv$.

Since $r_{mm} = 0$, taking absolute values of the mth component of this vector equation implies

$$\begin{aligned}&|r_{m1}v_1 + r_{m2}v_2 + \cdots + r_{m,m-1}v_{m-1} + r_{m,m+1}v_{m+1} + \cdots + r_{mn}v_n| \\ &\quad = |\lambda d_{mm}v_m| = |\lambda||d_{mm}|.\end{aligned}$$

Since all $|v_i| \leq 1$, the left-hand side is at most $\sum_{j \neq m} |r_{mj}|$, which, according to the strict diagonal dominance hypothesis, is less than $|d_{mm}|$. This implies that $|\lambda||d_{mm}| < |d_{mm}|$, which in turn forces $|\lambda| < 1$. Since λ was an arbitrary eigenvalue, we have shown $\rho(D^{-1}R) < 1$, as desired. Now Theorem A.7 from Appendix A implies that Jacobi converges to a solution of $Ax = b$. Finally, since $Ax = b$ has a solution for arbitrary b, A is a nonsingular matrix.

Putting the Gauss–Seidel Method into the form of (2.43) yields

$$x_{k+1} = -(L + D)^{-1}Ux_k + (L + D)^{-1}b.$$

It then becomes clear that convergence of Gauss–Seidel follows if the spectral radius of the matrix

$$(L + D)^{-1}U \tag{2.44}$$

is less than one. The next theorem shows that strict diagonal dominance implies that this requirement is imposed on the eigenvalues.

THEOREM 2.11 If the $n \times n$ matrix A is strictly diagonally dominant, then (1) A is a nonsingular matrix, and (2) for every vector b and every starting guess, the Gauss–Seidel Method applied to $Ax = b$ converges to a solution. ■

Proof. Let λ be an eigenvalue of (2.44), with corresponding eigenvector v. Choose the eigenvector so that $v_m = 1$ and all other components are smaller in magnitude, as in the preceding proof. Note that the entries of L are the a_{ij} for $i > j$, and the entries of U are the a_{ij} for $i < j$. Then viewing row m of the eigenvalue equation of (2.44),

$$\lambda(D + L)v = Uv,$$

yields a string of inequalities similar to the previous proof:

$$\begin{aligned}|\lambda|\left(\sum_{i>m}|a_{mi}|\right) &< |\lambda|\left(|a_{mm}| - \sum_{i<m}|a_{mi}|\right) \\ &\leq |\lambda|\left(|a_{mm}| - \left|\sum_{i<m}a_{mi}v_i\right|\right) \\ &\leq |\lambda|\left|a_{mm} + \sum_{i<m}a_{mi}v_i\right| \\ &= \left|\sum_{i>m}a_{mi}v_i\right| \\ &\leq \sum_{i>m}|a_{mi}|.\end{aligned}$$

It follows that $|\lambda| < 1$, which finishes the proof. ❐

2.5.4 Sparse matrix computations

Direct methods based on Gaussian elimination provide the user a finite number of steps that terminate in the solution. What is the reason for pursuing iterative methods, which are only approximate and may require several steps for convergence?

There are two major reasons for using iterative methods like Gauss–Seidel. Both reasons stem from the fact that one step of an iterative method requires only a fraction of the floating point operations of a full LU factorization. As we established earlier in the chapter, Gaussian elimination for an $n \times n$ matrix costs on the order of n^3 operations. A single step of Jacobi's Method, for example, requires about n^2 multiplications (one for each matrix entry) and about the same number of additions. The question is how many steps will be needed for convergence within the user's tolerance.

One particular circumstance that argues for an iterative technique is when a good approximation to the solution is already known. For example, suppose that a solution to $Ax = b$ is known, after which A and/or b change by a small amount. We could imagine a dynamic problem where A and b are remeasured constantly as they change, and an accurate updated solution x is constantly required. If the solution to the previous problem is used as a starting guess for the new but similar problem, fast convergence of Jacobi or Gauss–Seidel can be expected.

Suppose the b in problem (2.42) is changed slightly from the original $b = [2.5, 1.5, 1, 1, 1.5, 2.5]$ to a new $b = [2.2, 1.6, 0.9, 1.3, 1.4, 2.45]$. We can check that the true solution of the system is changed from $[1, 1, 1, 1, 1, 1]$ to $[0.9, 1, 1, 1.1, 1, 1]$. Assume that we have in memory the sixth step of the Gauss–Seidel iteration x_6 from the preceding table, to use as a starting guess. Continuing Gauss–Seidel with the new b and with the helpful starting guess x_6 yields a good approximation in only one additional step. The next two steps are as follows:

x_7	x_8
0.8980	0.8994
0.9980	0.9889
0.9659	0.9927
1.0892	1.0966
0.9971	1.0005
0.9993	1.0003

This technique is often called **polishing**, because the method begins with an approximate solution, which could be the solution from a previous, related problem, and then merely refines the approximate solution to make it more accurate. Polishing is common in real-time applications where the same problem needs to be re-solved repeatedly with data that is updated as time passes. If the system is large and time is short, it may be impossible to run an entire Gaussian elimination or even a back substitution in the allotted time. If the solution hasn't changed too much, a few steps of a relatively cheap iterative method might keep sufficient accuracy as the solution moves through time.

The second major reason to use iterative methods is to solve sparse systems of equations. A coefficient matrix is called **sparse** if many of the matrix entries are known to be zero. Often, of the n^2 eligible entries in a sparse matrix, only $O(n)$ of them are nonzero. A **full** matrix is the opposite, where few entries may be assumed to be zero. Gaussian elimination applied to a sparse matrix usually causes **fill-in**, where the coefficient matrix changes from sparse to full due to the necessary row operations. For this reason, the efficiency of Gaussian elimination and its $PA = LU$ implementation become questionable for sparse matrices, leaving iterative methods as a feasible alternative.

Example 2.24 can be extended to a sparse matrix as follows:

► EXAMPLE 2.25 Use the Jacobi Method to solve the 100,000-equation version of Example 2.24.

Let n be an even integer, and consider the $n \times n$ matrix A with 3 on the main diagonal, -1 on the super- and subdiagonal, and $1/2$ in the $(i, n+1-i)$ position for all $i = 1, \ldots, n$, except for $i = n/2$ and $n/2+1$. For $n = 12$,

$$A = \begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix}. \quad (2.45)$$

Define the vector $b = (2.5, 1.5, \ldots, 1.5, 1.0, 1.0, 1.5, \ldots, 1.5, 2.5)$, where there are $n-4$ repetitions of 1.5 and 2 repetitions of 1.0. Note that if $n = 6$, A and b define the system of Example 2.24. The solution of the system for general n is $[1, \ldots, 1]$. No row of A has more than 4 nonzero entries. Since fewer than $4n$ of the n^2 potential entries are nonzero, we may call the matrix A sparse.

If we want to solve this system of equations for $n = 100,000$ or more, what are the options? Treating the coefficient matrix A as a full matrix means storing $n^2 = 10^{10}$ entries, each as a floating point double precision number requiring 8 bytes of storage. Note that 8×10^{10} bytes is approximately 80 gigabytes. Depending on your computational setup, it may be impossible to fit the entire n^2 entries into RAM.

Not only is size an enemy, but so is time. The number of operations required by Gaussian elimination will be on the order of $n^3 \approx 10^{15}$. If your machine runs on the order of a few GHz (10^9 cycles per second), an upper bound on the number of floating point operations per second is around 10^8. Therefore, $10^{15}/10^8 = 10^7$ is a reasonable guess at the number of seconds required for Gaussian elimination. There are 3×10^7 seconds in a year. Although this is back-of-the-envelope accounting, it is clear that Gaussian elimination for this problem is not an overnight computation.

On the other hand, one step of an iterative method will require approximately $2 \times 4n = 800,000$ operations, two for each nonzero matrix entry. We could do 100 steps of Jacobi iteration and still finish with fewer than 10^8 operations, which should take roughly a second or less on a modern PC. For the system just defined, with $n = 100,000$, the following Jacobi code `jacobi.m` needs only 50 steps to converge from a starting guess of $(0, \ldots, 0)$ to the solution $(1, \ldots, 1)$ within six correct decimal places. The 50 steps require less than 1 second on a typical PC.

```
% Program 2.1 Sparse matrix setup
% Input: n = size of system
% Outputs: sparse matrix a, r.h.s. b
function [a,b] = sparsesetup(n)
e = ones(n,1); n2=n/2;
a = spdiags([-e 3*e -e],-1:1,n,n);    % Entries of a
c=spdiags([e/2],0,n,n);c=fliplr(c);a=a+c;
a(n2+1,n2) = -1; a(n2,n2+1) = -1;     % Fix up 2 entries
b=zeros(n,1);                          % Entries of r.h.s. b
b(1)=2.5;b(n)=2.5;b(2:n-1)=1.5;b(n2:n2+1)=1;
```

```
% Program 2.2 Jacobi Method
% Inputs: full or sparse matrix a, r.h.s. b,
%         number of Jacobi iterations, k
% Output: solution x
function x = jacobi(a,b,k)
n=length(b);       % find n
d=diag(a);         % extract diagonal of a
r=a-diag(d);       % r is the remainder
x=zeros(n,1);      % initialize vector x
for j=1:k          % loop for Jacobi iteration
  x = (b-r*x)./d;
end                % End of Jacobi iteration loop
```

Note a few interesting aspects of the preceding code. The program `sparsesetup.m` uses MATLAB's `spdiags` command, which defines the matrix A as a sparse data structure. Essentially, this means that the matrix is represented by a set of triples (i, j, d), where d is the real number entry in position (i, j) of the matrix. Memory is not reserved for the entire n^2 potential entries, but only on an as-needed basis. The `spdiags` command takes the columns of a matrix and places them along the main diagonal, or a specified sub- or super-diagonal below or above the main diagonal.

MATLAB's matrix manipulation commands are designed to work seamlessly with the sparse matrix data structure. For example, an alternative to the preceding code would be to use MATLAB's `lu` command to solve the system directly. However, for that example, even though A is sparse, the upper-triangular matrix U that follows from Gaussian elimination suffers from fill-in during the process. For example, the upper-triangular U from Gaussian elimination for size $n = 12$ of the preceding matrix A is

$$\begin{bmatrix}
3 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 \\
0 & 2.7 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.165 \\
0 & 0 & 2.6 & -1.0 & 0 & 0 & 0 & 0 & 0 & 0.500 & 0.187 & 0.062 \\
0 & 0 & 0 & 2.6 & -1.000 & 0 & 0 & 0 & 0.500 & 0.191 & 0.071 & 0.024 \\
0 & 0 & 0 & 0 & 2.618 & -1.000 & 0 & 0.500 & 0.191 & 0.073 & 0.027 & 0.009 \\
0 & 0 & 0 & 0 & 0 & 2.618 & -1.000 & 0.191 & 0.073 & 0.028 & 0.010 & 0.004 \\
0 & 0 & 0 & 0 & 0 & 0 & 2.618 & -0.927 & 0.028 & 0.011 & 0.004 & 0.001 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.562 & -1.032 & -0.012 & -0.005 & -0.001 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.473 & -1.047 & -0.018 & -0.006 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.445 & -1.049 & -0.016 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.440 & -1.044 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.458
\end{bmatrix}.$$

Since U turns out to be a relatively full matrix, the memory restrictions previously mentioned again become a limitation. A significant fraction of the n^2 memory locations will be necessary to store U on the way to completing the solution process. It is more efficient, by several orders of magnitude in execution time and storage, to solve this large sparse system by an iterative method. ◀

2.5 Exercises

1. Compute the first two steps of the Jacobi and the Gauss–Seidel Methods with starting vector $[0, \ldots, 0]$.

(a) $\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$ (b) $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$

(c) $\begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 5 \end{bmatrix}$

2. Rearrange the equations to form a strictly diagonally dominant system. Apply two steps of the Jacobi and Gauss–Seidel Methods from starting vector $[0, \ldots, 0]$.

$$\text{(a)} \quad \begin{aligned} u + 3v &= -1 \\ 5u + 4v &= 6 \end{aligned} \qquad \text{(b)} \quad \begin{aligned} u - 8v - 2w &= 1 \\ u + v + 5w &= 4 \\ 3u - v + w &= -2 \end{aligned} \qquad \text{(c)} \quad \begin{aligned} u + 4v &= 5 \\ v + 2w &= 2 \\ 4u + 3w &= 0 \end{aligned}$$

3. Apply two steps of SOR to the systems in Exercise 1. Use starting vector $[0, \ldots, 0]$ and $\omega = 1.5$.

4. Apply two steps of SOR to the systems in Exercise 2 after rearranging. Use starting vector $[0, \ldots, 0]$ and $\omega = 1$ and 1.2.

5. Let λ be an eigenvalue of an $n \times n$ matrix A. (a) Prove the Gershgorin Circle Theorem: There is a diagonal entry A_{mm} such that $|A_{mm} - \lambda| \leq \sum_{j \neq m} |A_{mj}|$. (Hint: Begin with an eigenvector v such that $||v||_\infty = 1$, as in the proof of Theorem 2.10.) (b) Prove that a strictly diagonally dominant matrix cannot have a zero eigenvalue. This is an alternative proof of part (1) of Theorem 2.10.

2.5 Computer Problems

1. Use the Jacobi Method to solve the sparse system within six correct decimal places (forward error in the infinity norm) for $n = 100$ and $n = 100000$. The correct solution is $[1, \ldots, 1]$. Report the number of steps needed and the backward error. The system is

$$\begin{bmatrix} 3 & -1 & & & & \\ -1 & 3 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 3 & -1 & \\ & & & -1 & 3 & \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ \vdots \\ 1 \\ 2 \end{bmatrix}.$$

2. Use the Jacobi Method to solve the sparse system within three correct decimal places (forward error in the infinity norm) for $n = 100$. The correct solution is $[1, -1, 1, -1, \ldots, 1, -1]$. Report the number of steps needed and the backward error. The system is

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & 2 & 1 & \\ & & & 1 & 2 & \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix}.$$

3. Rewrite Program 2.2 to carry out Gauss–Seidel iteration. Solve the problem in Example 2.24 to check your work.

4. Rewrite Program 2.2 to carry out SOR. Use $\omega = 1.1$ to recheck Example 2.24.

5. Carry out the steps of Computer Problem 1 with $n = 100$ for (a) Gauss–Seidel Method and (b) SOR with $\omega = 1.2$.

6. Carry out the steps of Computer Problem 2 for (a) Gauss–Seidel Method and (b) SOR with $\omega = 1.5$.

7. Using your program from Computer Problem 3, decide how large a system of type (2.38) you can solve accurately by the Gauss–Seidel Method in one second of computation. Report the time required and forward error for various values of n.

2.6 METHODS FOR SYMMETRIC POSITIVE-DEFINITE MATRICES

Symmetric matrices hold a favored position in linear systems analysis because of their special structure, and because they have only about half as many independent entries as general matrices. That raises the question whether a factorization like the LU can be realized for half the computational complexity, and using only half the memory locations. For symmetric positive-definite matrices, this goal can be achieved with the Cholesky factorization.

Symmetric positive-definite matrices also allow a quite different approach to solving $Ax = b$, one that does not depend on a matrix factorization. This new approach, called the conjugate gradient method, is especially useful for large, sparse matrices, where it falls into the family of iterative methods.

To begin the section, we define the concept of positive-definiteness for symmetric matrices. Then we show that every symmetric positive-definite matrix A can be factored as $A = R^T R$ for an upper-triangular matrix R, the Cholesky factorization. As a result, the problem $Ax = b$ can be solved using two back substitutions, just as with the LU factorization in the nonsymmetric case. We close the section with the conjugate gradient algorithm and an introduction to preconditioning.

2.6.1 Symmetric positive-definite matrices

DEFINITION 2.12 The $n \times n$ matrix A is **symmetric** if $A^T = A$. The matrix A is **positive-definite** if $x^T Ax > 0$ for all vectors $x \neq 0$. ❐

▶ EXAMPLE 2.26 Show that the matrix $A = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$ is symmetric positive-definite.

Clearly A is symmetric. To show it is positive-definite, one applies the definition:

$$\begin{aligned} x^T Ax &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 2x_1^2 + 4x_1x_2 + 5x_2^2 \\ &= 2(x_1 + x_2)^2 + 3x_2^2 \end{aligned}$$

This expression is always non-negative, and cannot be zero unless both $x_2 = 0$ and $x_1 + x_2 = 0$, which together imply $x = 0$. ◀

▶ EXAMPLE 2.27 Show that the symmetric matrix $A = \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix}$ is not positive-definite.

Compute $x^T Ax$ by completing the square:

$$\begin{aligned} x^T Ax &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= 2x_1^2 + 8x_1x_2 + 5x_2^2 \\ &= 2(x_1^2 + 4x_1x_2) + 5x_2^2 \\ &= 2(x_1 + 2x_2)^2 - 8x_2^2 + 5x_2^2 \\ &= 2(x_1 + 2x_2)^2 - 3x_2^2 \end{aligned}$$