3 Iterative Methods

3.1 Jacobi and Gauss-Seidel Iteration

Direct Methods

In the previous chapter we discussed direct methods for solving linear systems. These are the methods of choice for small to moderate-sized problems. Computing the LU decomposition of an $n \times n$ matrix, for example, requires about $\frac{2}{3}n^3$ floating point operations. If $n = 100$, this is about 6.7×10^5 floating point operations, a task that takes well under a second to execute on a standard desktop computer. If $n = 1000$, however, this is about 6.7×10^8 flops, a task that takes considerably longer to finish (due to memory management issues in addition to the thousand-fold extra flops required). Since in practice matrices as large as tens of thousands by tens of thousands are common (in problems involving the numerical solution of partial differential equations such as in meteorology) and matrices as large as hundreds of thousands by hundreds of thousands are not uncommon (for example, applications involving the analysis of genetic data), more efficient methods are clearly needed.

We will have little to say about large dense matrices. If approximating such a matrix with a simpler matrix is not acceptable, then the computation will take a long time. Moving entries of the matrix from memory to the processor(s) will likely be more time-consuming than the actual computations.

Fortunately, in practice large matrices are commonly sparse. For large sparse matrices there are a number of iterative methods that—when used in combination with a smart storage system for the sparse matrix—can considerably reduce computation times. As a rule, direct methods are $O(n^3)$ and iterative methods are in the cases in which they are appropriate $O(n^2)$. The goal for computing with sparse matrices is always to get a method that is $O(N)$, where N is the number of nonzero entries in the matrix.

In this section we discuss two classical methods for the iterative solution of linear systems.

Matrix Splittings

Consider a square linear system $Ax = b$. We seek an iteration of the form $x^{k+1} = F(x^k)$, where an initial guess $x^0 \in \mathbb{R}^n$ is given and F is simple to compute. One way to find such an iteration is based on additively decomposing A into its upper triangle, lower triangle, and diagonal parts, called a **matrix splitting:**

$$A = U + L + D \tag{3.1}$$

where U is a square matrix with the strict upper triangle of A in its upper triangle and is zero otherwise, L is a square matrix with the strict lower triangle of A in its lower triangle and is zero otherwise, and D is a diagonal matrix containing the main diagonal of A. For example,

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 7 & 8 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$
$$= U + L + D$$

is a splitting of the form of Eq. (3.1). Now we can write

$$\begin{aligned} Ax &= b \\ (U + L + D)x &= b \\ Dx &= -(U + L)x + b \\ x &= -D^{-1}(U + L)x + D^{-1}b \end{aligned} \tag{3.2}$$

(if no diagonal element of A is zero, which we assume throughout this section). This equation has the form $x = g(x)$ of a fixed point iteration; perhaps

$$x^{k+1} = -D^{-1}(U + L)x^k + D^{-1}b \tag{3.3}$$

Jacobi Iteration

will be a useful iteration. Indeed, Eq. (3.3) is called **Jacobi iteration.** Note that D is diagonal so the computation of D^{-1} is trivial.

Example 3.1.1 Let's apply Jacobi iteration to the system $Ax = (3, -1, 4)^T$ with the matrix $A =$ [4 2 1; 1 3 1; 1 1 4]. From Eq. (3.2) we form $Dx = -(U + L)x + b$, which is equivalent to moving the nondiagonal entries to the RHS; that is,

$$\begin{aligned} 4x_1 + 2x_2 + x_3 &= 3 \\ x_1 + 3x_2 + x_3 &= -1 \\ x_1 + x_2 + 4x_3 &= 4 \end{aligned}$$

becomes

$$\begin{aligned} 4x_1 &= 3 - 2x_2 - x_3 \\ 3x_2 &= -1 - x_1 - x_3 \\ 4x_3 &= 4 - x_1 - x_2 \end{aligned}$$

corresponding to the matrix-vector form $Dx = -(U + L)x + b$. We obtain Eq. (3.3)

$$x_1^{k+1} = \frac{1}{4}\left(3 - 2x_2^k - x_3^k\right)$$
$$x_2^{k+1} = \frac{1}{3}\left(-1 - x_1^k - x_3^k\right)$$
$$x_3^{k+1} = \frac{1}{4}\left(4 - x_1^k - x_2^k\right)$$

by dividing out the diagonal entries and iterating. Let's take $x^0 = (0, 0, 0)^T$ as the initial guess. Then we have

$$\begin{aligned}
x_1^1 &= \frac{1}{4}(3 - 2 \cdot 0 - 0) \\
&= \frac{3}{4} \\
x_2^1 &= \frac{1}{3}(-1 - 0 - 0) \\
&= -\frac{1}{3} \\
x_3^1 &= \frac{1}{4}(4 - 0 - 0) \\
&= 1 \\
x_1^2 &= \frac{1}{4}\left(3 - 2 \cdot \frac{-1}{3} - 1\right) \\
&\doteq 0.6667 \\
x_2^2 &= \frac{1}{3}\left(-1 - \frac{3}{4} - 1\right) \\
&\doteq -0.9167 \\
x_3^2 &= \frac{1}{4}\left(4 - \frac{3}{4} - \frac{-1}{3}\right) \\
&\doteq 0.8958
\end{aligned}$$

which seems to be slowly converging to the true solution $x_1 = 1, x_2 = -1, x_3 = 1$. The method does not recognize that x_3^1 is exact, but notice that the absolute errors for these estimates are $\alpha_1 = \|x^1 - x\| \doteq .7120$ and $\alpha_2 = \|x^2 - x\| \doteq .3590$, so the method is certainly reducing the overall error. We would continue iterating until some convergence criterion was met. ■

The formula for Jacobi iteration may also be written in component form. If x_i^k is used to denote the ith component of x^k, then Jacobi iteration corresponds to solving equation

i for the diagonal entry x_i,

$$a_{ii}x_i = b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j$$

$$x_i = \frac{b_i}{a_{ii}} - \frac{1}{a_{ii}} \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j$$

and then iterating

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \frac{1}{a_{ii}} \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^k$$

until some convergence criterion is met. The convergence criterion used for Jacobi iteration may be that the absolute error, relative error, or residual error

$$r_k = \|b - Ax_k\|$$

be less than some tolerance. We will use the approximate relative error

$$\rho_k = \frac{\|x^k - x^{k-1}\|}{\|x^k\|}$$

for the convergence criterion. Often the tolerance τ is set as some small percentage of $\|x^0\|$ (but not smaller than some $\tau_\alpha > 0$; see Section 1.8).

We now ask the standard questions: Does Jacobi iteration (Eq. (3.3)) converge for all matrices and all initial guesses x^0? How rapidly does it converge? Is the method stable? And can we accelerate the convergence?

Analysis

Experience quickly shows that Jacobi iteration can diverge for some initial guesses for some matrices. In order to explore the convergence properties of this method, let's write Eq. (3.3) as

$$x^{k+1} = Mx^k + \beta,$$

where the matrix $M = -D^{-1}(U + L)$ and vector $\beta = D^{-1}b$ are known. Any iteration that can be written in the form $x^{k+1} = Mx^k + \beta$ is said to be a **stationary method** (since M and β do not depend on k). Note that

$$\begin{aligned} \|x^{k+1}\| &= \|Mx^k + \beta\| \\ &\leq \|M\|\|x^k\| + \|\beta\| \end{aligned}$$

for the natural norm induced by the vector norm (see Section 2.5). So

$$\begin{aligned} \left\|x^{k+1}\right\| &\leq \|M\| \left\|x^k\right\| + \|\beta\| \\ &\leq \|M\| \left(\|M\| \left\|x^{k-1}\right\| + \|\beta\|\right) + \|\beta\| \\ &= \|M\|^2 \left\|x^{k-1}\right\| + \|M\| \|\beta\| + \|\beta\| \end{aligned}$$

$$
\begin{aligned}
&= \|M\|^2 \left\|x^{k-1}\right\| + (\|M\| + 1)\|\beta\| \\
&\leq \|M\|^2 (\|M\| \left\|x^{k-2}\right\| + \|\beta\|) + (\|M\| + 1)\|\beta\| \\
&= \|M\|^3 \left\|x^{k-2}\right\| + (\|M\|^2 + \|M\| + 1)\|\beta\| \\
&\vdots \\
&\leq \|M\|^{k+1} \left\|x^0\right\| + (\|M\|^k + \cdots + \|M\| + 1)\|\beta\|
\end{aligned}
$$

and so if $\|M\| < 1$, then

$$\left\|x^{k+1}\right\| \leq \|M\|^{k+1} \left\|x^0\right\| + \frac{1}{1 - \|M\|}\|\beta\| \tag{3.4}$$

since $\|M\|^k + \cdots + \|M\| + 1$ is a partial sum of the geometric series

$$\sum_{k=0}^{\infty} \|M\|^k = \frac{1}{1 - \|M\|}. \tag{3.5}$$

Because the term $\|M\|^{k+1} \to 0$ as $k \to \infty$ if $\|M\| < 1$, we suspect from Eq. (3.4) that $\|M\| < 1$ will give convergence. While this is true, there is a more precise result. Recall from Section 2.5 that the spectral radius of a matrix M is

$$\rho(M) = \max_{i=1}^{n}(|\lambda_i|),$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of M. It is a fact that if $\rho(M) < 1$ then M is **convergent:** that is, $M^k \to 0$ (the zero matrix) as $k \to \infty$, and also

$$\sum_{k=0}^{\infty} M^k = (I - M)^{-1}$$

(an analogue of Eq. (3.5)). The quantity $(I - M)^{-1}$ is called the **resolvent** of M; note that M itself need not be invertible for the resolvent to exist. Now, consider again the linear fixed point iteration

$$
\begin{aligned}
x^{k+1} &= Mx^k + \beta \\
&= M(Mx^{k-1} + \beta) + \beta \\
&= M^2x^{k-1} + (M + I)\beta \\
&= M^2(Mx^{k-2} + \beta) + (M + I)\beta \\
&= M^3x^{k-2} + (M^2 + M + I)\beta \\
&\vdots \\
&= M^{k+1}x^0 + (M^k + \cdots + M + I)\beta.
\end{aligned}
$$

So, if M is convergent, we have

$$\begin{aligned}\lim_{k\to\infty} x^{k+1} &= \lim_{k\to\infty} (M^{k+1}x^0 + (M^k + \cdots + M + I)\beta) \\ &= 0 + (I - M)^{-1}\beta \\ &= (I - M)^{-1}\beta\end{aligned}$$

for any initial guess x^0. In fact, the iteration $x^{k+1} = Mx^k + \beta$ converges to this unique value for every initial guess if and only if M is convergent. (If $\rho(M) \geq 1$, the method can still converge for some x^0, in principle, but not for every x^0. For such an M the method is unstable however.) Indeed, $x = (I - M)^{-1}\beta$ is a fixed point of the method, for

$$\begin{aligned}Mx + \beta &= M(I - M)^{-1}\beta + \beta \\ &= M\sum_{k=0}^{\infty} M^k\beta + \beta \\ &= \sum_{k=0}^{\infty} M^{k+1}\beta + \beta \\ &= \sum_{k=1}^{\infty} M^k\beta + I\beta \\ &= \sum_{k=0}^{\infty} M^k\beta \\ &= (I - M)^{-1}\beta \\ &= x\end{aligned}$$

(as $M^0 = I$ by convention). In summary, if M is convergent, then $x = Mx + \beta$ has a unique fixed point at $x = (I - M)^{-1}\beta$, and fixed point iteration always converges to this value.

Note the similarlity to the fixed point analysis we performed on Newton's method in Section 1.4. Here the requirement that $\rho(M) < 1$ replaces the requirement that $|g'(x)| < 1$. Indeed, M is the derivative of the iteration function $Mx + \beta$ with respect to x.

Hence, Jacobi iteration, or any other matrix splitting technique, will converge if the matrix $A = U + L + D$ leads to an M that is a convergent matrix. Since the iteration will converge for *any* initial guess, stability is immediate (a perturbation of a given x^k will necessarily perturb it to another value from which the method converges). The speed of convergence can be shown to be like $\rho(M)^k$, that is,

$$\frac{\|x^k - x\|}{\|x^0 - x\|} = O(\rho(M)^k)$$

as $k \to \infty$. (This is linear convergence.) Hence a small spectral radius is desirable, and if $\rho(M)$ is near unity, then we must expect very slow convergence.

Diagonal Dominance

Clearly, A is not always such that $M = -D^{-1}(U + L)$ is convergent. However, there is an important class of matrices for which this is always the case. We say that a square matrix A is **diagonally dominant** if

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

$(i = 1, \ldots, n)$, that is, if the diagonal element of each row exceeds in absolute value the sum of the absolute values of all other entries in that row. If the inequality is strict we say that the matrix is **strictly diagonally dominant,** and if it is weak we say that the matrix is **weakly diagonally dominant.** If A is strictly diagonally dominant, then $M = -D^{-1}(U + L)$ is convergent and Jacobi iteration will converge. In fact, the method will frequently converge if A is weakly diagonally dominant. Diagonally dominant matrices occur frequently in applications involving the numerical solution of partial differential equations.

If A is not diagonally dominant, then in principle we must check $\rho(M)$ to see if the method is applicable. This is usually too expensive, however, and so we use the fact that

$$\rho(M) \leq \|M\|$$

for any natural norm. If $\|M\| < 1$ in some natural norm, then M is convergent and we may use Jacobi iteration.

Example 3.1.2 The matrix $A = [1\ 1\ 3; 1\ 3\ 1; 3\ 1\ 1]$ is not strictly diagonally dominant, but interchanging the first and last rows gives $A_1 = [3\ 1\ 1; 1\ 3\ 1; 1\ 1\ 3]$ which is strictly diagonally dominant. Jacobi iteration will converge if applied to A_1.

The matrix $M = [0.3\ 0.2\ 0.1; 0.2\ 0.2\ -0.2; 0.4\ -0.5\ 0]$ has Frobenius norm (see Section 2.5) $\|M\| = \sqrt{.3^2 + .2^2 + \cdots + (-.5)^2 + 0^2} \doteq .8185$ so $\rho(M) < 1$ (in fact, $\rho(M) \doteq .4531$). Hence, iteration of $x^{k+1} = Mx^k + c$ will converge for any c and any x^0. ■

The easiest norm to use in checking whether $\|M\| < 1$ is the matrix norm that is induced by the l_∞ vector norm (see Section 2.5),

$$\|M\|_\rho = \max_{i=1}^{n} \left\{ \sum_{j=1}^{n} |m_{ij}| \right\}$$

that is, the maximum row sum of M. This matrix norm is called the **max norm** or **inf norm.** The spectral norm $\|M\|_s$ will be closer to $\rho(M)$ but is harder to compute. For the matrix M in Example 3.1.2, the row sums are 0.6, 0.6, and 0.9, so $\|M\| = .9$. This is larger than the Frobenius norm $\|A\|_F \doteq .8185$ and the spectral norm $\|A\|_s \doteq .6419$, and all exceed the spectral radius $\rho(M) \doteq .4531$, as they must. Basing our estimate only on the max norm, we estimate that the error decreases to about 90% of its previous level each iteration; basing it on the Frobenius norm, about 82%; on the spectral norm, about 64%. The spectral radius gives the correct value, about 45%. The spectral radius may be estimated if needed.

If $\rho(M)$ is near unity, convergence will be slow. Can we speed it up? One possibility arises from considering again what we did in Example 3.1.1.

Example 3.1.3 Consider again the system $Ax = (3, -1, 4)^T$ with the matrix $A = [4\ 2\ 1; 1\ 3\ 1; 1\ 1\ 4]$. We have from Eq. (3.3)

$$x_1^{k+1} = \frac{1}{4}\left(3 - 2x_2^k - x_3^k\right)$$
$$x_2^{k+1} = \frac{1}{3}\left(-1 - x_1^k - x_3^k\right)$$
$$x_3^{k+1} = \frac{1}{4}\left(4 - x_1^k - x_2^k\right),$$

and we again take $x^0 = (0, 0, 0)^T$ as the initial guess. Then we have

$$x_1^1 = \frac{1}{4}(3 - 2 \cdot 0 - 0)$$
$$= \frac{3}{4}$$
$$x_2^1 = \frac{1}{3}\left(-1 - x_1^0 - x_3^0\right).$$

It may now occur to us that we have presumably improved the estimate of the true value of x_1, namely x_1^1, and we may wish to use this better estimate in place of x_1^0. Let's do that:

$$x_2^1 = \frac{1}{3}\left(-1 - x_1^1 - x_3^0\right)$$
$$= \frac{1}{3}\left(-1 - \frac{3}{4} - 0\right)$$
$$= -\frac{7}{12},$$

which is a better estimate of $x_2 = -1$ than the previous $x_2^1 = -1/3$. Let's use the same trick for x_3^1:

$$x_3^1 = \frac{1}{4}\left(4 - x_1^1 - x_2^1\right)$$
$$\frac{1}{4}\left(4 - \frac{3}{4} - \frac{-7}{12}\right)$$
$$= \frac{23}{24}$$
$$\doteq 0.9583$$

which is close to the previous estimate of $3/4$. Continuing in this way, using new estimates

as soon as they become available, we have

$$\begin{aligned} x_1^2 &= \frac{1}{4}\left(3 - 2 \cdot \frac{-7}{12} - \frac{23}{24}\right) \\ &\doteq 0.8021 \\ x_2^2 &= \frac{1}{3}\left(-1 - 0.8021 - \frac{23}{24}\right) \\ &\doteq -0.9201 \\ x_3^2 &= \frac{1}{4}(4 - 0.8021 - -0.9201) \\ &\doteq 1.0295, \end{aligned}$$

which is superior to the estimate $x^2 = (0.7292, -0.8333, 0.6458)^T$ from Jacobi iteration in every component (recall that the solution is $x = (1, -1, 1)^T$). ■

Gauss-Seidel Iteration

The method of Example 3.1.3 is referred to as **Gauss-Seidel iteration.** It simply uses newer information as soon as it becomes available (a simple and widely applicable idea). Gauss-Seidel iteration corresponds to the matrix splitting

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

(i.e., $M = -(L + D)^{-1}U$ if $(L + D)^{-1}$ exists), giving the iteration

$$x^{k+1} = -(L + D)^{-1}Ux^k + (L + D)^{-1}b. \tag{3.6}$$

Of course, we would not form $(L + D)^{-1}$ explicitly but would instead proceed as in Example 3.1.3. In component form,

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \frac{1}{a_{ii}} \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \frac{1}{a_{ii}} \sum_{j=i+1}^{n} a_{ij} x_j^k,$$

where by convention an empty sum is zero. Again, we assume throughout that a_{ii} is nonzero ($i = 1, \ldots, n$).

Comparison

Typically, if Jacobi iteration converges for a given matrix A, then Gauss-Seidel iteration also converges and is faster (that is, the spectral radius of its iteration matrix M is smaller). However, this is not always the case. If A is strictly diagonally dominant, then Gauss-Seidel iteration will converge for any initial guess. There are block forms of both algorithms.

Gauss-Seidel iteration is generally superior to Jacobi iteration. One exception might be on a parallel machine: If $x \in \mathbb{R}^n$ and there are n processors available, Jacobi iteration can be performed very efficiently (processor i computes the updated x_i), whereas Gauss-Seidel does not gain much benefit. However, the number p of processors is usually much less than n, so the acceleration is not as great as indicated. In that case we might have

each processor compute n/p of the entries of x using the idea of Gauss-Seidel iteration (that is, making use of any updated variables available to it) but might or might not try to pass updated values between processors. There are a number of strategies in common use for performing Gauss-Seidel iteration on a parallel machine.

Note that the particular iterates in Jacobi or Gauss-Seidel iteration depend on the ordering of the unknowns. If we write the same linear system in a different order we generate a different Jacobi or Gauss-Seidel sequence.

For many years, these methods were used to solve large linear systems arising from the discretization of partial differential equations, among other applications. Today, however, the Jacobi and Gauss-Seidel iterations are used most often in conjunction with another iterative method, such as the technique that we discuss in Section 3.3.

PROBLEMS 3.1

1. a. Perform another three iterations of the method in Example 3.1.1. Compute the relative error of your answer.

b. Perform another three iterations of the method in Example 3.1.3. Compute the relative error of your answer.

2. a. Find the spectral radius of the Jacobi iteration matrix for Example 3.1.1, and use it to estimate the size of the error after 5 iterations. Compare this to the estimate found by using the max, Frobenius, and spectral norms.

b. Find the spectral radius of the Gauss-Seidel iteration matrix for Example 3.1.3, and use it to estimate the size of the error after 5 iterations. Compare this to the estimate found by using the max, Frobenius, and spectral norms.

c. Compare your estimates with the results from Problems 1(a) and (b).

3. a. Prove that Jacobi iteration must converge for a strictly diagonally dominant matrix. (*Hint:* Show that $\|M\| < 1$ for some convenient norm.)

b. Prove that Gauss-Seidel iteration must converge for a strictly diagonally dominant matrix.

4. a. Verify that Eq. (3.6) is equivalent to the scalar form of Gauss-Seidel iteration as described in Example 3.1.3.

b. Jacobi iteration is also known as the **method of simultaneous displacements,** while Gauss-Seidel iteration is also known as the **method of successive displacements.** Why are these descriptions appropriate?

5. a. Prove that $\rho(A) \leq \|A\|$ for any natural matrix norm.

b. Show that if $\rho(M) < 1$, then the resolvent $(I - M)^{-1}$ of M exists.

c. Show that the Gauss-Seidel matrix $-(L + D)^{-1}U$ must be singular. Is this also true of the Jacobi iteration matrix?

MATLAB 3.1

In principle we could implement Jacobi iteration using Eq. (3.3), $x^{k+1} = -D^{-1}(U + L)x^k + D^{-1}b$, and Gauss-Seidel iteration using Eq. (3.6), $x^{k+1} = -(L + D)^{-1}Ux^k + (L + D)^{-1}b$, but because n is generally large in these applications solving even the triangular system $(L + D)x^{k+1} = -Ux^k + b$ for Gauss-Seidel iteration may be too time consuming. In fact, it is often the case that we avoid ever forming A, L, U, or D explicitly, owing to memory concerns and the fact that A is typically sparse. Instead we write a program that returns selected entries of A or that computes Ax (given x). We then implement the updates as separate formulas for $x_1^{k+1}, \ldots, x_n^{k+1}$.

For convenience, however, let us use the matrix forms for these iterations here. Enter:

```
» A=4*eye(10);for i=2:9;A(i,[i-1 i+1])=[1 1];end
» A(10,1)=1;A(1,10)=1;A(1,2)=1;A(10,9)=1
```