

6.2 Iterative Methods

New solution methods are needed when a problem $Ax = b$ is too large and expensive for ordinary elimination. We are thinking of *sparse matrices* A, so that multiplications Ax are relatively cheap. If A has at most p nonzeros in every row, then ***Ax needs at most pn multiplications***. Typical applications are to large finite difference or finite element equations, where we often write $A = K$.

We are turning from elimination to look at **iterative methods**. There are really two big decisions, the preconditioner P and the choice of the method itself:

1. A good preconditioner P is close to A but much simpler to work with.

2. Options include ***pure iterations*** (6.2), ***multigrid*** (6.3), and ***Krylov methods*** (6.4), including the conjugate gradient method.

Pure iterations compute each new x_{k+1} from $x_k - P^{-1}(Ax_k - b)$. This is called "*stationary*" because every step is the same. Convergence to $x_\infty = A^{-1}b$, studied below, will be fast when all eigenvalues of $M = I - P^{-1}A$ are small. It is easy to suggest a preconditioner, but not so easy to suggest an excellent P (*Incomplete LU is a success*). The older iterations of Jacobi and Gauss-Seidel are less favored (but they are still important, you will see good points and bad points).

Multigrid begins with Jacobi or Gauss-Seidel iterations, for the one job that they do well. They remove high frequency components (rapidly oscillating parts) to leave a smooth error. Then the central idea is to *move to a coarser grid*—where the rest of the error can be destroyed. Multigrid is often dramatically successful.

Krylov spaces contain all combinations of $b, Ab, A^2b, \ldots$ and Krylov methods look for the best combination. Combined with preconditioning, the result is terrific. When the growing subspaces reach the whole space $\mathbf{R}^n$, those methods give the exact solution $A^{-1}b$. But in reality we stop much earlier, long before n steps are complete. The *conjugate gradient method* (*for positive definite A, and with a good preconditioner*) has become truly important.

The goal of numerical linear algebra is clear: **Find a fast stable algorithm that uses the special properties of the matrix.** We meet matrices that are symmetric or triangular or orthogonal or tridiagonal or Hessenberg or Givens or Householder. Those are at the core of matrix computations. The algorithm doesn't need details of the entries (which come from the specific application). By concentrating on the matrix structure, numerical linear algebra offers major help.

Overall, elimination with good numbering is the first choice! But storage and CPU time can become excessive, especially in three dimensions. At that point we turn from elimination to iterative methods, which require more expertise than $K\backslash F$. The next pages aim to help the reader at this frontier of scientific computing.

Stationary Iterations

We begin with old-style pure stationary iteration. The letter K will be reserved for "Krylov" so we leave behind the notation $KU = F$. The linear system becomes $Ax = b$. The large sparse matrix A is not necessarily symmetric or positive definite:

Linear system $Ax = b$ **Residual** $r_k = b - Ax_k$ **Preconditioner** $P \approx A$

The preconditioner P attempts to be close to A and still allow fast iterations. The Jacobi choice $P =$ diagonal of A is one extreme (fast but not very close). The other extreme is $P = A$ (too close). Splitting the matrix A gives a new form of $Ax = b$:

Splitting $$\boldsymbol{Px = (P - A)x + b}. \tag{1}$$

This form suggests an iteration, in which every vector x_k leads to the next x_{k+1}:

Iteration $$\boldsymbol{Px_{k+1} = (P - A)x_k + b}. \tag{2}$$

Starting from any x_0, the first step finds x_1 from $Px_1 = (P - A)x_0 + b$. The iteration continues to x_2 with the same matrix P, so it often helps to know its triangular factors in $P = LU$. Sometimes P itself is triangular, or L and U are approximations to the triangular factors of A. Two conditions on P make the iteration successful:

1. The new x_{k+1} must be quickly computable. Equation (2) must be fast to solve.

2. The errors $e_k = x - x_k$ should approach zero as rapidly as possible.

Subtract equation (2) from (1) to find the **error equation**. It connects e_k to e_{k+1}:

Error $Pe_{k+1} = (P - A)e_k$ which means $$\boldsymbol{e_{k+1} = (I - P^{-1}A)e_k = Me_k}. \tag{3}$$

The right side b disappears in this error equation. Each step multiplies the error vector e_k by M. The speed of convergence of x_k to x (and of e_k to zero) depends entirely on M. *The test for convergence is given by the eigenvalues of* M:

Convergence test Every eigenvalue of $M = I - P^{-1}A$ must have $|\lambda(M)| < 1$.

The largest eigenvalue (in absolute value) is the **spectral radius** $\boldsymbol{\rho(M)} = \max|\lambda(M)|$. Convergence requires $\rho(M) < 1$. The **convergence rate** is set by the largest eigenvalue. For a large problem, we are happy with $\rho(M) = .9$ and even $\rho(M) = .99$.

When the initial error e_0 happens to be an eigenvector of M, the next error is $e_1 = Me_0 = \lambda e_0$. At every step the error is multiplied by λ. *So we must have* $|\lambda| < 1$. Normally e_0 is a combination of all the eigenvectors. When the iteration multiplies by M, each eigenvector is multiplied by its own eigenvalue. After k steps those multipliers are λ^k, and the largest is $(\rho(M))^k$.

If we don't use a preconditioner then $M = I - A$. All the eigenvalues of A must be inside a unit circle centered at 1, for convergence. Our second difference matrices $A = K$ would fail this test ($I - K$ is too large). The first job of a preconditioner is to get the matrix decently scaled. Jacobi will now give $\rho(I - \frac{1}{2}K) < 1$, and a really good P will do more.

Jacobi Iterations

For preconditioner we first propose a simple choice:

Jacobi iteration $\boldsymbol{P}$ **= diagonal part** $\boldsymbol{D}$ **of** $\boldsymbol{A}$

Typical examples have spectral radius $\rho(M) = 1 - cN^{-2}$, where N counts meshpoints in the longest direction. This comes closer and closer to 1 (too close) as the mesh is refined and N increases. But Jacobi is important, it does part of the job.

For our tridiagonal matrices K, Jacobi's preconditioner is just $P = 2I$ (the diagonal of K). **The Jacobi iteration matrix becomes** $\boldsymbol{M = I - D^{-1}A = I - \frac{1}{2}K}$:

Iteration matrix for a Jacobi step

$$M = I - \frac{1}{2}K = \frac{1}{2}\begin{bmatrix} 0 & 1 & & \\ 1 & 0 & 1 & \\ & 1 & 0 & 1 \\ & & 1 & 0 \end{bmatrix}. \tag{4}$$

Here is x^{new} from x^{old}, in detail. You see how Jacobi shifts the off-diagonal entries of A to the right-hand side, and divides by the diagonal part $D = 2I$:

$$\begin{aligned} 2x_1 - x_2 &= b_1 \\ -x_1 + 2x_2 - x_3 &= b_2 \\ -x_2 + 2x_3 - x_4 &= b_3 \\ -x_3 + 2x_4 &= b_4 \end{aligned} \quad \text{becomes} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^{\text{new}} = \frac{1}{2}\begin{bmatrix} & x_2 & \\ x_1 & + & x_3 \\ x_2 & + & x_4 \\ x_3 & & \end{bmatrix}^{\text{old}} + \frac{1}{2}\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}. \tag{5}$$

The equation is solved when $x^{\text{new}} = x^{\text{old}}$, but this only happens in the limit. The real question is the number of iterations to get close to convergence, and that depends on the eigenvalues of M. How close is $\lambda_{\max}$ to 1?

Those eigenvalues are simple cosines. In Section 1.5 we actually computed all the eigenvalues $\lambda(K) = 2 - 2\cos(\frac{j\pi}{N+1})$. Since $M = I - \frac{1}{2}K$, we now divide those eigenvalues by 2 and subtract from 1. The eigenvalues $\cos j\theta$ of M are less than 1!

Jacobi eigenvalues $\lambda_j(M) = 1 - \dfrac{2 - 2\cos j\theta}{2} = \boldsymbol{\cos j\theta}$ with $\theta = \dfrac{\pi}{N+1}$. (6)

Convergence is safe (but slow) because $|\cos\theta| < 1$. Small angles have $\cos\theta \approx 1 - \frac{1}{2}\theta^2$. The choice $j = 1$ gives us the first (and largest) eigenvalue of M:

Spectral radius $$\lambda_{\max}(M) = \cos\theta \approx 1 - \frac{1}{2}\left(\frac{\pi}{N+1}\right)^2. \tag{7}$$

Convergence is slow for the lowest frequency. The matrix M in (4) has the four eigenvalues $\cos\frac{\pi}{5}, \cos\frac{2\pi}{5}, \cos\frac{3\pi}{5}$, and $\cos\frac{4\pi}{5}$ (which is $-\cos\frac{\pi}{5}$) in Figure 6.8.

There is an important point about those Jacobi eigenvalues $\lambda_j(M) = \cos j\theta$. The magnitude $|\lambda_j|$ at $j = N$ is the same as the magnitude at $j = 1$. This is not good for multigrid, where high frequencies need to be strongly damped. So **weighted Jacobi** has a valuable place, with a weighting factor ω in $M = I - \omega D^{-1}A$:

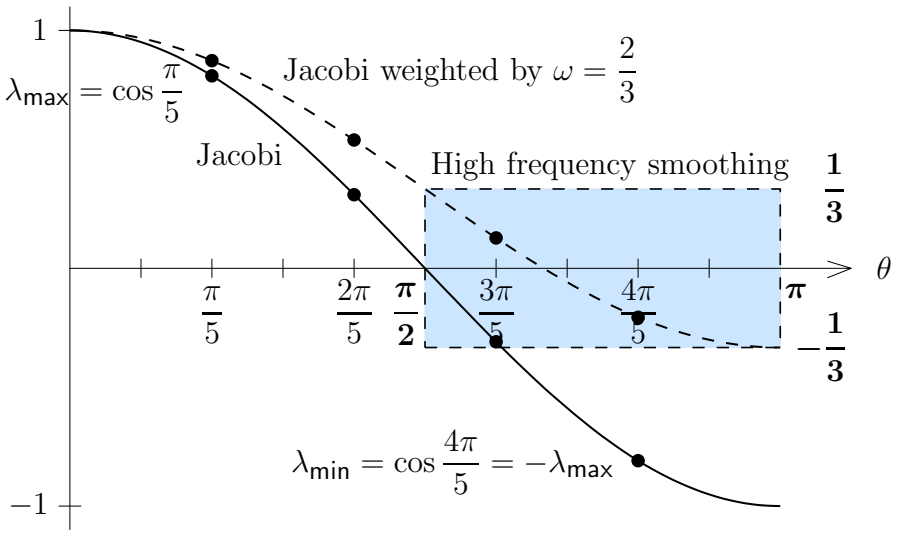

Figure 6.8: The eigenvalues of Jacobi's $M = I - \frac{1}{2}K$ are $\cos j\theta$, starting near $\lambda = 1$ and ending near $\lambda = -1$. Weighted Jacobi has $\lambda = 1 - \omega + \omega\cos j\theta$, ending near $\lambda = 1 - 2\omega$. Both graphs show $j = 1, 2, 3, 4$ and $\theta = \frac{\pi}{N+1} = \frac{\pi}{5}$ (with $\omega = \frac{2}{3}$).

Jacobi's iteration matrix $M = I - D^{-1}A$ changes to $M = I - \omega D^{-1}A$.

The preconditioner is now $P = D/\omega$. Here are the eigenvalues $\lambda(M)$ when $A = K$:

Weighted Jacobi

$$D = 2I \quad \text{and} \quad M = I - \frac{\omega}{2}A \quad \text{and} \quad \omega < 1$$
$$\lambda_j(M) = 1 - \frac{\omega}{2}(2 - 2\cos j\theta) = \mathbf{1 - \omega + \omega\cos j\theta}. \tag{8}$$

The dashed-line graph in Figure 6.8 shows these values $\lambda_j(M)$ for $\omega = \frac{2}{3}$. This ω is optimal in damping the high frequencies ($j\theta$ between $\pi/2$ and π) by at least $\frac{1}{3}$:

$$\text{At } j\theta = \frac{\pi}{2} \qquad \lambda(M) = 1 - \omega + \omega\cos\frac{\pi}{2} = 1 - \frac{2}{3} = \mathbf{\frac{1}{3}}$$

$$\text{At } j\theta = \pi \qquad \lambda(M) = 1 - \omega + \omega\cos\pi = 1 - \frac{4}{3} = \mathbf{-\frac{1}{3}}$$

If we move away from $\omega = \frac{2}{3}$, one of those eigenvalues will increase in magnitude. A weighted Jacobi iteration will be a good smoother within multigrid.

In two dimensions the picture is essentially the same. **The N^2 eigenvalues of $K2D$ are the sums $\lambda_j + \lambda_k$ of the N eigenvalues of K.** All eigenvectors are samples of $\sin j\pi x \sin k\pi y$. (In general, the eigenvalues of $\texttt{kron}(A, B)$ are $\lambda_j(A)\lambda_k(B)$. For $\texttt{kron}(K, I) + \texttt{kron}(I, K)$, sharing eigenvectors means we can add eigenvalues.)

Jacobi has $P = 4I$, from the diagonal of $K2D$. So $M2D = I2D - \frac{1}{4}K2D$:

$$\lambda_{jk}(M2D) = 1 - \frac{1}{4}\left[\lambda_j(K) + \lambda_k(K)\right] = \frac{1}{2}\cos j\theta + \frac{1}{2}\cos k\theta\,. \tag{9}$$

With $j = k = 1$, the spectral radius $\lambda_{\max}(M) = \cos\theta$ is the same $1 - cN^{-2}$ as in 1D.

Numerical Experiments

The multigrid method grew out of the slow convergence of Jacobi iterations. You have to see how typical error vectors e_k begin to decrease and then stall. For weighted Jacobi, the high frequencies disappear long before the low frequencies. Figure 6.___ shows a drop between e_0 and e_{--}, and then very slow decay toward $e_\infty = 0$. We have chosen the second difference matrix $A = K_{500}$ and started the iterations with the right-hand side $x_0 = b = \texttt{rand}(500, 1)$.

This unacceptably slow convergence is hidden if we only look at the residual $r_k = b - Ax_k$. Instead of measuring the error $x - x_k$ in the solution, r_k measures the error in the equation. That residual error *does* fall quickly. The key that led to multigrid is the rapid drop in r with such a slow drop in e.

Figure 6.9: The residual $b - Ax$ falls quickly but the solution error gets stalled.

Gauss-Seidel and the Red-Black Ordering

The Gauss-Seidel idea is to use the components of x^{new} as soon as they are computed. This cuts the storage requirement in half, since x^{old} is overwritten by x^{new}. The preconditioner $P = D + L$ becomes *triangular* instead of diagonal (still easy to use):

Gauss-Seidel iteration $\quad \boldsymbol{P}$ **= lower triangular part of** $\boldsymbol{A}$

Gauss-Seidel gives faster error reduction than ordinary Jacobi, because the Jacobi eigenvalues $\cos j\theta$ become $(\cos j\theta)^2$. The spectral radius is squared, so one Gauss-Seidel step is worth two Jacobi steps. The (large) number of iterations is cut in half, when the -1's below the diagonal stay with the 2's on the *left side* of Px^{new}:

Gauss-Seidel $\boldsymbol{Px^{\text{new}} = (P-A)x^{\text{old}} + b}$

$$-1\begin{bmatrix}0\\x_1\\x_2\\x_3\end{bmatrix}^{\text{new}} + 2\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}^{\text{new}} = \begin{bmatrix}x_2\\x_3\\x_4\\0\end{bmatrix}^{\text{old}} + \begin{bmatrix}b_1\\b_2\\b_3\\b_4\end{bmatrix}. \quad (10)$$

A new x_1 comes from the first equation, because P is triangular. Using x_1^{new} in the second equation gives x_2^{new}, which enters the third equation. Problem 1 shows that $x^{\text{new}} = (\cos j\theta)^2 x^{\text{old}}$, with the correct eigenvector x^{old}. All the Jacobi eigenvalues $\cos j\theta$ are squared for Gauss-Seidel, so they become smaller.

Symmetric Gauss-Seidel comes from a double sweep, reversing the order of components to make the combined process symmetric. By itself, $I - P^{-1}A$ is not symmetric for triangular P.

A **red-black ordering** produces a neat compromise between Jacobi and Gauss-Seidel. Imagine that a two-dimensional grid is a *checkerboard*. Number the red nodes before the black nodes. The numbering will not change Jacobi's method (which keeps

all of x^{old} to create x^{new}). But Gauss-Seidel will be improved. In one dimension, Gauss-Seidel updates all the even (red) components x_{2j} using known black values. Then it updates the odd (black) components x_{2j+1} using the new red values:

$$x_{2j} \longleftarrow \frac{1}{2}\left(x_{2j-1} + x_{2j+1} + b_{2j}\right) \quad \text{and then} \quad x_{2j+1} \longleftarrow \frac{1}{2}\left(x_{2j} + x_{2j+2} + b_{2j+1}\right). \tag{11}$$

In two dimensions, $x_{i,j}$ is red when $i+j$ is even, and black when $i+j$ is odd. Laplace's five-point difference matrix uses four black values to update each center value (red). Then red values update black, giving one example of *block Gauss-Seidel*.

For **line Gauss-Seidel**, each row of grid values forms a block. The preconditioner P is *block triangular*. That is the right way to see P in 2D:

$$P_{\text{red}-\text{black}} = \begin{bmatrix} 4I & 0 \\ -1\text{'s} & 4I \end{bmatrix} \quad \text{and} \quad P_{\text{line G}-\text{S}} = \begin{bmatrix} K+2I & 0 & 0 \\ -I & K+2I & 0 \\ 0 & -I & K+2I \end{bmatrix}$$

A great feature is the option to compute all red values *in parallel*. They need only black values and can be updated in any order—there are no connections among red values (or within black values in the second half-step). For line Jacobi, the rows in 2D can be updated in parallel (and the plane blocks in 3D). Block matrix computations are efficient.

Overrelaxation (SOR) is a combination of Jacobi and Gauss-Seidel, using a factor ω that almost reaches 2. The preconditioner is $P = D + \omega L$. (By *overcorrecting* from x_k to x_{k+1}, hand calculators noticed that they could finish in a few weeks.) My earlier book and many other references show how ω is chosen to minimize the spectral radius $\rho(M)$, improving $\rho = 1 - cN^{-2}$ to $\rho(M) = 1 - cN^{-1}$. Then convergence is much faster (N steps instead of N^2, to reduce the error by a constant factor like e).

Incomplete *LU*

A different approach has given much more flexibility in constructing a good P. The idea is to compute an ***incomplete LU factorization*** of the true matrix A:

Incomplete *LU* $\quad \boldsymbol{P = (approximation\ to\ L)(approximation\ to\ U)}$ (12)

The exact $A = LU$ has fill-in. So does Cholesky's $A = R^{\mathrm{T}}R$. Zero entries in A become nonzero in L and U and R. But $P = L_{\text{approx}}U_{\text{approx}}$ can keep only the fill-in entries $\mathbf{F}$ above a fixed tolerance. The MATLAB commands for incomplete LU are

$$[\,L, U, \texttt{Perm}\,] = \texttt{luinc}(A, \texttt{tol}) \qquad \text{or} \qquad R = \texttt{cholinc}(A, \texttt{tol})\,.$$

If you set `tol` $= 0$, those letters `inc` have no effect. This becomes ordinary sparse LU (and Cholesky for positive definite A). A large value of `tol` will remove all fill-in.

Difference matrices like $K2D$ can maintain zero row sums by adding entries below `tol` to the main diagonal (instead of destroying those entries completely). This **modified *ILU*** is a success (`mluinc` and `mcholinc`). The variety of options, and especially the fact that the computer can decide automatically how much fill-in to keep, has made incomplete LU a very popular starting point.

In the end, $Px_{k+1} = (P-A)x_k + b$ is too simple! Often the smooth (low frequency) errors decrease too slowly. *Multigrid will fix this.* And pure iteration is choosing one particular vector in a "Krylov subspace." With relatively little work we can make a much better choice of x_k. Multigrid methods and Krylov projections are the state of the art in today's iterative methods.

Problem Set 6.2

1 If $x^{\mathsf{old}} = (\cos\frac{k\pi}{N+1}\sin\frac{k\pi}{N+1}, \cos^2\frac{k\pi}{N+1}\sin\frac{2k\pi}{N+1}, \ldots, \cos^N\frac{k\pi}{N+1}\sin\frac{Nk\pi}{N+1})$ show that the Gauss-Seidel iteration (10) is satisfied with $x^{\mathsf{new}} = \left[\cos^2\frac{k\pi}{N+1}\right] x^{\mathsf{old}}$. This shows that M for Gauss-Seidel has $\lambda = \cos^2\frac{k\pi}{N+1}$ (squares of Jacobi eigenvalues).