11.3 Iterative Methods and Preconditioners

Up to now, our approach to $A\boldsymbol{x} = \boldsymbol{b}$ has been direct. We accepted $\boldsymbol{A}$ as it came. We attacked it by elimination with row exchanges. We now look at **iterative methods, which replace $\boldsymbol{A}$ by a simpler matrix $\boldsymbol{S}$**. The difference $T = S - A$ is moved over to the right side of the equation. The problem becomes easier to solve, with S instead of A. But there is a price—*the simpler system has to be solved over and over.*

An iterative method is easy to invent. Just split A (carefully) into $S - T$.

$$\textbf{Rewrite } A\boldsymbol{x} = \boldsymbol{b} \qquad S\boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{b}. \tag{1}$$

The novelty is to solve (1) iteratively. Each guess $\boldsymbol{x}_k$ leads to the next $\boldsymbol{x}_{k+1}$:

$$\textbf{Pure iteration} \qquad S\boldsymbol{x}_{k+1} = T\boldsymbol{x}_k + \boldsymbol{b}. \tag{2}$$

Start with any $\boldsymbol{x}_0$. Then solve $S\boldsymbol{x}_1 = T\boldsymbol{x}_0 + \boldsymbol{b}$. Continue to $S\boldsymbol{x}_2 = T\boldsymbol{x}_1 + \boldsymbol{b}$. A hundred iterations are very common—often more. Stop when (and if!) $\boldsymbol{x}_{k+1}$ is sufficiently close to $\boldsymbol{x}_k$—or when the **residual** $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ is near zero. Our hope is to get near the true solution, more quickly than by elimination. When the $\boldsymbol{x}_k$ converge, their limit x_∞ does solve equation (1): $S\boldsymbol{x}_\infty = T\boldsymbol{x}_\infty + \boldsymbol{b}$ means $A\boldsymbol{x}_\infty = \boldsymbol{b}$.

The two goals of the splitting $A = S - T$ are ***speed per step*** and ***fast convergence***. The speed of each step depends on S and the speed of convergence depends on $S^{-1}T$:

1 Equation (2) should be easy to solve for $\boldsymbol{x}_{k+1}$. The "***preconditioner***" S could be the diagonal or triangular part of A. A fast way uses $S = L_0U_0$, where those factors have many zeros compared to the exact $A = LU$. This is "*incomplete LU*".

2 The difference $\boldsymbol{x} - \boldsymbol{x}_k$ (this is the error $\boldsymbol{e}_k$) should go quickly to zero. Subtracting equation (2) from (1) cancels $\boldsymbol{b}$, and it leaves the ***equation for the error*** $\boldsymbol{e}_k$:

$$\textbf{Error equation} \qquad S\boldsymbol{e}_{k+1} = T\boldsymbol{e}_k \quad \text{which means} \quad \boldsymbol{e}_{k+1} = S^{-1}T\boldsymbol{e}_k. \tag{3}$$

At every step the error is multiplied by $S^{-1}T$. If $S^{-1}T$ is small, its powers go quickly to zero. But what is "small"?

The extreme splitting is $S = A$ and $T = 0$. Then the first step of the iteration is the original $A\boldsymbol{x} = \boldsymbol{b}$. Convergence is perfect and $S^{-1}T$ is zero. But the cost of that step is what we wanted to avoid. The choice of S is a battle between speed per step (a simple S) and fast convergence (S close to A). Here are some choices of S:

J $S =$ diagonal part of A (the iteration is called *Jacobi's method*)

GS $S =$ lower triangular part of A including the diagonal (*Gauss-Seidel method*)

ILU $S =$ approximate L times approximate U (*incomplete $L\,U$ method*).

Our first question is pure linear algebra: ***When do the x_k's converge to x?*** The answer uncovers the number $|\lambda|_{\max}$ that controls convergence. In examples of Jacobi and Gauss-Seidel, we will compute this "*spectral radius*" $|\lambda|_{\max}$. It is the largest eigenvalue of the ***iteration matrix*** $B = S^{-1}T$.

The Spectral Radius $\rho(B)$ Controls Convergence

Equation (3) is $\boldsymbol{e}_{k+1} = S^{-1}T\boldsymbol{e}_k$. Every iteration step multiplies the error by the same matrix $B = S^{-1}T$. The error after k steps is $\boldsymbol{e}_k = B^k\boldsymbol{e}_0$. ***The error approaches zero if the powers of $B = S^{-1}T$ approach zero.*** It is beautiful to see how the eigenvalues of B—the largest eigenvalue in particular—control the matrix powers B^k.

The powers B^k approach zero if and only if every eigenvalue of B has $|\lambda| < 1$.
The rate of convergence is controlled by the spectral radius of B: $\boldsymbol{\rho = \max|\lambda(B)|.}$

The test for convergence is $|\lambda|_{\max} < 1$. Real eigenvalues must lie between -1 and 1. Complex eigenvalues $\lambda = a + ib$ must have $|\lambda|^2 = a^2 + b^2 < 1$. The spectral radius "*rho*" is the largest distance from 0 to the eigenvalues of $B = S^{-1}T$. This is $\rho = |\lambda|_{\max}$.

To see why $|\lambda|_{\max} < 1$ is necessary, suppose the starting error $\boldsymbol{e}_0$ happens to be an eigenvector of B. After one step the error is $B\boldsymbol{e}_0 = \lambda\boldsymbol{e}_0$. After k steps the error is $B^k\boldsymbol{e}_0 = \lambda^k\boldsymbol{e}_0$. If we start with an eigenvector, we continue with that eigenvector—and the *factor* λ^k *only goes to zero when* $|\lambda| < 1$. This condition is required of every eigenvalue.

To see why $|\lambda|_{\max} < 1$ is sufficient for the error to approach zero, suppose $\boldsymbol{e}_0$ is a combination of eigenvectors:

$$\boldsymbol{e}_0 = c_1\boldsymbol{x}_1 + \cdots + c_n\boldsymbol{x}_n \quad \text{leads to} \quad \boldsymbol{e}_k = c_1(\lambda_1)^k\boldsymbol{x}_1 + \cdots + c_n(\lambda_n)^k\boldsymbol{x}_n. \tag{4}$$

This is the point of eigenvectors! When we multiply by B, each eigenvector $\boldsymbol{x}_i$ is multiplied by λ_i. If all $|\lambda_i| < 1$ then equation (4) ensures that $\boldsymbol{e}_k$ goes to zero.

Example 1 $B = \begin{bmatrix} .6 & .5 \\ .6 & .5 \end{bmatrix}$ has $\lambda_{\max} = 1.1$ $\qquad B' = \begin{bmatrix} .6 & 1.1 \\ 0 & .5 \end{bmatrix}$ has $\lambda_{\max} = .6$

B^2 is 1.1 times B. Then B^3 is $(1.1)^2$ times B. The powers of B will blow up. Contrast with the powers of B'. The matrix $(B')^k$ has $(.6)^k$ and $(.5)^k$ on its diagonal. The off-diagonal entries also involve $\rho^k = (.6)^k$, which sets the speed of convergence.

Note When there are too few eigenvectors, equation (4) is not correct. We turn to the *Jordan form* when eigenvectors are missing and the matrix B can't be diagonalized:

$$\textbf{Jordan form } J \qquad B = MJM^{-1} \qquad \text{and} \qquad B^k = MJ^kM^{-1}. \tag{5}$$

Section 8.3 shows how J and J^k are made of "blocks" with one repeated eigenvalue:

$$\text{The powers of a 2 by 2 block in } J \text{ are } \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix}^k = \begin{bmatrix} \lambda^k & k\lambda^{k-1} \\ 0 & \lambda^k \end{bmatrix}.$$

If $|\lambda| < 1$ then these powers approach zero. The extra factor k from a double eigenvalue is overwhelmed by the decreasing factor λ^{k-1}. This applies to every block:

Diagonalizable or not: Convergence $B^k \to 0$ and its speed depend on $\rho = |\lambda|_{\max} < 1$.

Jacobi versus Gauss-Seidel

We now solve a specific 2 by 2 problem by splitting A. Watch for that number $|\lambda|_{\max}$.

$$A\boldsymbol{x} = \boldsymbol{b} \qquad \begin{matrix} 2u - \ v = \ 4 \\ -u + 2v = -2 \end{matrix} \quad \text{has the solution} \quad \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}. \tag{6}$$

The first splitting is ***Jacobi's method***. Keep the *diagonal* of A on the left side (this is S). Move the off-diagonal part of A to the right side (this is T). Then iterate:

Jacobi iteration $\quad S\boldsymbol{x}_{k+1} = T\boldsymbol{x}_k + \boldsymbol{b}$ $\quad \begin{matrix} 2u_{k+1} & = v_k + 4 \\ 2v_{k+1} & = u_k - 2. \end{matrix}$

Start from $u_0 = v_0 = 0$. The first step finds $u_1 = 2$ and $v_1 = -1$. Keep going:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad \begin{bmatrix} 3/2 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 2 \\ -1/4 \end{bmatrix} \quad \begin{bmatrix} 15/8 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 2 \\ -1/16 \end{bmatrix} \quad \text{approaches} \quad \begin{bmatrix} 2 \\ 0 \end{bmatrix}.$$

This shows convergence. At steps 1, 3, 5 the second component is -1, $-1/4$, $-1/16$. Those drop by 4 in each two steps. *The error equation is $S\boldsymbol{e}_{k+1} = T\boldsymbol{e}_k$*:

$$\textbf{Error equation} \qquad \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \boldsymbol{e}_{k+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{e}_k \quad \text{or} \quad \boldsymbol{e}_{k+1} = \begin{bmatrix} 0 & \mathbf{\frac{1}{2}} \\ \mathbf{\frac{1}{2}} & 0 \end{bmatrix} \boldsymbol{e}_k. \tag{7}$$

That last matrix $S^{-1}T$ has eigenvalues $\frac{1}{2}$ and $-\frac{1}{2}$. So its spectral radius is $\rho(B) = \frac{1}{2}$:

$$B = S^{-1}T = \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix} \quad \text{has } |\lambda|_{\max} = \tfrac{1}{2} \quad \text{and} \quad \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix}^2 = \begin{bmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}.$$

Two steps multiply the error by $\frac{1}{4}$ exactly, in this special example. The important message is this: Jacobi's method works well when the main diagonal of A is large compared to the off-diagonal part. The diagonal part is S, the rest is $-T$. We want the diagonal to dominate.

The eigenvalue $\lambda = \frac{1}{2}$ is unusually small. Ten iterations reduce the error by $2^{10} = 1024$. More typical and more expensive is $|\lambda|_{\max} = .99$ or $.999$.

The ***Gauss-Seidel method*** keeps the whole lower triangular part of A as S:

$$\textbf{Gauss-Seidel} \qquad \begin{matrix} 2u_{k+1} & & = v_k + 4 \\ -u_{k+1} + 2v_{k+1} & = & -2 \end{matrix} \qquad \text{or} \qquad \begin{matrix} u_{k+1} = \frac{1}{2}v_k & & + 2 \\ v_{k+1} = \frac{1}{2}u_{k+1} & - & 1. \end{matrix} \tag{8}$$

Notice the change. The new u_{k+1} from the first equation is used *immediately* in the second equation. With Jacobi, we saved the old u_k until the whole step was complete. With Gauss-Seidel, the new values enter right away and the old u_k is destroyed. This cuts the storage in half. It also speeds up the iteration (usually). And it costs no more than the Jacobi method.

Test the iteration starting from another start $u_0 = 0$ and $v_0 = -1$:

$$\begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \begin{bmatrix} 3/2 \\ -1/4 \end{bmatrix} \quad \begin{bmatrix} 15/8 \\ -1/16 \end{bmatrix} \quad \begin{bmatrix} 63/32 \\ -1/64 \end{bmatrix} \quad \text{approaches} \quad \begin{bmatrix} 2 \\ 0 \end{bmatrix}.$$

The errors in the first component are 2, 1/2, 1/8, 1/32. The errors in the second component are -1, $-1/4$, $-1/16$, $-1/32$. We divide by 4 in *one step* not two steps. ***Gauss-Seidel is twice as fast as Jacobi***. We have $\rho_{\mathbf{GS}} = (\rho_{\mathbf{J}})^2$ when A is positive definite tridiagonal:

$$S = \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad S^{-1}T = \begin{bmatrix} \mathbf{0} & \frac{1}{2} \\ 0 & \frac{1}{4} \end{bmatrix}.$$

The Gauss-Seidel eigenvalues are 0 and $\frac{1}{4}$. Compare with $\frac{1}{2}$ and $-\frac{1}{2}$ for Jacobi.

With a small push we can describe the ***successive overrelaxation method*** (**SOR**). The new idea is to introduce a parameter ω (omega) into the iteration. Then choose this number ω to make the spectral radius of $S^{-1}T$ as small as possible.

Rewrite $A\boldsymbol{x} = \boldsymbol{b}$ as $\omega A\boldsymbol{x} = \omega\boldsymbol{b}$. The matrix S in **SOR** has the diagonal of the original A, but below the diagonal we use ωA. On the right side T is $S - \omega A$:

$$\textbf{SOR} \qquad \begin{aligned} 2u_{k+1} &= (2-2\omega)u_k + \omega v_k + 4\omega \\ -\omega u_{k+1} + 2v_{k+1} &= (2-2\omega)v_k - 2\omega. \end{aligned} \tag{9}$$

This looks more complicated to us, but the computer goes as fast as ever. SOR is like Gauss-Seidel, with an adjustable number ω. The best ω makes it faster.

I will put on record the most valuable test matrix of order n. It is our favorite -1, 2, -1 tridiagonal matrix K. The diagonal is $2I$. Below and above are -1's. Our example had $n = 2$, which leads to $\cos\frac{\pi}{3} = \frac{1}{2}$ as the Jacobi eigenvalue found above. Notice especially that this $|\lambda|_{\max}$ is squared for Gauss-Seidel:

The splittings of the -1, 2, -1 matrix K of order n yield these eigenvalues of B:

Jacobi ($S = 0, 2, 0$ matrix): $S^{-1}T$ has $|\lambda|_{\max} = \cos\dfrac{\pi}{n+1}$

Gauss-Seidel ($S = -1, 2, 0$ matrix): $S^{-1}T$ has $|\lambda|_{\max} = \left(\cos\dfrac{\pi}{n+1}\right)^2$

SOR (with the best ω): $S^{-1}T$ has $|\lambda|_{\max} = \left(\cos\dfrac{\pi}{n+1}\right)^2 \Big/ \left(1+\sin\dfrac{\pi}{n+1}\right)^2$.

Let me be clear: For the -1, 2, -1 matrix you should not use any of these iterations! Elimination on a tridiagonal matrix is very fast (exact LU). Iterations are intended for a large sparse matrix that has nonzeros far from the central diagonal. Those create many more nonzeros in the exact L and U. This **fill-in** is why elimination becomes expensive.

We mention one more splitting. The idea of "***incomplete*** $L\,U$" is to set the small nonzeros in L and U *back to zero*. This leaves triangular matrices L_0 and U_0 which are again sparse. The splitting has $S = L_0U_0$ on the left side. Each step is quick:

Incomplete LU $$L_0U_0\boldsymbol{x}_{k+1} = (L_0U_0 - A)\boldsymbol{x}_k + \boldsymbol{b}.$$

On the right side we do sparse matrix-vector multiplications. Don't multiply L_0 times U_0, those are matrices. Multiply $\boldsymbol{x}_k$ by U_0 and then multiply that vector by L_0. On the left side we do forward and back substitutions. If L_0U_0 is close to A, then $|\lambda|_{\max}$ is small. A few iterations will give a close answer.

Multigrid and Conjugate Gradients

I cannot leave the impression that Jacobi and Gauss-Seidel are great methods. Generally the "low-frequency" part of the error decays very slowly, and many iterations are needed. Here are two important ideas that bring tremendous improvement. **Multigrid** can solve problems of size n in $O(n)$ steps. With a good preconditioner, **conjugate gradients** becomes one of the most popular and powerful algorithms in numerical linear algebra.

Multigrid Solve smaller problems with coarser grids. Each iteration will be cheaper and faster. Then interpolate between the coarse grid values to get a quick headstart on the full-size problem. Multigrid might go 4 levels down and back.

Conjugate gradients An ordinary iteration like $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - A\boldsymbol{x}_k + \boldsymbol{b}$ involves multiplication by A at each step. If A is sparse, this is not too expensive: $A\boldsymbol{x}_k$ is what we are willing to do. It adds one more basis vector to the growing "Krylov spaces" that contain our approximations. But $\boldsymbol{x}_{k+1}$ is **not the best combination** of $\boldsymbol{x}_0$, $A\boldsymbol{x}_0, \ldots, A^k\boldsymbol{x}_0$. The ordinary iterations are simple but far from optimal.

The conjugate gradient method chooses **the best combination** $\boldsymbol{x}_k$ at every step. The extra cost (beyond one multiplication by A) is not great. We will give the CG iteration, emphasizing that this method was created for a *symmetric positive definite matrix*. When A is not symmetric, one good choice is GMRES. When $A = A^{\mathrm{T}}$ is not positive definite, there is MINRES. A world of high-powered iterative methods has been created around the idea of making optimal choices of each successive $\boldsymbol{x}_k$.

My textbook *Computational Science and Engineering* describes multigrid and CG in much more detail. Among books on numerical linear algebra, Trefethen-Bau is deservedly popular (others are terrific too). Golub-Van Loan is a level up.

The Problem Set reproduces the five steps in each conjugate gradient cycle from $\boldsymbol{x}_{k-1}$ to $\boldsymbol{x}_k$. We compute that new approximation $\boldsymbol{x}_k$, the new residual $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$, and the new search direction $\boldsymbol{d}_k$ to look for the next $\boldsymbol{x}_{k+1}$.

I wrote those steps for the original matrix A. But a **preconditioner** S can make convergence much faster. Our original equation is $A\boldsymbol{x} = \boldsymbol{b}$. The preconditioned equation is $S^{-1}A\boldsymbol{x} = S^{-1}\boldsymbol{b}$. Small changes in the code give the *preconditioned conjugate gradient method*—the leading iterative method to solve positive definite systems.

The biggest competition is direct elimination, with the equations reordered to take maximum advantage of the zeros in A. It is not easy to outperform Gauss.

Iterative Methods for Eigenvalues

We move from $A\boldsymbol{x} = \boldsymbol{b}$ to $A\boldsymbol{x} = \lambda\boldsymbol{x}$. Iterations are an option for linear equations. They are a necessity for eigenvalue problems. The eigenvalues of an n by n matrix are the roots of an nth degree polynomial. The determinant of $A - \lambda I$ starts with $(-\lambda)^n$. This book must not leave the impression that eigenvalues should be computed that way! Working from $\det(A - \lambda I) = 0$ is a *very poor approach*—except when n is small.

For $n > 4$ there is no formula to solve $\det(A - \lambda I) = 0$. Worse than that, the λ's can be very unstable and sensitive. It is much better to work with A itself, gradually making it diagonal or triangular. (Then the eigenvalues appear on the diagonal.) Good computer codes are available in the LAPACK library—individual routines are free on **www.netlib.org/lapack**. This library combines the earlier LINPACK and EISPACK, with many improvements (to use matrix-matrix operations in the Level 3 BLAS). It is a collection of Fortran 77 programs for linear algebra on high-performance computers. For your computer and mine, a high quality matrix package is all we need. For supercomputers with parallel processing, move to ScaLAPACK and block elimination.

We will briefly discuss the power method and the QR method (chosen by LAPACK) for computing eigenvalues. It makes no sense to give full details of the codes.

1 Power methods and inverse power methods. Start with any vector $\boldsymbol{u}_0$. Multiply by A to find $\boldsymbol{u}_1$. Multiply by A again to find $\boldsymbol{u}_2$. If $\boldsymbol{u}_0$ is a combination of the eigenvectors, then A multiplies each eigenvector $\boldsymbol{x}_i$ by λ_i. After k steps we have $(\lambda_i)^k$:

$$\boldsymbol{u}_k = A^k\boldsymbol{u}_0 = c_1(\lambda_1)^k\boldsymbol{x}_1 + \cdots + c_n(\lambda_n)^k\boldsymbol{x}_n. \tag{10}$$

As the power method continues, ***the largest eigenvalue begins to dominate***. The vectors $\boldsymbol{u}_k$ point toward that dominant eigenvector $\boldsymbol{x}_1$. We saw this for Markov matrices:

$$A = \begin{bmatrix} .9 & .3 \\ .1 & .7 \end{bmatrix} \quad \text{has} \quad \lambda_{\max} = 1 \quad \text{with eigenvector} \quad \begin{bmatrix} .75 \\ .25 \end{bmatrix}.$$

Start with $\boldsymbol{u}_0$ and multiply at every step by A:

$$\boldsymbol{u}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ \boldsymbol{u}_1 = \begin{bmatrix} .9 \\ .1 \end{bmatrix},\ \boldsymbol{u}_2 = \begin{bmatrix} .84 \\ .16 \end{bmatrix} \quad \text{is approaching} \quad \boldsymbol{u}_\infty = \begin{bmatrix} .75 \\ .25 \end{bmatrix}.$$

The speed of convergence depends on the *ratio* of the second largest eigenvalue λ_2 to the largest λ_1. We don't want λ_1 to be small, we want λ_2/λ_1 to be small. Here $\lambda_2 = .6$ and $\lambda_1 = 1$, giving good speed. For large matrices it often happens that $|\lambda_2/\lambda_1|$ is very close to 1. Then the power method is too slow.

Is there a way to find the *smallest* eigenvalue—which is often the most important in applications? Yes, by the *inverse* power method: Multiply $\boldsymbol{u}_0$ by A^{-1} instead of A. Since we never want to compute A^{-1}, we actually solve $A\boldsymbol{u}_1 = \boldsymbol{u}_0$. By saving the LU factors, the next step $A\boldsymbol{u}_2 = \boldsymbol{u}_1$ is fast. Step k has $Au_k = u_{k-1}$:

Inverse power method $$\boldsymbol{u}_k = A^{-k}\boldsymbol{u}_0 = \frac{c_1\boldsymbol{x}_1}{(\lambda_1)^k} + \cdots + \frac{c_n\boldsymbol{x}_n}{(\lambda_n)^k}. \tag{11}$$

Now the *smallest* eigenvalue $\lambda_{\min}$ is in control. When it is very small, the factor $1/\lambda_{\min}^k$ is large. For high speed, we make $\boldsymbol{\lambda}_{\min}$ even smaller by shifting the matrix to $A - \lambda^* I$.

That shift doesn't change the eigenvectors. (λ^* might come from the diagonal of A, even better is a Rayleigh quotient $x^{\mathrm{T}}Ax/x^{\mathrm{T}}x$). If λ^* is close to $\lambda_{\min}$ then $(A - \lambda^* I)^{-1}$ has the very large eigenvalue $(\lambda_{\min} - \lambda^*)^{-1}$. Each ***shifted inverse power step*** multiplies the eigenvector by this big number, and that eigenvector quickly dominates.

2 The QR Method This is a major achievement in numerical linear algebra. Sixty years ago, eigenvalue computations were slow and inaccurate. We didn't even realize that solving $\det(A - \lambda I) = 0$ was a terrible method. Jacobi had suggested earlier that A should gradually be made triangular—then the eigenvalues appear automatically on the diagonal. He used 2 by 2 rotations to produce off-diagonal zeros. (Unfortunately the previous zeros can become nonzero again. But Jacobi's method made a partial comeback with parallel computers.) The ***QR method*** is now a leader in eigenvalue computations.

The basic step is to factor A, whose eigenvalues we want, into QR. Remember from Gram-Schmidt (Section 4.4) that Q has orthonormal columns and R is triangular. For eigenvalues the key idea is: ***Reverse Q and R.*** The new matrix (same λ's) is $A_1 = RQ$. *The eigenvalues are not changed* in RQ *because* $A = QR$ *is similar to* $A_1 = Q^{-1}AQ$:

$\boldsymbol{A_1 = RQ}$ has the same $\boldsymbol{\lambda}$ $$QR\boldsymbol{x} = \lambda\boldsymbol{x} \quad \text{gives} \quad RQ(Q^{-1}\boldsymbol{x}) = \lambda(Q^{-1}\boldsymbol{x}). \tag{12}$$

This process continues. Factor the new matrix A_1 into Q_1R_1. Then reverse the factors to R_1Q_1. This is the similar matrix A_2 and again no change in the eigenvalues. Amazingly, those eigenvalues begin to show up on the diagonal. Soon the last entry of A_4 holds an accurate eigenvalue. In that case we remove the last row and column and continue with a smaller matrix to find the next eigenvalue.

Two extra ideas make this method a success. One is to shift the matrix by a multiple of I, before factoring into QR. Then RQ is shifted back to give A_{k+1}:

$$\text{Factor } A_k - c_k I \text{ into } Q_k R_k. \text{ The next matrix is } A_{k+1} = R_k Q_k + c_k I.$$

A_{k+1} has the same eigenvalues as A_k, and the same as the original $A_0 = A$. A good shift chooses c near an (unknown) eigenvalue. That eigenvalue appears more accurately on the diagonal of A_{k+1}—which tells us a better c for the next step to A_{k+2}.

The second idea is to obtain off-diagonal zeros before the QR method starts. An elimination step E will do it, or a Givens rotation, but don't forget E^{-1} (or λ will change):

$$EAE^{-1} = \begin{bmatrix} 1 & & \\ & 1 & \\ & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ \mathbf{1} & 4 & 5 \\ \mathbf{1} & \mathbf{6} & 7 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 3 \\ \mathbf{1} & 9 & 5 \\ \mathbf{0} & \mathbf{4} & 2 \end{bmatrix}. \textbf{ Same } \boldsymbol{\lambda}\textbf{'s.}$$

We must leave those nonzeros 1 and 4 along *one subdiagonal*. More E's could remove them, but E^{-1} would fill them in again. This is a "***Hessenberg matrix***" (one nonzero

subdiagonal). The zeros in the lower left corner will stay zero through the QR method. The operation count for each QR factorization drops from $O(n^3)$ to $O(n^2)$.

Golub and Van Loan give this example of one shifted QR step on a Hessenberg matrix. The shift is $7I$, taking 7 from all diagonal entries of A (then shifting back for A_1):

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & .001 & 7 \end{bmatrix} \quad \text{leads to} \quad A_1 = \begin{bmatrix} -.54 & 1.69 & 0.835 \\ .31 & 6.53 & -6.656 \\ 0 & .00002 & 7.012 \end{bmatrix}.$$

Factoring $A-7I$ into QR produced $A_1 = RQ+7I$. Notice the very small number .00002. The diagonal entry 7.012 is almost an exact eigenvalue of A_1, and therefore of A. Another QR step on A_1 with shift by $7.012I$ would give terrific accuracy.

For a few eigenvalues of a large sparse matrix I would look to **ARPACK**. Problems 25–27 describe the Arnoldi iteration that orthogonalizes the basis—each step has only three terms when A is symmetric. The matrix becomes *tridiagonal*: a wonderful start for computing eigenvalues.

Problem Set 11.3

Problems 1–12 are about iterative methods for $Ax = b$.

1 Change $A\boldsymbol{x} = \boldsymbol{b}$ to $\boldsymbol{x} = (I - A)\boldsymbol{x} + \boldsymbol{b}$. What are S and T for this splitting? What matrix $S^{-1}T$ controls the convergence of $\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + \boldsymbol{b}$?

2 If λ is an eigenvalue of A, then _____ is an eigenvalue of $B = I - A$. The real eigenvalues of B have absolute value less than 1 if the real eigenvalues of A lie between _____ and _____.

3 Show why the iteration $\boldsymbol{x}_{k+1} = (I-A)\boldsymbol{x}_k + \boldsymbol{b}$ does not converge for $A = \begin{bmatrix} \mathbf{2} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{2} \end{bmatrix}$.

4 Why is the norm of B^k never larger than $\|B\|^k$? Then $\|B\| < 1$ guarantees that the powers B^k approach zero (convergence). No surprise since $|\lambda|_{\max}$ is below $\|B\|$.

5 If A is singular then all splittings $A = S - T$ must fail. From $A\boldsymbol{x} = \boldsymbol{0}$ show that $S^{-1}T\boldsymbol{x} = \boldsymbol{x}$. So this matrix $B = S^{-1}T$ has $\lambda = 1$ and fails.

6 Change the 2's to 3's and find the eigenvalues of $S^{-1}T$ for Jacobi's method:

$$S\boldsymbol{x}_{k+1} = T\boldsymbol{x}_k + \boldsymbol{b} \quad \text{is} \quad \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \boldsymbol{x}_{k+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{b}.$$

7 Find the eigenvalues of $S^{-1}T$ for the Gauss-Seidel method applied to Problem 6:

$$\begin{bmatrix} 3 & 0 \\ -1 & 3 \end{bmatrix} \boldsymbol{x}_{k+1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x}_k + \boldsymbol{b}.$$

Does $|\lambda|_{\max}$ for Gauss-Seidel equal $|\lambda|^2_{\max}$ for Jacobi?

8 For any 2 by 2 matrix $\begin{bmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{bmatrix}$ show that $|\lambda|_{\max}$ equals $|bc/ad|$ for Gauss-Seidel and $|bc/ad|^{1/2}$ for Jacobi. We need $ad \neq 0$ for the matrix S to be invertible.

9 Write a computer code (MATLAB or other) for the Gauss-Seidel method. You can define S and T from A, or set up the iteration loop directly from the entries a_{ij}. Test it on the $-1, 2, -1$ matrices A of order 10, 20, 50 with $\boldsymbol{b} = (1, 0, \ldots, 0)$.

10 The Gauss-Seidel iteration at component i uses earlier parts of $\boldsymbol{x}^{\text{new}}$:

$$\textbf{Gauss-Seidel} \qquad x_i^{\text{new}} = x_i^{\text{old}} + \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{\text{new}} - \sum_{j=i}^{n} a_{ij}x_j^{\text{old}}\Big).$$

If every $x_i^{\text{new}} = x_i^{\text{old}}$ how does this show that the solution $\boldsymbol{x}$ is correct? How does the formula change for Jacobi's method? For **SOR** insert ω outside the parentheses.

11 Divide equation (10) by λ_1^k and explain why $|\lambda_2/\lambda_1|$ controls the convergence of the power method. Construct a matrix A for which this method *does not converge*.

12 The Markov matrix $A = \begin{bmatrix} \mathbf{.9} & \mathbf{.3} \\ \mathbf{.1} & \mathbf{.7} \end{bmatrix}$ has $\lambda = 1$ and .6, and the power method $\boldsymbol{u}_k = A^k\boldsymbol{u}_0$ converges to $\begin{bmatrix} \mathbf{.75} \\ \mathbf{.25} \end{bmatrix}$. Find the eigenvectors of A^{-1}. What does the inverse power method $\boldsymbol{u}_{-k} = A^{-k}\boldsymbol{u}_0$ converge to (after you multiply by $.6^k$)?

13 The tridiagonal matrix of size $n - 1$ with diagonals $-1, 2, -1$ has eigenvalues $\lambda_j = 2 - 2\cos(j\pi/n)$. Why are the smallest eigenvalues approximately $(j\pi/n)^2$? The inverse power method converges at the speed $\lambda_1/\lambda_2 \approx 1/4$.

14 For $A = \begin{bmatrix} \mathbf{2} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{2} \end{bmatrix}$ apply the power method $\boldsymbol{u}_{k+1} = A\boldsymbol{u}_k$ three times starting with $\boldsymbol{u}_0 = \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}$. What eigenvector is the power method converging to?

15 For $A = -1,\ 2, -1$ matrix, apply the *inverse* power method $\boldsymbol{u}_{k+1} = A^{-1}\boldsymbol{u}_k$ three times with the same $\boldsymbol{u}_0$. What eigenvector are the $\boldsymbol{u}_k$'s approaching?

16 In the QR method for eigenvalues when A is shifted to make $A_{22} = 0$, show that the $2, 1$ entry drops from $\mathbf{\sin\theta}$ in $A = QR$ to $\mathbf{-\sin^3\theta}$ in RQ. (*Compute R and RQ.*) This "cubic convergence" makes the method a success:

$$A = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & 0 \end{bmatrix} = QR = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & ? \\ 0 & ? \end{bmatrix}.$$

17 If A is an orthogonal matrix, its QR factorization has $Q =$ _____ and $R =$ _____. Therefore $RQ =$ _____. These are among the rare examples when the QR method goes nowhere.

18 The shifted QR method factors $A - cI$ into QR. Show that the next matrix $A_1 = RQ + cI$ equals $Q^{-1}AQ$. Therefore A_1 has the _____ eigenvalues as A (but A_1 is closer to triangular).

19 When $A = A^{\mathrm{T}}$, the "*Lanczos method*" finds a's and b's and orthonormal $\boldsymbol{q}$'s so that $A\boldsymbol{q}_j = b_{j-1}\boldsymbol{q}_{j-1} + a_j\boldsymbol{q}_j + b_j\boldsymbol{q}_{j+1}$ (with $\boldsymbol{q}_0 = \boldsymbol{0}$). Multiply by $\boldsymbol{q}_j^{\mathrm{T}}$ to find a formula for a_j. The equation says that $AQ = QT$ where T is a tridiagonal matrix.

20 The equation in Problem 19 develops from this loop with $b_0 = 1$ and $\boldsymbol{r}_0 =$ any $\boldsymbol{q}_1$:

$$\boldsymbol{q}_{j+1} = \boldsymbol{r}_j/b_j;\ j = j+1;\ a_j = \boldsymbol{q}_j^{\mathrm{T}} A\boldsymbol{q}_j;\ \boldsymbol{r}_j = A\boldsymbol{q}_j - b_{j-1}\boldsymbol{q}_{j-1} - a_j\boldsymbol{q}_j;\ b_j = \|\boldsymbol{r}_j\|.$$

Write a code and test it on the $-1, 2, -1$ matrix A. $Q^{\mathrm{T}}Q$ should be I.

21 Suppose A is *tridiagonal and symmetric in the QR method*. From $A_1 = Q^{-1}AQ$ show that A_1 is symmetric. Write $A_1 = RAR^{-1}$ to show that A_1 is also tridiagonal. (If the lower part of A_1 is proved tridiagonal then by symmetry the upper part is too.) Symmetric tridiagonal matrices are the best way to start in the QR method.

Problems 22–25 present two fundamental iterations. Each step involves $A\boldsymbol{q}$ or $A\boldsymbol{d}$.

The key point for large matrices is that matrix-vector multiplication is much faster than matrix-matrix multiplication. A crucial construction starts with a vector $\boldsymbol{b}$. Repeated multiplication will produce $A\boldsymbol{b}, A^2\boldsymbol{b}, \ldots$ but those vectors are far from orthogonal. The "**Arnoldi iteration**" creates an orthonormal basis $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots$ for the same space by the Gram-Schmidt idea: *orthogonalize each new $A\boldsymbol{q}_n$ against the previous $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{n-1}$*. The "Krylov space" spanned by $\boldsymbol{b}, A\boldsymbol{b}, \ldots, A^{n-1}\boldsymbol{b}$ then has a much better basis $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$.

Here in pseudocode are two of the most important algorithms in numerical linear algebra: Arnoldi gives a good basis and CG gives a good approximation to $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

Arnoldi Iteration	**Conjugate Gradient Iteration for Positive Definite A**	
$\boldsymbol{q}_1 = \boldsymbol{b}/\|\boldsymbol{b}\|$	$\boldsymbol{x}_0 = 0, \boldsymbol{r}_0 = \boldsymbol{b}, \boldsymbol{d}_0 = \boldsymbol{r}_0$	
for $n = 1$ **to** $N-1$	**for** $n = 1$ **to** N	
$\boldsymbol{v} = A\boldsymbol{q}_n$	$\alpha_n = (\boldsymbol{r}_{n-1}^{\mathrm{T}}\boldsymbol{r}_{n-1})/(\boldsymbol{d}_{n-1}^{\mathrm{T}}A\boldsymbol{d}_{n-1})$	step length $\boldsymbol{x}_{n-1}$ to $\boldsymbol{x}_n$
for $j = 1$ **to** n	$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \alpha_n\boldsymbol{d}_{n-1}$	approximate solution
$h_{jn} = \boldsymbol{q}_j^{\mathrm{T}}\boldsymbol{v}$	$\boldsymbol{r}_n = \boldsymbol{r}_{n-1} - \alpha_n A\boldsymbol{d}_{n-1}$	new residual $\boldsymbol{b} - A\boldsymbol{x}_n$
$\boldsymbol{v} = \boldsymbol{v} - h_{jn}\boldsymbol{q}_j$	$\beta_n = (\boldsymbol{r}_n^{\mathrm{T}}\boldsymbol{r}_n)/(\boldsymbol{r}_{n-1}^{\mathrm{T}}\boldsymbol{r}_{n-1})$	improvement this step
$h_{n+1,n} = \|\boldsymbol{v}\|$	$\boldsymbol{d}_n = \boldsymbol{r}_n + \beta_n\boldsymbol{d}_{n-1}$	next search direction
$\boldsymbol{q}_{n+1} = \boldsymbol{v}/h_{n+1,n}$	% *Notice: only* 1 *matrix-vector multiplication $A\boldsymbol{q}$ and $A\boldsymbol{d}$*	

For conjugate gradients, the residuals $\boldsymbol{r}_n$ are orthogonal and the search directions are A-orthogonal: all $\boldsymbol{d}_j^{\mathrm{T}}A\boldsymbol{d}_k = 0$. The iteration solves $A\boldsymbol{x} = \boldsymbol{b}$ by minimizing the error $\boldsymbol{e}^{\mathrm{T}}A\boldsymbol{e}$ over all vectors in the *Krylov space* = span of $\boldsymbol{b}, A\boldsymbol{b}, \ldots, A^{n-1}\boldsymbol{b}$. It is a fantastic algorithm.

22 For the diagonal matrix $A = \mathsf{diag}([1 \quad 2 \quad 3 \quad 4])$ and the vector $\boldsymbol{b} = (1, 1, 1, 1)$, go through one Arnoldi step to find the orthonormal vectors $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$.

23 Arnoldi's method is finding Q so that $AQ = QH$ (column by column):

$$AQ = \begin{bmatrix} A\boldsymbol{q}_1 & \cdots & A\boldsymbol{q}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_N \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1N} \\ h_{21} & h_{22} & \cdot & h_{2N} \\ 0 & h_{32} & \cdot & \cdot \\ 0 & 0 & \cdot & h_{NN} \end{bmatrix} = QH$$

H is a "Hessenberg matrix" with one nonzero subdiagonal. Here is the crucial fact when A is symmetric: ***The Hessenberg matrix*** $H = Q^{-1}AQ = Q^{\mathrm{T}}AQ$ ***is symmetric and therefore it is tridiagonal***. Explain that sentence.

24 This tridiagonal H (when A is symmetric) gives the **Lanczos iteration**:

$$\textbf{Three terms only} \qquad \boldsymbol{q}_{j+1} = (A\boldsymbol{q}_j - h_{j,j}\boldsymbol{q}_j - h_{j-1,j}\boldsymbol{q}_{j-1})/h_{j+1,j}$$

From $H = Q^{-1}AQ$, why are the eigenvalues of H the same as the eigenvalues of A? For large matrices, the "Lanczos method" computes the leading eigenvalues by stopping at a smaller tridiagonal matrix H_k. The QR method in the text is applied to compute the eigenvalues of H_k.

25 Apply the conjugate gradient method to solve $A\boldsymbol{x} = \boldsymbol{b} = \textbf{ones}(100, 1)$, where A is the $-1, 2, -1$ second difference matrix $A = \textbf{toeplitz}([2 \ -1 \ \textbf{zeros}(1, 98)])$. Graph $\boldsymbol{x}_{10}$ and $\boldsymbol{x}_{20}$ from CG, along with the exact solution $\boldsymbol{x}$. (Its 100 components are $x_i = (ih - i^2h^2)/2$ with $h = 1/101$. "$\textbf{plot}(i, x(i))$" should produce a parabola.)

26 For unsymmetric matrices, the spectral radius $\rho = \max|\lambda_i|$ is not a norm. But still $\|A^n\|$ grows or decays like ρ^n for large n. Compare those numbers for $A = [1 \quad 1; \quad 0 \quad 1.1]$ using the command **norm**.

$\boldsymbol{A^n \to 0}$ **if and only if** $\boldsymbol{\rho < 1}$. When $A = S^{-1}T$, this is the key to convergence.