Chapter 9

Numerical Linear Algebra

9.1 Gaussian Elimination in Practice

Numerical linear algebra is a struggle for *quick* solutions and also *accurate* solutions. We need efficiency but we have to avoid instability. In Gaussian elimination, the main freedom (always available) is to ***exchange equations***. This section explains when to exchange rows for the sake of speed, and when to do it for the sake of accuracy.

The key to accuracy is to avoid unnecessarily large numbers. Often that requires us to avoid small numbers! A small pivot generally means large multipliers (since we divide by the pivot). A good plan is "***partial pivoting***", to choose the *largest candidate* in each new column as the pivot. We will see why this strategy is built into computer programs.

Other row exchanges are done to save elimination steps. In practice, most large matrices are ***sparse***—almost all entries are zeros. Elimination is fastest when the equations are ordered to put those zeros (as far as possible) *outside the band of nonzeros*. Zeros inside the band "fill in" during elimination—the zeros are destroyed and don't help.

Section 9.2 is about instability that can't be avoided. It is built into the problem, and this sensitivity is measured by the "***condition number***". Then Section 9.3 describes how to solve $Ax = b$ by ***iterations***. Instead of direct elimination, the computer solves an easier equation many times. Each answer x_k leads to the next guess x_{k+1}. For good iterations, like **conjugate gradients**, the x_k converge quickly to $x = A^{-1}b$.

The Fastest Supercomputer

A new supercomputing record was announced by IBM and Los Alamos on May 20, 2008. The Roadrunner was the first to achieve a quadrillion (10^{15}) floating-point operations per second: *a petaflop machine*. The benchmark for this world record was a large dense linear system $Ax = b$: linear algebra.

The LINPACK software does elimination with partial pivoting. The biggest difference from this book is to organize the steps to use large submatrices and never single numbers. Roadrunner is a multicore Linux cluster with very remarkable processors, based on the

Cell Broadband Engine from Sony's PlayStation 3. The market for video games dwarfs scientific computing and led to astonishing acceleration in the chips.

This path to petascale is not the approach taken by IBM's BlueGene. A key issue was to count the standard quad-core processors that a petaflop machine would need: 32,000. The new architecture uses much less power, but its hybrid design has a price: a code needs three separate compilers and explicit instructions to move all the data. Please see the excellent article in *SIAM News* (**siam.org**, July 2008) and the details on **www.lanl.gov/roadrunner**.

The TOP500 project ranks the most powerful computer systems in the world. Roadrunner and BlueGene are #1 and #2 as this page is written in 2009.

Our thinking about matrix calculations is reflected in the highly optimized **BLAS** (*Basic Linear Algebra Subroutines*). They come at levels 1, 2, and 3:

1 Linear combinations of vectors $au + v$: $O(n)$ work

2 Matrix-vector multiplications $Au + v$: $O(n^2)$ work

3 Matrix-matrix multiplications $AB + C$: $O(n^3)$ work

Level 1 is a single elimination step (multiply row j by ℓ_{ij} and subtract from row i). Level 2 can eliminate a whole column at once. A high performance solver is rich in Level 3 BLAS (AB has $2n^3$ flops and $2n^2$ data, a good ratio of work to talk).

It is *data passing* and *storage retrieval* that limit the speed of parallel processing. The high-velocity cache between main memory and floating-point computation has to be fully used! Top speed demands a ***block matrix approach*** to elimination.

The big change, coming now, is parallel processing at the chip level.

Roundoff Error and Partial Pivoting

Up to now, any pivot (nonzero of course) was accepted. In practice a small pivot is dangerous. A catastrophe can occur when numbers of different sizes are added. Computers keep a fixed number of significant digits (say three decimals, for a very weak machine). The sum $10{,}000 + 1$ is rounded off to 10,000. The "1" is completely lost. Watch how that changes the solution to this problem:

$$\begin{aligned} .0001u + v &= 1 \\ -u + v &= 0 \end{aligned} \quad \text{starts with coefficient matrix} \quad A = \begin{bmatrix} .0001 & 1 \\ -1 & 1 \end{bmatrix}.$$

If we accept .0001 as the pivot, elimination adds 10,000 times row 1 to row 2. Roundoff leaves

$$10{,}000v = 10{,}000 \qquad \text{instead of} \qquad 10{,}001v = 10{,}000.$$

The computed answer $v = 1$ is near the true $v = .9999$. But then back substitution puts the wrong v into the equation for u:

$$.0001\,u + 1 = 1 \quad \text{instead of} \quad .0001\,u + .9999 = 1.$$

The first equation gives $u = 0$. The correct answer (look at the second equation) is $u = 1.000$. By losing the "1" in the matrix, we have lost the solution. ***The change from* 10,001 *to* 10,000 *has changed the answer from* $u = 1$ *to* $u = 0$** (100% error!).

If we exchange rows, even this weak computer finds an answer that is correct to three places:

$$\begin{matrix} -u + v = 0 \\ .0001u + v = 1 \end{matrix} \quad \longrightarrow \quad \begin{matrix} -u + v = 0 \\ v = 1 \end{matrix} \quad \longrightarrow \quad \begin{matrix} u = 1 \\ v = 1. \end{matrix}$$

The original pivots were .0001 and 10,000—badly scaled. After a row exchange the exact pivots are -1 and 1.0001—well scaled. The computed pivots -1 and 1 come close to the exact values. Small pivots bring numerical instability, and the remedy is ***partial pivoting***. The kth pivot is decided when we reach and search column k:

Choose the largest number in row k or below. Exchange its row with row k.

The strategy of *complete pivoting* looks also in later columns for the largest pivot. It exchanges columns as well as rows. This expense is seldom justified, and all major codes use partial pivoting. Multiplying a row or column by a scaling constant can also be very worthwhile. *If the first equation above is* $u + 10{,}000v = 10{,}000$ *and we don't rescale, then* 1 *looks like a good pivot and we would miss the essential row exchange.*

For positive definite matrices, row exchanges are *not* required. It is safe to accept the pivots as they appear. Small pivots can occur, but the matrix is not improved by row exchanges. When its condition number is high, the problem is in the matrix and not in the code. In this case the output is unavoidably sensitive to the input.

The reader now understands how a computer actually solves $Ax = b$***—by elimination with partial pivoting***. Compared with the theoretical description—***find*** A^{-1} ***and multiply*** $A^{-1}b$—the details took time. But in computer time, elimination is much faster. I believe this algorithm is also the best approach to the algebra of row spaces and nullspaces.

Operation Counts: Full Matrices and Band Matrices

Here is a practical question about cost. *How many separate operations are needed to solve* $Ax = b$ *by elimination*? This decides how large a problem we can afford.

Look first at A, which changes gradually into U. When a multiple of row 1 is subtracted from row 2, we do n operations. The first is a division by the pivot, to find the multiplier ℓ. For the other $n - 1$ entries along the row, the operation is a "multiply-subtract". For convenience, we count this as a single operation. If you regard multiplying by ℓ and subtracting from the existing entry as two separate operations, *multiply all our counts by* 2.

The matrix A is n by n. The operation count applies to all $n - 1$ rows below the first. Thus it requires n times $n - 1$ operations, or $n^2 - n$, to produce zeros below the first pivot. *Check: All* n^2 *entries are changed, except the n entries in the first row.*

When elimination is down to k equations, the rows are shorter. We need only $k^2 - k$ operations (instead of $n^2 - n$) to clear out the column below the pivot. This is true for $1 \le k \le n$. The last step requires no operations ($1^2 - 1 = 0$), since the pivot is set and forward elimination is complete. The total count to reach U is the sum of $k^2 - k$ over all values of k from 1 to n:

$$(1^2 + \cdots + n^2) - (1 + \cdots + n) = \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{n^3 - n}{3}.$$

Those are known formulas for the sum of the first n numbers and the sum of the first n squares. Substituting $n = 1$ into $n^3 - n$ gives zero. Substituting $n = 100$ gives a million minus a hundred—then divide by 3. (That translates into one second on a workstation.) We will ignore the last term n in comparison with the larger term n^3, to reach our main conclusion:

The multiply-subtract count for forward elimination (A to U, producing L) is $\frac{1}{3}n^3$.

That means $\frac{1}{3}n^3$ multiplications and $\frac{1}{3}n^3$ subtractions. Doubling n increases this cost by eight (because n is cubed). 100 equations are easy, 1000 are more expensive, 10000 dense equations are close to impossible. We need a faster computer or a lot of zeros or a new idea.

On the right side of the equations, the steps go much faster. We operate on single numbers, not whole rows. ***Each right side needs exactly n^2 operations.*** Down and back up we are solving two triangular systems, $Lc = \boldsymbol{b}$ forward and $Ux = c$ backward. In back substitution, the last unknown needs only division by the last pivot. The equation above it needs two operations—substituting x_n and dividing by *its* pivot. The kth step needs k multiply-subtract operations, and the total for back substitution is

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \tfrac{1}{2}n^2 \quad \text{operations.}$$

The forward part is similar. *The n^2 total exactly equals the count for multiplying $A^{-1}\boldsymbol{b}$!* This leaves Gaussian elimination with two big advantages over $A^{-1}\boldsymbol{b}$:

1 Elimination requires $\frac{1}{3}n^3$ compared to n^3 for A^{-1}.

2 If A is *banded* so are L and U. But A^{-1} is full of nonzeros.

Band Matrices

These counts are improved when A has "*good zeros*". A good zero is an entry that remains zero in L and U. ***The best zeros are at the beginning of a row.*** They require no elimination steps (the multipliers are zero). So we also find those same good zeros in L. That is especially clear for this *tridiagonal matrix* A:

Tridiagonal
Bidiagonal
times
bidiagonal

$$\begin{bmatrix} 1 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & -1 & 1 & \\ & & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & & \\ & 1 & -1 & \\ & & 1 & -1 \\ & & & 1 \end{bmatrix}.$$

Rows 3 and 4 of A begin with zeros. No multiplier is needed, so L has the same zeros. Also columns 3 and 4 *start* with zeros. When a multiple of row 1 is subtracted from row 2, no calculation is required beyond the second column. The rows are short. They stay short! Figure 9.1 shows how a band matrix A has band factors L and U.

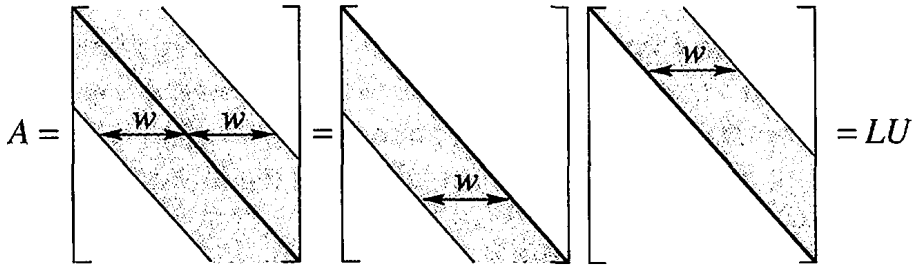


Figure 9.1: $A = LU$ for a band matrix. Good zeros in A *stay zero* in L and U.

These zeros lead to a complete change in the operation count, for "half-bandwidth" w:

A band matrix has $a_{ij} = 0$ when $|i - j| > w$.

Thus $w = 1$ for a diagonal matrix, $w = 2$ for tridiagonal, $w = n$ for dense. The length of the pivot row is at most w. There are no more than $w - 1$ nonzeros below any pivot. Each stage of elimination is complete after $w(w-1)$ operations, and *the band structure survives.* There are n columns to clear out. Therefore:

Elimination on a band matrix (A to L and U) needs less than $w^2 n$ operations.

For a band matrix, the count is proportional to n instead of n^3. It is also proportional to w^2. A full matrix has $w = n$ and we are back to n^3. For an exact count, remember that the bandwidth drops below w in the lower right corner (not enough space):

$$\textbf{Band} \quad \frac{w(w-1)(3n-2w+1)}{3} \qquad \textbf{Dense} \quad \frac{n(n-1)(n+1)}{3} = \frac{n^3-n}{3}$$

On the right side, to find $\boldsymbol{x}$ from $\boldsymbol{b}$, the cost is about $2wn$ (compared to the usual n^2). *Main point: For a band matrix the operation counts are* ***proportional to*** n. This is extremely fast. A tridiagonal matrix of order 10,000 is very cheap, provided *we don't compute* A^{-1}. That inverse matrix has no zeros at all:

$$A = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \quad \text{has} \quad A^{-1} = U^{-1}L^{-1} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

We are actually worse off knowing A^{-1} than knowing L and U. Multiplication by A^{-1} needs the full n^2 steps. Solving $Lc = b$ and $Ux = c$ needs only $2wn$. A band structure is very common in practice, when the matrix reflects connections between near neighbors: $a_{13} = 0$ and $a_{14} = 0$ because 1 is not a neighbor of 3 and 4.

We close with counts for Gauss-Jordan and Gram-Schmidt-Householder:

A^{-1} costs n^3 multiply-subtract steps. ***QR costs $\frac{2}{3}n^3$ steps.***

Start with $AA^{-1} = I$. The jth column of A^{-1} solves $Ax_j =$ jth column of I. The left side costs $\frac{1}{3}n^3$ as usual. (This is a one-time cost! L and U are not repeated.) The special

saving for the jth column of I comes from its first $j-1$ zeros. No work is required on the right side until elimination reaches row j. The forward cost is $\frac{1}{2}(n-j)^2$ instead of $\frac{1}{2}n^2$. Summing over j, the total for forward elimination on the n right sides is $\frac{1}{6}n^3$. The final multiply-subtract count for A^{-1} is n^3 if we actually want the inverse:

$$\textbf{For } A^{-1} \qquad \frac{n^3}{3}\,(L \text{ and } U) + \frac{n^3}{6}\,(\text{forward}) + n\left(\frac{n^2}{2}\right)(\text{back substitutions}) = \boldsymbol{n^3}. \quad (1)$$

Orthogonalization *(A to Q)*: The key difference from elimination is that *each multiplier is decided by a dot product.* That takes n operations, where elimination just divides by the pivot. Then there are n "multiply-subtract" operations to remove from column k its projection along column $j < k$ (see Section 4.4). The combined cost is $2n$ where for elimination it is n. This factor 2 is the price of orthogonality. We are changing a dot product to zero where elimination changes an entry to zero.

Caution To judge a numerical algorithm, it is **not enough** to count the operations. Beyond "flop counting" is a study of stability (Householder wins) and the flow of data.

Reordering Sparse Matrices

In discussing band matrices, we assumed a constant width w. The rows were in an optimal order. But for most sparse matrices in real computations, the width of the band is *not constant* and there are many zeros inside the band. Those zeros can fill in as elimination proceeds—they are lost. We need to ***renumber the equations to reduce fill-in***, and thereby speed up elimination.

Generally speaking, we want to move zeros to early rows and columns. Later rows and columns are shorter anyway. The "approximate minimum degree" algorithm in sparse MATLAB is *greedy*—it chooses the row to eliminate without counting all the consequences. We may reach a nearly full matrix near the end, but the total operation count to reach LU is still much smaller. To renumber for an absolute minimum of nonzeros in L and U is an NP-hard problem, much too expensive, and **amd** is a good compromise.

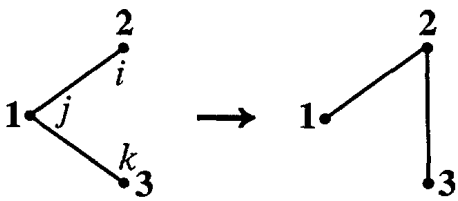
We only need the *positions* of the nonzeros, not their exact values. Think of the n rows as n nodes in a graph. *Node i is connected to node j if $a_{ij} \neq 0$.* Watch to see how elimination can create a new edge from i to k. This means that a zero is filled in, which we are trying to avoid:

When a_{kj} is eliminated, a multiple of the pivot row $j = 1$ is subtracted from row $k = 3$.

If a_{ji} was nonzero in row j, then a_{ki} becomes nonzero in the new row k. A new edge.

$$\begin{bmatrix} 1 & 1 & 1 \\ -2 & 1 & \mathbf{0} \\ -2 & \mathbf{0} & 2 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 3 & \mathbf{2} \\ 0 & \mathbf{2} & 4 \end{bmatrix}$$

$a_{32} = \mathbf{0}$ $\qquad$ $a_{32} = \mathbf{2}$



$a_{32} = \mathbf{0}$ **before** $\qquad$ $a_{32} \neq \mathbf{0}$ **after**

In this example, the 1's change the 0's into 2's. Those entries fill in.

The graph shows each step—look at the eliminationmovie on **math.mit.edu/18086**. The command **nnz**(L) counts the nonzero multipliers in the lower triangular L, **find** (L) will list them, and **spy**(L) shows them all.

The matrix in the movie is the 2D version of our $-1, 2, -1$ matrix. Instead of second differences along a line, the matrix has x and y differences on a plane grid. Each point is connected to its four nearest neighbors. But it is impossible to number all the points so that neighbors stay together. If we number by rows of the grid, there is a long wait to come around to the gridpoint above.

The goal of **colamd** and **symamd** is a better ordering (permutation P) that reduces fill-in for PA and PAP^{T}—*by choosing the pivot with the fewest nonzeros below it.*

Fast Orthogonalization

There are three ways to reach the important factorization $A = QR$. Gram-Schmidt works to find the orthonormal vectors in Q. Then R is upper triangular because of the order of Gram-Schmidt steps. Now we look at better methods (Householder and Givens), which use a product of specially simple Q's that we *know* are orthogonal.

Elimination gives $A = LU$, orthogonalization gives $A = QR$. We don't want a triangular L, we want an orthogonal Q. L is a product of E's, with 1's on the diagonal and the multiplier ℓ_{ij} below. *Q will be a product of orthogonal matrices.*

There are two simple orthogonal matrices to take the place of the E's. The ***reflection matrices*** $I - 2uu^{\mathrm{T}}$ are named after Householder. The ***plane rotation matrices*** are named after Givens. The simple matrix that rotates the xy plane by θ is Q_{21}:

Givens rotation
$$Q_{21} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Use Q_{21} the way you used E_{21}, to produce a zero in the $(2, 1)$ position. That determines the angle θ. Bill Hager gives this example in *Applied Numerical Linear Algebra*:

$$Q_{21}A = \begin{bmatrix} .6 & .8 & 0 \\ -.8 & .6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 90 & -153 & 114 \\ 120 & -79 & -223 \\ 200 & -40 & 395 \end{bmatrix} = \begin{bmatrix} 150 & -155 & -110 \\ 0 & 75 & -225 \\ 200 & -40 & 395 \end{bmatrix}.$$

The zero came from $-.8(90) + .6(120)$. No need to find θ, what we needed was $\cos\theta$:

$$\cos\theta = \frac{90}{\sqrt{90^2 + 120^2}} \quad \text{and} \quad \sin\theta = \frac{-120}{\sqrt{90^2 + 120^2}}. \tag{2}$$

Now we attack the $(3, 1)$ entry. The rotation will be in rows and columns 3 and 1. The numbers $\cos\theta$ and $\sin\theta$ are determined from 150 and 200, instead of 90 and 120.

$$Q_{31}Q_{21}A = \begin{bmatrix} .6 & 0 & .8 \\ 0 & 1 & 0 \\ -.8 & 0 & .6 \end{bmatrix} \begin{bmatrix} 150 & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 200 & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} 250 & -125 & 250 \\ 0 & 75 & -225 \\ 0 & 100 & 325 \end{bmatrix}.$$

One more step to R. The $(3,2)$ entry has to go. The numbers $\cos\theta$ and $\sin\theta$ now come from 75 and 100. The rotation is now in rows and columns 2 and 3:

$$Q_{32}Q_{31}Q_{21}A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & .6 & .8 \\ 0 & -.8 & .6 \end{bmatrix} \begin{bmatrix} 250 & -125 & \cdot \\ 0 & 75 & \cdot \\ 0 & 100 & \cdot \end{bmatrix} = \begin{bmatrix} 250 & -125 & 250 \\ 0 & 125 & 125 \\ 0 & 0 & 375 \end{bmatrix}.$$

We have reached the upper triangular R. What is Q? Move the plane rotations Q_{ij} to the other side to find $A = QR$—just as you moved the elimination matrices E_{ij} to the other side to find $A = LU$:

$$Q_{32}Q_{31}Q_{21}A = R \qquad \text{means} \qquad A = (Q_{21}^{-1}Q_{31}^{-1}Q_{32}^{-1})R = QR. \tag{3}$$

The inverse of each Q_{ij} is Q_{ij}^{T} (rotation through $-\theta$). The inverse of E_{ij} was not an orthogonal matrix! LU and QR are similar but not the same.

Householder reflections are faster because each one clears out a whole column below the diagonal. Watch how the first column $\boldsymbol{a}_1$ of A becomes column $\boldsymbol{r}_1$ of R:

Reflection by H_1
$H_1 = I - 2\boldsymbol{u}_1\boldsymbol{u}_1^{\mathrm{T}}$

$$H_1\,\boldsymbol{a}_1 = \begin{bmatrix} \|\boldsymbol{a}_1\| \\ 0 \\ \cdot \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -\|\boldsymbol{a}_1\| \\ 0 \\ \cdot \\ 0 \end{bmatrix} = \boldsymbol{r}_1\,. \tag{4}$$

The length was not changed, and $\boldsymbol{u}_1$ is in the direction of $\boldsymbol{a}_1 - \boldsymbol{r}_1$. We have $n-1$ entries in the unit vector $\boldsymbol{u}_1$ to get $n-1$ zeros in $\boldsymbol{r}_1$. (Rotations had one angle θ to get one zero.) When we reach column k, $n-k$ available choices in the unit vector $\boldsymbol{u}_k$ lead to $n-k$ zeros in $\boldsymbol{r}_k$. *We just store the $\boldsymbol{u}$'s and $\boldsymbol{r}$'s to know Q and R:*

Inverse of H_i is H_i $\quad (H_{n-1}\ldots H_1)A = R$ means $A = (H_1\ldots H_{n-1})R = QR$. (5)

This is how LAPACK improves on Gram-Schmidt. Q is *exactly* orthogonal.

Section 9.3 explains how $A = QR$ is used in the other big computation of linear algebra—***the eigenvalue problem***. The factors QR are reversed to give $A_1 = RQ$ which is $Q^{-1}AQ$. Since A_1 is similar to A, the eigenvalues are unchanged. Then A_1 is factored into Q_1R_1, and reversing the factors gives A_2. Amazingly, the entries below the diagonal get smaller in $A_1, A_2, A_3, \ldots$ and we can identify the eigenvalues. This is the "QR method" for $A\boldsymbol{x} = \lambda\boldsymbol{x}$, a big success of numerical linear algebra.

Problem Set 9.1

1 Find the two pivots with and without row exchange to maximize the pivot:

$$A = \begin{bmatrix} .001 & 0 \\ 1 & 1000 \end{bmatrix}.$$

With row exchanges to maximize pivots, why are no entries of L larger than 1? Find a 3 by 3 matrix A with all $|a_{ij}| \le 1$ and $|\ell_{ij}| \le 1$ but third pivot $= 4$.

2 Compute the exact inverse of the Hilbert matrix A by elimination. Then compute A^{-1} again by rounding all numbers to three figures:

Ill-conditioned matrix $$A = \text{hilb}(3) = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}.$$

3 For the same A compute $\boldsymbol{b} = A\boldsymbol{x}$ for $\boldsymbol{x} = (1, 1, 1)$ and $\boldsymbol{x} = (0, 6, -3.6)$. A small change $\Delta \boldsymbol{b}$ produces a large change $\Delta \boldsymbol{x}$.

4 Find the eigenvalues (by computer) of the 8 by 8 Hilbert matrix $a_{ij} = 1/(i + j - 1)$. In the equation $A\boldsymbol{x} = \boldsymbol{b}$ with $\|\boldsymbol{b}\| = 1$, how large can $\|\boldsymbol{x}\|$ be? If $\boldsymbol{b}$ has roundoff error less than 10^{-16}, how large an error can this cause in $\boldsymbol{x}$? See Section 9.2.

5 For back substitution with a band matrix (width w), show that the number of multiplications to solve $U\boldsymbol{x} = \boldsymbol{c}$ is approximately wn.

6 If you know L and U and Q and R, is it faster to solve $LU\boldsymbol{x} = \boldsymbol{b}$ or $QR\boldsymbol{x} = \boldsymbol{b}$?

7 Show that the number of multiplications to invert an upper triangular n by n matrix is about $\frac{1}{6}n^3$. Use back substitution on the columns of I, upward from 1's.

8 Choosing the largest available pivot in each column (partial pivoting), factor each A into $PA = LU$:

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 2 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 2 & 0 \\ 0 & 2 & 0 \end{bmatrix}.$$

9 Put 1's on the three central diagonals of a 4 by 4 tridiagonal matrix. Find the cofactors of the six zero entries. Those entries are nonzero in A^{-1}.

10 (Suggested by C. Van Loan.) Find the LU factorization and solve by elimination when $\varepsilon = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}, 10^{-15}$:

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 + \varepsilon \\ 2 \end{bmatrix}.$$

The true $\boldsymbol{x}$ is $(1, 1)$. Make a table to show the error for each ε. Exchange the two equations and solve again—the errors should almost disappear.

11 (a) Choose $\sin\theta$ and $\cos\theta$ to triangularize A, and find R:

Givens rotation $$Q_{21}A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} * & * \\ 0 & * \end{bmatrix} = R.$$

(b) Choose $\sin\theta$ and $\cos\theta$ to make QAQ^{-1} triangular. What are the eigenvalues?

12 When A is multiplied by a plane rotation Q_{ij}, which n^2 entries of A are changed? When $Q_{ij}A$ is multiplied on the right by Q_{ij}^{-1}, which entries are changed now?

13 How many multiplications and how many additions are used to compute $Q_{ij}A$? Careful organization of the whole sequence of rotations gives $\frac{2}{3}n^3$ multiplications and $\frac{2}{3}n^3$ additions—the same as for QR by reflectors and twice as many as for LU.

Challenge Problems

14 (**Turning a robot hand**) The robot produces any 3 by 3 rotation A from plane rotations around the x, y, z axes. Then $Q_{32}Q_{31}Q_{21}A = R$, where A is orthogonal so R is I! The three robot turns are in $A = Q_{21}^{-1}Q_{31}^{-1}Q_{32}^{-1}$. The three angles are "Euler angles" and $\det Q = 1$ to avoid reflection. Start by choosing $\cos\theta$ and $\sin\theta$ so that

$$Q_{21}A = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{3} \begin{bmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ 2 & 2 & -1 \end{bmatrix} \text{ is zero in the } (2,1) \text{ position.}$$

15 Create the 10 by 10 second difference matrix $K = \mathbf{toeplitz}([2\ -1\ \mathbf{zeros}(1,8)])$. Permute rows and columns randomly by $KK = K(\mathbf{randperm}(10), \mathbf{randperm}(10))$. Factor by $[L,U] = \mathbf{lu}(K)$ and $[LL,UU] = \mathbf{lu}(KK)$, and count nonzeros by $\mathbf{nnz}(L)$ and $\mathbf{nnz}(LL)$. In this case L is in perfect tridiagonal order, but not LL.

16 Another ordering for this matrix K colors the meshpoints alternately red and black. This permutation P changes the normal $1, \ldots, 10$ to $1, 3, 5, 7, 9, 2, 4, 6, 8, 10$:

Red-black ordering $\quad PKP^{\mathrm{T}} = \begin{bmatrix} 2I & D \\ D^{\mathrm{T}} & 2I \end{bmatrix}$. *Find the matrix* D.

So many interesting experiments are possible. If you send good ideas they can go on the linear algebra website math.mit.edu/linearalgebra. I also recommend learning the command $B = \mathbf{sparse}(A)$, after which $\mathbf{find}(B)$ will list the nonzero entries and $\boldsymbol{\ell}\mathbf{u}(B)$ will factor B using that sparse format for L and U. Only the nonzeros are computed, where ordinary (dense) MATLAB computes all the zeros too.

17 Jeff Stuart has created a student activity that brilliantly demonstrates ill-conditioning:

$$\begin{bmatrix} 1 & 1.0001 \\ 1 & 1.0000 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3.0001 + e \\ 3.0000 + E \end{bmatrix} \quad \begin{matrix} \textbf{With errors} \\ e \textbf{ and } E \end{matrix} \quad \begin{matrix} x = 2 - 10000(e - E) \\ y = 1 + 10000(e - E) \end{matrix}$$

The algebra shows how errors e and E are amplified by 10000 unless $e = E$.

As always, the solution of a 2 by 2 system is the meeting point of two lines. The neat idea is to replace mathematical lines by ***long sticks held by students.*** The sticks for these two equations are almost parallel, and A is almost singular. Perpendicular sticks come from well-conditioned equations.

In Stuart's *Shake a Stick* activity, the students plot where the sticks cross (after multiple shakes). See ***www.plu.edu/~stuartjl*** for the wild movements of that crossing point (x, y), when the sticks are nearly parallel.

9.2 Norms and Condition Numbers

How do we measure the size of a matrix? For a vector, the length is $\|x\|$. For a matrix, ***the norm is*** $\|A\|$. This word "*norm*" is sometimes used for vectors, instead of length. It is always used for matrices, and there are many ways to measure $\|A\|$. We look at the requirements on all "matrix norms" and then choose one.

Frobenius squared all the $|a_{ij}|^2$ and added; his norm $\|A\|_F$ is the square root. This treats A like a long vector with n^2 components: sometimes useful, but not the choice here.

I prefer to start with a vector norm. The triangle inequality says that $\|x + y\|$ is not greater than $\|x\| + \|y\|$. The length of $2x$ or $-2x$ is doubled to $2\|x\|$. The same rules will apply to matrix norms:

$$\|A + B\| \le \|A\| + \|B\| \quad \text{and} \quad \|cA\| = |c|\,\|A\|. \tag{1}$$

The second requirements for a matrix norm are new, because matrices multiply. The norm $\|A\|$ controls the growth from x to Ax, and from B to AB:

Growth factor $\|A\|$ $$\|Ax\| \le \|A\|\,\|x\| \quad \text{and} \quad \|AB\| \le \|A\|\,\|B\|. \tag{2}$$

This leads to a natural way to define $\|A\|$, the norm of a matrix:

The norm of A is the largest ratio $\|Ax\|/\|x\|$: $$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}. \tag{3}$$

$\|Ax\|/\|x\|$ is never larger than $\|A\|$ (its maximum). This says that $\|Ax\| \le \|A\|\,\|x\|$.

Example 1 If A is the identity matrix I, the ratios are $\|x\|/\|x\|$. Therefore $\|I\| = 1$. If A is an orthogonal matrix Q, lengths are again preserved: $\|Qx\| = \|x\|$. The ratios still give $\|Q\| = 1$. An orthogonal Q is good to compute with: errors don't grow.

Example 2 The norm of a diagonal matrix is its largest entry (using absolute values):

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \quad \text{has norm} \quad \|A\| = 3. \quad \text{The eigenvector} \quad x = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{has} \quad Ax = 3x.$$

The eigenvalue is 3. For this A (but not all A), the largest eigenvalue equals the norm.

For a positive definite symmetric matrix the norm is $\|A\| = \lambda_{\max}(A)$.

Choose x to be the eigenvector with maximum eigenvalue. Then $\|Ax\|/\|x\|$ equals $\lambda_{\max}$. The point is that no other x can make the ratio larger. The matrix is $A = Q\Lambda Q^T$, and the orthogonal matrices Q and Q^T leave lengths unchanged. So the ratio to maximize is really $\|\Lambda x\|/\|x\|$. The norm is the largest eigenvalue in the diagonal Λ.

Symmetric matrices Suppose A is symmetric but not positive definite. $A = Q\Lambda Q^T$ is still true. Then the norm is the largest of $|\lambda_1|, |\lambda_2|, \ldots, |\lambda_n|$. We take absolute values,

because the norm is only concerned with length. For an eigenvector $\|Ax\| = \|\lambda x\| = |\lambda|$ times $\|x\|$. The x that gives the maximum ratio is the eigenvector for the maximum $|\lambda|$.

Unsymmetric matrices If A is not symmetric, its eigenvalues may not measure its true size. *The norm can be larger than any eigenvalue.* A very unsymmetric example has $\lambda_1 = \lambda_2 = 0$ but its norm is not zero:

$$\|A\| > \lambda_{\max} \quad A = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \qquad \text{has norm} \qquad \|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = 2.$$

The vector $x = (0, 1)$ gives $Ax = (2, 0)$. The ratio of lengths is $2/1$. This is the maximum ratio $\|A\|$, even though x is not an eigenvector.

It is the ***symmetric matrix*** $A^{\mathrm{T}}A$, not the unsymmetric A, that has eigenvector $x = (0, 1)$. The norm is really decided by ***the largest eigenvalue of*** $A^{\mathrm{T}}A$:

The norm of A (symmetric or not) ***is the square root of*** $\lambda_{\max}(A^{\mathrm{T}}A)$:

$$\|A\|^2 = \max_{x \neq 0} \frac{\|Ax\|^2}{\|x\|^2} = \max_{x \neq 0} \frac{x^{\mathrm{T}}A^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} = \lambda_{\max}(A^{\mathrm{T}}A)\,. \tag{4}$$

The unsymmetric example with $\lambda_{\max}(A) = 0$ has $\lambda_{\max}(A^{\mathrm{T}}A) = 4$:

$$A = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \text{ leads to } A^{\mathrm{T}}A = \begin{bmatrix} 0 & 0 \\ 0 & 4 \end{bmatrix} \text{ with } \lambda_{\max} = 4. \text{ So the norm is } \|A\| = \sqrt{4}.$$

For any A Choose x to be the eigenvector of $A^{\mathrm{T}}A$ with largest eigenvalue $\lambda_{\max}$. The ratio in equation (4) is $x^{\mathrm{T}}A^{\mathrm{T}}Ax = x^{\mathrm{T}}(\lambda_{\max})x$ divided by $x^{\mathrm{T}}x$. This is $\lambda_{\max}$.

No x can give a larger ratio. The symmetric matrix $A^{\mathrm{T}}A$ has eigenvalues $\lambda_1, \ldots, \lambda_n$ and orthonormal eigenvectors $q_1, q_2, \ldots, q_n$. Every x is a combination of those vectors. Try this combination in the ratio and remember that $q_i^{\mathrm{T}}q_j = 0$:

$$\frac{x^{\mathrm{T}}A^{\mathrm{T}}Ax}{x^{\mathrm{T}}x} = \frac{(c_1q_1 + \cdots + c_nq_n)^{\mathrm{T}}(c_1\lambda_1q_1 + \cdots + c_n\lambda_nq_n)}{(c_1q_1 + \cdots + c_nq_n)^{\mathrm{T}}(c_1q_1 + \cdots + c_nq_n)} = \frac{c_1^2\lambda_1 + \cdots + c_n^2\lambda_n}{c_1^2 + \cdots + c_n^2}.$$

The maximum ratio $\lambda_{\max}$ is when all c's are zero, except the one that multiplies $\lambda_{\max}$.

Note 1 The ratio in equation (4) is the ***Rayleigh quotient*** for the symmetric matrix $A^{\mathrm{T}}A$. Its maximum is the largest eigenvalue $\lambda_{\max}(A^{\mathrm{T}}A)$. The minimum ratio is $\lambda_{\min}(A^{\mathrm{T}}A)$. If you substitute any vector x into the Rayleigh quotient $x^{\mathrm{T}}A^{\mathrm{T}}Ax/x^{\mathrm{T}}x$, you are guaranteed to get a number between $\lambda_{\min}(A^{\mathrm{T}}A)$ and $\lambda_{\max}(A^{\mathrm{T}}A)$.

Note 2 The norm $\|A\|$ equals the ***largest singular value*** $\sigma_{\max}$ of A. The singular values $\sigma_1, \ldots, \sigma_r$ are the square roots of the positive eigenvalues of $A^{\mathrm{T}}A$. So certainly $\sigma_{\max} = (\lambda_{\max})^{1/2}$. Since U and V are orthogonal in $A = U\Sigma V^{\mathrm{T}}$, the norm is $\|A\| = \sigma_{\max}$.

The Condition Number of A

Section 9.1 showed that roundoff error can be serious. Some systems are sensitive, others are not so sensitive. The sensitivity to error is measured by the ***condition number***. This is the first chapter in the book which intentionally introduces errors. We want to estimate how much they change $\boldsymbol{x}$.

The original equation is $A\boldsymbol{x} = \boldsymbol{b}$. Suppose the right side is changed to $\boldsymbol{b} + \Delta\boldsymbol{b}$ because of roundoff or measurement error. The solution is then changed to $\boldsymbol{x} + \Delta\boldsymbol{x}$. Our goal is to estimate the change $\Delta\boldsymbol{x}$ in the solution from the change $\Delta\boldsymbol{b}$ in the equation. Subtraction gives the *error equation* $A(\Delta\boldsymbol{x}) = \Delta\boldsymbol{b}$:

$$\text{Subtract } A\boldsymbol{x} = \boldsymbol{b} \text{ from } A(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b} \quad \text{to find} \quad A(\Delta\boldsymbol{x}) = \Delta\boldsymbol{b}. \tag{5}$$

The error is $\Delta\boldsymbol{x} = A^{-1}\Delta\boldsymbol{b}$. It is large when A^{-1} is large (then A is nearly singular). The error $\Delta\boldsymbol{x}$ is especially large when $\Delta\boldsymbol{b}$ points in the worst direction—which is amplified most by A^{-1}. ***The worst error has*** $\|\Delta\boldsymbol{x}\| = \|A^{-1}\|\,\|\Delta\boldsymbol{b}\|$.

This error bound $\|A^{-1}\|$ has one serious drawback. If we multiply A by 1000, then A^{-1} is divided by 1000. The matrix looks a thousand times better. But a simple rescaling cannot change the reality of the problem. It is true that $\Delta\boldsymbol{x}$ will be divided by 1000, but so will the exact solution $\boldsymbol{x} = A^{-1}\boldsymbol{b}$. The ***relative error*** $\|\Delta\boldsymbol{x}\|/\|\boldsymbol{x}\|$ will stay the same. It is this relative change in $\boldsymbol{x}$ that should be compared to the relative change in $\boldsymbol{b}$.

Comparing relative errors will now lead to the "condition number" $c = \|A\|\,\|A^{-1}\|$. Multiplying A by 1000 does not change this number, because A^{-1} is divided by 1000 and the condition number c stays the same. It measures the sensitivity of $A\boldsymbol{x} = \boldsymbol{b}$.

The solution error is less than $c = \|A\|\,\|A^{-1}\|$ ***times the problem error:***

Condition number c
$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le c\frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}. \tag{6}$$

If the problem error is ΔA (error in A instead of $\boldsymbol{b}$), still c controls $\Delta\boldsymbol{x}$:

Error ΔA in A
$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x} + \Delta\boldsymbol{x}\|} \le c\frac{\|\Delta A\|}{\|A\|}. \tag{7}$$

Proof The original equation is $\boldsymbol{b} = A\boldsymbol{x}$. The error equation (5) is $\Delta\boldsymbol{x} = A^{-1}\Delta\boldsymbol{b}$. Apply the key property $\|A\boldsymbol{x}\| \le \|A\|\,\|\boldsymbol{x}\|$ of matrix norms:

$$\|\boldsymbol{b}\| \le \|A\|\,\|\boldsymbol{x}\| \qquad \text{and} \qquad \|\Delta\boldsymbol{x}\| \le \|A^{-1}\|\,\|\Delta\boldsymbol{b}\|.$$

Multiply the left sides to get $\|\boldsymbol{b}\|\,\|\Delta\boldsymbol{x}\|$, and multiply the right sides to get $c\|\boldsymbol{x}\|\,\|\Delta\boldsymbol{b}\|$. Divide both sides by $\|\boldsymbol{b}\|\,\|\boldsymbol{x}\|$. The left side is now the relative error $\|\Delta\boldsymbol{x}\|/\|\boldsymbol{x}\|$. The right side is now the upper bound in equation (6).

The same condition number $c = \|A\|\,\|A^{-1}\|$ appears when the error is in the matrix. We have ΔA instead of $\Delta \boldsymbol{b}$ in the error equation:

Subtract $A\boldsymbol{x} = \boldsymbol{b}$ from $(A + \Delta A)(\boldsymbol{x} + \Delta \boldsymbol{x}) = \boldsymbol{b}$ to find $A(\Delta \boldsymbol{x}) = -(\Delta A)(\boldsymbol{x} + \Delta \boldsymbol{x})$.

Multiply the last equation by A^{-1} and take norms to reach equation (7):

$$\|\Delta \boldsymbol{x}\| \le \|A^{-1}\|\,\|\Delta A\|\,\|\boldsymbol{x} + \Delta \boldsymbol{x}\| \quad \text{or} \quad \frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x} + \Delta \boldsymbol{x}\|} \le \|A\|\,\|A^{-1}\|\frac{\|\Delta A\|}{\|A\|}.$$

Conclusion Errors enter in two ways. They begin with an error ΔA or $\Delta \boldsymbol{b}$—a wrong matrix or a wrong $\boldsymbol{b}$. This problem error is amplified (a lot or a little) into the solution error $\Delta \boldsymbol{x}$. That error is bounded, relative to $\boldsymbol{x}$ itself, by the condition number c.

The error $\Delta \boldsymbol{b}$ depends on computer roundoff and on the original measurements of $\boldsymbol{b}$. The error ΔA also depends on the elimination steps. Small pivots tend to produce large errors in L and U. Then $L + \Delta L$ times $U + \Delta U$ equals $A + \Delta A$. When ΔA or the condition number is very large, the error $\Delta \boldsymbol{x}$ can be unacceptable.

Example 3 When A is symmetric, $c = \|A\|\,\|A^{-1}\|$ comes from the eigenvalues:

$$A = \begin{bmatrix} 6 & 0 \\ 0 & 2 \end{bmatrix} \text{ has norm } 6. \qquad A^{-1} = \begin{bmatrix} \frac{1}{6} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \text{ has norm } \tfrac{1}{2}.$$

This A is symmetric positive definite. Its norm is $\lambda_{\max} = 6$. The norm of A^{-1} is $1/\lambda_{\min} = \frac{1}{2}$. Multiplying norms gives the *condition number* $\|A\|\,\|A^{-1}\| = \lambda_{\max}/\lambda_{\min}$:

Condition number for positive definite A $$c = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{6}{2} = 3.$$

Example 4 Keep the same A, with eigenvalues 6 and 2. To make $\boldsymbol{x}$ small, choose $\boldsymbol{b}$ along the first eigenvector $(1, 0)$. To make $\Delta \boldsymbol{x}$ large, choose $\Delta \boldsymbol{b}$ along the second eigenvector $(0, 1)$. Then $\boldsymbol{x} = \frac{1}{6}\boldsymbol{b}$ and $\Delta \boldsymbol{x} = \frac{1}{2}\Delta \boldsymbol{b}$. The ratio $\|\Delta \boldsymbol{x}\|/\|\boldsymbol{x}\|$ is exactly $c = 3$ times the ratio $\|\Delta \boldsymbol{b}\|/\|\boldsymbol{b}\|$.

This shows that the worst error allowed by the condition number $\|A\|\,\|A^{-1}\|$ can actually happen. Here is a useful rule of thumb, experimentally verified for Gaussian elimination: *The computer can lose* $\log c$ *decimal places to roundoff error.*

Problem Set 9.2

1 Find the norms $\|A\| = \lambda_{\max}$ and condition numbers $c = \lambda_{\max}/\lambda_{\min}$ of these positive definite matrices:

$$\begin{bmatrix} .5 & 0 \\ 0 & 2 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}.$$

2 Find the norms and condition numbers from the square roots of $\lambda_{\max}(A^{\mathrm{T}}A)$ and $\lambda_{\min}(A^{\mathrm{T}}A)$. Without positive definiteness in A, we go to $A^{\mathrm{T}}A$!

$$\begin{bmatrix} -2 & 0 \\ 0 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}.$$

3 Explain these two inequalities from the definitions (3) of $\|A\|$ and $\|B\|$:

$$\|ABx\| \le \|A\|\,\|Bx\| \le \|A\|\,\|B\|\,\|x\|.$$

From the ratio of $\|ABx\|$ to $\|x\|$, deduce that $\|AB\| \le \|A\|\,\|B\|$. This is the key to using matrix norms. The norm of A^n is never larger than $\|A\|^n$.

4 Use $\|AA^{-1}\| \le \|A\|\,\|A^{-1}\|$ to prove that the condition number is at least 1.

5 Why is I the only symmetric positive definite matrix that has $\lambda_{\max} = \lambda_{\min} = 1$? Then the only other matrices with $\|A\| = 1$ and $\|A^{-1}\| = 1$ must have $A^{\mathrm{T}}A = I$. Those are _____ matrices: perfectly conditioned.

6 Orthogonal matrices have norm $\|Q\| = 1$. If $A = QR$ show that $\|A\| \le \|R\|$ and also $\|R\| \le \|A\|$. Then $\|A\| = \|Q\|\,\|R\|$. Find an example of $A = LU$ with $\|A\| < \|L\|\|U\|$.

7 (a) Which famous inequality gives $\|(A+B)x\| \le \|Ax\| + \|Bx\|$ for every x?

(b) Why does the definition (3) of matrix norms lead to $\|A+B\| \le \|A\| + \|B\|$?

8 Show that if λ is any eigenvalue of A, then $|\lambda| \le \|A\|$. Start from $Ax = \lambda x$.

9 The "*spectral radius*" $\rho(A) = |\lambda_{\max}|$ is the largest absolute value of the eigenvalues. Show with 2 by 2 examples that $\rho(A+B) \le \rho(A) + \rho(B)$ and $\rho(AB) \le \rho(A)\rho(B)$ can both be *false*. The spectral radius is not acceptable as a norm.

10 (a) Explain why A and A^{-1} have the same condition number.

(b) Explain why A and A^{T} have the same norm, based on $\lambda(A^{\mathrm{T}}A)$ and $\lambda(AA^{\mathrm{T}})$.

11 Estimate the condition number of the ill-conditioned matrix $A = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix}$.

12 Why is the determinant of A no good as a norm? Why is it no good as a condition number?

13 (Suggested by C. Moler and C. Van Loan.) Compute $b - Ay$ and $b - Az$ when

$$b = \begin{bmatrix} .217 \\ .254 \end{bmatrix} \quad A = \begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \quad y = \begin{bmatrix} .341 \\ -.087 \end{bmatrix} \quad z = \begin{bmatrix} .999 \\ -1.0 \end{bmatrix}.$$

Is y closer than z to solving $Ax = b$? Answer in two ways: Compare the *residual* $b - Ay$ to $b - Az$. Then compare y and z to the true $x = (1, -1)$. Both answers can be right. Sometimes we want a small residual, sometimes a small Δx.

14 (a) Compute the determinant of A in Problem 13. Compute A^{-1}.

(b) If possible compute $\|A\|$ and $\|A^{-1}\|$ and show that $c > 10^6$.

Problems 15–19 are about vector norms other than the usual $\|x\| = \sqrt{x \cdot x}$.

15 The "ℓ^1 norm" and the "ℓ^∞ norm" of $x = (x_1, \ldots, x_n)$ are

$$\|x\|_1 = |x_1| + \cdots + |x_n| \quad \text{and} \quad \|x\|_\infty = \max_{1 \le i \le n} |x_i|.$$

Compute the norms $\|x\|$ and $\|x\|_1$ and $\|x\|_\infty$ of these two vectors in $\mathbf{R}^5$:

$$x = (1, 1, 1, 1, 1) \qquad x = (.1, .7, .3, .4, .5).$$

16 Prove that $\|x\|_\infty \le \|x\| \le \|x\|_1$. Show from the Schwarz inequality that the ratios $\|x\|/\|x\|_\infty$ and $\|x\|_1/\|x\|$ are never larger than $\sqrt{n}$. Which vector $(x_1, \ldots, x_n)$ gives ratios equal to $\sqrt{n}$?

17 All vector norms must satisfy the *triangle inequality*. Prove that

$$\|x + y\|_\infty \le \|x\|_\infty + \|y\|_\infty \qquad \text{and} \qquad \|x + y\|_1 \le \|x\|_1 + \|y\|_1.$$

18 Vector norms must also satisfy $\|cx\| = |c|\,\|x\|$. The norm must be positive except when $x = \mathbf{0}$. Which of these are norms for vectors (x_1, x_2) in $\mathbf{R}^2$?

$$\|x\|_A = |x_1| + 2|x_2| \qquad \|x\|_B = \min(|x_1|, |x_2|)$$

$$\|x\|_C = \|x\| + \|x\|_\infty \qquad \|x\|_D = \|Ax\| \quad \text{(this answer depends on } A\text{)}.$$

Challenge Problems

19 Show that $x^{\mathrm{T}} y \le \|x\|_1 \, \|y\|_\infty$ by choosing components $y_i = \pm 1$ to make $x^{\mathrm{T}} y$ as large as possible.

20 The eigenvalues of the $-1, 2, -1$ difference matrix K are $\lambda = 2 - 2\cos(j\pi/n+1)$. Estimate $\lambda_{\min}$ and $\lambda_{\max}$ and $c = \mathbf{cond}(K) = \lambda_{\max}/\lambda_{\min}$ as n increases: $c \approx Cn^2$ with what constant C?

Test this estimate with $\mathbf{eig}(K)$ and $\mathbf{cond}(K)$ for $n = 10, 100, 1000$.

21 For unsymmetric matrices, the spectral radius $\rho = \max|\lambda_i|$ is not a norm (Problem 9). But still $\|A^n\|$ grows or decays like ρ^n for large n. Compare those numbers for $A =$ [1 1; 0 1.1] using the command **norm**.

In particular $A^n \to 0$ when $\rho < 1$. This is the key to Section 9.3 with $A = S^{-1}T$.

9.3 Iterative Methods and Preconditioners

Up to now, our approach to $Ax = b$ has been direct. We accepted A as it came. We attacked it by elimination with row exchanges. This section is about ***iterative methods***, which replace A by a simpler matrix 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 $S - T$.

$$\textbf{Rewrite } Ax = b \qquad Sx = Tx + b. \tag{1}$$

The novelty is to solve (1) iteratively. Each guess x_k leads to the next x_{k+1}:

$$\textbf{Pure iteration} \qquad Sx_{k+1} = Tx_k + b. \tag{2}$$

Start with any x_0. Then solve $Sx_1 = Tx_0 + b$. Continue to the second iteration $Sx_2 = Tx_1 + b$. A hundred iterations are very common—often more. Stop when (and if!) the new vector x_{k+1} is sufficiently close to x_k—or when the **residual** $r_k = b - Ax_k$ is near zero. We choose the stopping test. Our hope is to get near the true solution, more quickly than by elimination. When the sequence x_k converges, its limit $x = x_\infty$ does solve equation (1). The proof is to let $k \to \infty$ in equation (2).

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 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 $x - x_k$ (this is the error e_k) should go quickly to zero. Subtracting equation (2) from (1) cancels b, and it leaves the ***equation for the error*** e_k:

$$\textbf{Error equation} \qquad Se_{k+1} = Te_k \text{ which means } e_{k+1} = S^{-1}Te_k. \tag{3}$$

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 $Ax = b$. 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 popular choices:

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*)

SOR S = combination of Jacobi and Gauss-Seidel (*successive overrelaxation*)

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 **J** and **GS** and **SOR**, 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 $e_{k+1} = S^{-1}Te_k$. Every iteration step multiplies the error by the same matrix $B = S^{-1}T$. The error after k steps is $e_k = B^k e_0$. ***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: $\rho = \max |\lambda(B)|$.

The test for convergence is $|\lambda|_{\max} < 1$. Real eigenvalues must lie between -1 and 1. Complex eigenvalues $\lambda = a + ib$ must have $|\lambda|^2 = a^2 + b^2 < 1$. (Chapter 10 will discuss complex numbers.) The spectral radius "*rho*" is the largest distance from 0 to the eigenvalues $\lambda_1, \ldots, \lambda_n$ of $B = S^{-1}T$. This is $\rho = |\lambda|_{\max}$.

To see why $|\lambda|_{\max} < 1$ is necessary, suppose the starting error e_0 happens to be an eigenvector of B. After one step the error is $Be_0 = \lambda e_0$. After k steps the error is $B^k e_0 = \lambda^k e_0$. If we start with an eigenvector, we continue with that eigenvector—and it grows or decays with the powers λ^k. *This factor λ^k goes to zero when* $|\lambda| < 1$. Since this condition is required of every eigenvalue, we need $\rho = |\lambda|_{\max} < 1$.

To see why $|\lambda|_{\max} < 1$ is sufficient for the error to approach zero, suppose e_0 is a combination of eigenvectors:

$$e_0 = c_1 x_1 + \cdots + c_n x_n \quad \text{leads to} \quad e_k = c_1(\lambda_1)^k x_1 + \cdots + c_n(\lambda_n)^k x_n. \tag{4}$$

This is the point of eigenvectors! They grow independently, each one controlled by its eigenvalue. When we multiply by B, the eigenvector x_i is multiplied by λ_i. If all $|\lambda_i| < 1$ then equation (4) ensures that 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 There is a technical difficulty when B does not have n independent eigenvectors. (To produce this effect in B', change .5 to .6.) The starting error e_0 may not be a combination of eigenvectors—there are too few for a basis. Then diagonalization is impossible and equation (4) is not correct. We turn to the *Jordan form* when eigenvectors are missing:

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

Section 6.6 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 all Jordan blocks. A block of size $S + 1$ has $k^S \lambda^{k-S}$ in J^k, which also approaches zero when $|\lambda| < 1$.

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. Watch for that number $|\lambda|_{\max}$.

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

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

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 $-1, -1/4, -1/16$. The error is multiplied by $\frac{1}{4}$ every two steps. The components 0, 3/2, 15/8 have errors 2, $\frac{1}{2}$, $\frac{1}{8}$. Those also drop by 4 in each two steps. *The error equation is* $Se_{k+1} = Te_k$:

$$\textbf{Error equation} \qquad \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} e_{k+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} e_k \quad \text{or} \quad e_{k+1} = \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix} e_k. \tag{7}$$

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 and $S^{-1}T$ to be small.

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:

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

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.

Starting from $(0, 0)$, the exact answer $(2, 0)$ is reached in one step. That is an accident I did not expect. Test the iteration 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 $\rho_{GS} = (\rho_J)^2$.

This double speed is true for every positive definite tridiagonal matrix. Anything is possible when A is strongly nonsymmetric—Jacobi is sometimes better, and both methods might fail. Our example is small and 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 explain 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 $Ax = b$ as $\omega Ax = \omega b$. 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 $S - \omega A$:

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

This looks more complicated to us, but the computer goes as fast as ever. Each new u_{k+1} from the first equation is used immediately to find v_{k+1} in the second equation. This is like Gauss-Seidel, with an adjustable number ω. The key matrix is $S^{-1}T$:

$$\textbf{SOR iteration matrix} \quad S^{-1}T = \begin{bmatrix} 1 - \omega & \frac{1}{2}\omega \\ \frac{1}{2}\omega(1 - \omega) & 1 - \omega + \frac{1}{4}\omega^2 \end{bmatrix}. \tag{10}$$

The determinant is $(1 - \omega)^2$. At the best ω, both eigenvalues turn out to equal $7 - 4\sqrt{3}$, which is close to $(\frac{1}{4})^2$. Therefore **SOR** is twice as fast as Gauss-Seidel in this example. In other examples **SOR** can converge ten or a hundred times as fast.

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 eigenvalue 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 is very fast (exact LU). Iterations are intended for large sparse matrices—when a high percentage of the entries are zero. The not good zeros are inside the band, which is wide. They become nonzero in the exact L and U, which is why elimination becomes expensive.

We mention one more splitting. The idea of "***incomplete*** LU" 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 $\qquad L_0U_0\boldsymbol{x}_{k+1} = (L_0U_0 - A)\boldsymbol{x}_k + \boldsymbol{b}.$

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 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 (often coming from coarser grids and doubled step-sizes Δx and Δy). Each iteration will be cheaper and convergence will be faster. Then interpolate between the values computed on the coarse grid to get a quick and close head-start on the full-size problem. Multigrid might go 4 levels down and back.

Conjugate gradients An ordinary iteration like $x_{k+1} = x_k - Ax_k + b$ involves multiplication by A at each step. If A is sparse, this is not too expensive: Ax_k is what we are willing to do. It adds one more basis vector to the growing "Krylov spaces" that contain our approximations. But x_{k+1} is **not the best combination** of $x_0, Ax_0, \ldots, A^k x_0$. The ordinary iterations are simple but far from optimal.

The conjugate gradient method chooses **the best combination** 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 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 x_{k-1} to x_k. We compute that new approximation x_k, the new residual $r_k = b - Ax_k$, and the new search direction d_k to look for the next 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 $Ax = b$. The preconditioned equation is $S^{-1}Ax = S^{-1}b$. 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 many zeros in A. It is not easy to outperform Gauss.

Iterative Methods for Eigenvalues

We move from $Ax = b$ to $Ax = \lambda x$. 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 $(\lambda_i)^k$:

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

As the power method continues, ***the largest eigenvalue begins to dominate.*** The vectors $\boldsymbol{u}_k$ point toward that dominant eigenvector. We saw this for Markov matrices in Chapter 8:

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

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 $A\boldsymbol{u}_k = \boldsymbol{u}_{k-1}$:

Inverse power method
$$\boldsymbol{u}_k = A^{-k}\boldsymbol{u}_0 = \frac{c_1\boldsymbol{x}_1}{(\lambda_1)^k} + \cdots + \frac{c_n\boldsymbol{x}_n}{(\lambda_n)^k}. \tag{12}$$

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 $\lambda_{\min}$ 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. Fifty 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.) At present the ***QR method*** is the leader in eigenvalue computations and we describe it briefly.

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* $A_1 = Q^{-1}AQ$:

$$\textbf{\textit{A}}_1 = \textbf{\textit{RQ}} \textbf{ has the same } \lambda \qquad QRx = \lambda x \quad \text{gives} \quad RQ(Q^{-1}x) = \lambda(Q^{-1}x). \tag{13}$$

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. Often 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:

$$\text{Factor } A_k - c_kI \text{ into } Q_kR_k. \text{ The next matrix is } A_{k+1} = R_kQ_k + c_kI.$$

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

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

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 (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 large sparse matrices I would look to ARPACK. Problems 27–29 describe the Arnoldi iteration that orthogonalizes the basis—each step has only three terms when A is symmetric. The matrix becomes tridiagonal and still orthogonally similar to the original A: a wonderful start for computing eigenvalues.

Problem Set 9.3

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

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

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 $x_{k+1} = (I - A)x_k + b$ does not converge for $A = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$.

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 $Ax = 0$ show that $S^{-1}Tx = x$. 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:

$$Sx_{k+1} = Tx_k + b \quad \text{is} \quad \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} x_{k+1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} x_k + b.$$

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

$$\begin{bmatrix} 3 & 0 \\ -1 & 3 \end{bmatrix} x_{k+1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_k + b.$$

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

8 For any 2 by 2 matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ 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 The best ω produces two equal eigenvalues for $S^{-1}T$ in the **SOR** method. Those eigenvalues are $\omega - 1$ because the determinant is $(\omega - 1)^2$. Set the trace in equation (10) equal to $(\omega - 1) + (\omega - 1)$ and find this optimal ω.

10 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 $b = (1, 0, \ldots, 0)$.

11 The Gauss-Seidel iteration at component i uses earlier parts of x^{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 x is correct? How does the formula change for Jacobi's method? For **SOR** insert ω outside the parentheses.

12 The **SOR** splitting matrix S is the same as for Gauss-Seidel except that the diagonal is divided by ω. Write a program for **SOR** on an n by n matrix. Apply it with $\omega = 1$, 1.4, 1.8, 2.2 when A is the $-1, 2, -1$ matrix of order $n = 10$.

13 Divide equation (11) 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.*

14 The Markov matrix $A = \begin{bmatrix} .9 & .3 \\ .1 & .7 \end{bmatrix}$ has $\lambda = 1$ and .6, and the power method $\boldsymbol{u}_k = A^k \boldsymbol{u}_0$ converges to $\begin{bmatrix} .75 \\ .25 \end{bmatrix}$. 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 $.6^k$)?

15 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 $\lambda_1/\lambda_2 \approx 1/4$.

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

17 In Problem 11 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?

18 In the QR method for eigenvalues, show that the 2, 1 entry drops from $\sin\theta$ in $A = QR$ to $-\sin^3\theta$ in RQ. (*Compute R and RQ.*) This "cubic convergence" makes the method a success:

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

19 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.

20 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 is closer to triangular).

21 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.

22 The equation in Problem 21 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.

23 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.

Questions 24–26 are about quick ways to estimate the location of the eigenvalues.

24 If the sum of $|a_{ij}|$ along every row is less than 1, explain this proof that $|\lambda| < 1$. Suppose $Ax = \lambda x$ and $|x_i|$ is larger than the other components of x. Then $|\Sigma a_{ij}x_j|$ is less than $|x_i|$. That means $|\lambda x_i| < |x_i|$ so $|\lambda| < 1$.

(Gershgorin circles) Every eigenvalue of A is in one or more of n circles. Each circle is centered at a diagonal entry a_{ii} with radius $r_i = \Sigma_{j \neq i}|a_{ij}|$.

This follows from $(\lambda - a_{ii})x_i = \Sigma_{j \neq i} a_{ij}x_j$. If $|x_i|$ is larger than the other components of x, this sum is at most $r_i|x_i|$. Dividing by $|x_i|$ leaves $|\lambda - a_{ii}| \leq r_i$.

25 What bound on $|\lambda|_{\max}$ does Problem 24 give for these matrices? What are the three Gershgorin circles that contain all the eigenvalues? Those circles show immediately that K is at least positive semidefinite (*actually definite*) and A has $\lambda_{\max} = 1$.

$$A = \begin{bmatrix} .3 & .5 & .2 \\ .3 & .4 & .3 \\ .4 & .1 & .5 \end{bmatrix} \qquad K = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

26 These matrices are diagonally dominant because each $a_{ii} > r_i$ = absolute sum along the rest of row i. From the Gershgorin circles containing all λ's, show that diagonally dominant matrices are invertible.

$$A = \begin{bmatrix} 1 & .3 & .4 \\ .3 & 1 & .5 \\ .4 & .5 & 1 \end{bmatrix} \qquad A = \begin{bmatrix} 4 & 2 & 1 \\ 1 & 3 & 1 \\ 2 & 2 & 5 \end{bmatrix}.$$

Problems 27–30 present two fundamental iterations. Each step involves Aq or Ad.

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 $x = A^{-1}b$.

Arnoldi Iteration	**Conjugate Gradient Iteration for Positive Definite** A	
$q_1 = b/\|b\|$	$x_0 = 0, r_0 = b, d_0 = r_0$	
for $n = 1$ **to** $N-1$	**for** $n = 1$ **to** N	
$v = Aq_n$	$\alpha_n = (r_{n-1}^{\mathrm{T}} r_{n-1})/(d_{n-1}^{\mathrm{T}} A d_{n-1})$	step length x_{n-1} to x_n
for $j = 1$ **to** n	$x_n = x_{n-1} + \alpha_n d_{n-1}$	approximate solution
$h_{jn} = q_j^{\mathrm{T}} v$	$r_n = r_{n-1} - \alpha_n A d_{n-1}$	new residual $b - Ax_n$
$v = v - h_{jn} q_j$	$\beta_n = (r_n^{\mathrm{T}} r_n)/(r_{n-1}^{\mathrm{T}} r_{n-1})$	improvement this step
$h_{n+1,n} = \|v\|$	$d_n = r_n + \beta_n d_{n-1}$	next search direction
$q_{n+1} = v/h_{n+1,n}$	*% Notice: only 1 matrix-vector multiplication* Aq *and* Ad	

For conjugate gradients, the residuals r_n are orthogonal and the search directions are A-orthogonal: all $d_j^{\mathrm{T}} A d_k = 0$. The iteration solves $Ax = b$ by minimizing the error $e^{\mathrm{T}} Ae$ over all vectors in the Krylov subspace. It is a fantastic algorithm.

27 For the diagonal matrix $A = \mathrm{diag}([1 \quad 2 \quad 3 \quad 4])$ and the vector $b = (1, 1, 1, 1)$, go through one Arnoldi step to find the orthonormal vectors q_1 and q_2.

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

$$AQ = \begin{bmatrix} Aq_1 & \cdots & Aq_N \end{bmatrix} = \begin{bmatrix} q_1 & \cdots & q_N \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1N} \\ h_{21} & h_{22} & \cdot & h_{2N} \\ 0 & h_{32} & \cdot & \cdot \\ 0 & 0 & \cdot & h_{NN} \end{bmatrix} = QH$$

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

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

$$\textbf{Three terms only} \qquad q_{j+1} = (Aq_j - h_{j,j}q_j - h_{j-1,j}q_{j-1})/h_{j+1,j}$$

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.

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