Chapter 11

Numerical Linear Algebra

1 The goals of numerical linear algebra are **speed** and **accuracy** and **stability** : $n > 10^3$ or 10^6.

2 Matrices can be full or sparse or banded or structured : special algorithms for each.

3 Accuracy of elimination is controlled by the **condition number** $||\boldsymbol{A}||\ ||\boldsymbol{A^{-1}}||$.

4 Gram-Schmidt is often computed by using **Householder reflections** $H = I - 2\boldsymbol{u}\boldsymbol{u}^{\mathrm{T}}$ to find Q.

5 Eigenvalues use $\boldsymbol{QR}$ **iterations** $\boldsymbol{A_0} = Q_0R_0 \to R_0Q_0 = \boldsymbol{A_1} = Q_1R_1 \to \to \boldsymbol{A_n}$.

6 **Shifted** $\boldsymbol{QR}$ is even better : Shift to $A_k - c_kI = Q_kR_k$, shift back $A_{k+1} = R_kQ_k + c_kI$.

7 Iteration $S\boldsymbol{x}_{k+1} = \boldsymbol{b} - T\boldsymbol{x}_k$ solves $(S+T)\,\boldsymbol{x} = \boldsymbol{b}$ if all eigenvalues of $S^{-1}T$ have $|\lambda| < 1$.

8 Iterative methods often use **preconditioners** P. Change $A\boldsymbol{x} = \boldsymbol{b}$ to $PA\boldsymbol{x} = P\boldsymbol{b}$ with $PA \approx I$.

9 **Conjugate gradients** and **GMRES** are Krylov methods; see Trefethen-Bau (and other texts).

11.1 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 available pivot* in each new column. We will see why this pivoting 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 produce a narrow band of nonzeros*. Zeros inside the band "fill in" during elimination—those zeros are destroyed and don't save computing time.

Section 11.2 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 11.3 describes how to solve $A\boldsymbol{x} = \boldsymbol{b}$ by ***iterations***. Instead of direct elimination, the computer solves an easier equation many times. Each answer $\boldsymbol{x}_k$ leads to the next guess $\boldsymbol{x}_{k+1}$. For good iterations (the **conjugate gradient method** is extremely good), the $\boldsymbol{x}_k$ converge quickly to $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

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 $A\boldsymbol{x} = \boldsymbol{b}$: computer speed is tested by linear algebra.

That machine was shut down in 2013! The TOP500 project ranks the 500 most powerful computer systems in the world. As I write this page in October 2015, the first four are from NUDT in China, Cray and IBM in the US, and Fujitsu in Japan. They all use a LINUX-based system. And all vector processors have fallen out of the top 500.

Looking ahead, the Summit is expected to take first place with 150-300 petaflops. President Obama has just ordered the development of an exascale system (1000 petaflops). Up to now we are following Moore's Law of doubling every 14 months.

The LAPACK 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. And graphics processing units (GPU's) are now almost required for success. The market for video games dwarfs scientific computing and led to astonishing acceleration in the chips.

Before 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 update on **www.lanl.gov/roadrunner**.

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

Level 1 Linear combinations of vectors $a\boldsymbol{u} + \boldsymbol{v}$: $O(n)$ work

Level 2 Matrix-vector multiplications $A\boldsymbol{u} + \boldsymbol{v}$: $O(n^2)$ work

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

Level 1 is an 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:

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

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 $v = 1$ into the equation for u:

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

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 small 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 3 places:

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

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***. Here is our strategy when we reach and search column k for the best available pivot:

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 $A\boldsymbol{x} = \boldsymbol{b}$***—by elimination with partial pivoting***. Compared with the theoretical description—***find A^{-1} and multiply $A^{-1}\boldsymbol{b}$***—the details took time. But in computer time, elimination is much faster. I believe that elimination is also the best approach to the algebra of row spaces and nullspaces.

Operation Counts: Full Matrices

Here is a practical question about cost. *How many separate operations are needed to solve* $A\boldsymbol{x} = \boldsymbol{b}$ *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 ($1^2-1=0$); 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:

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

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

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

That means $\frac{1}{3}n^3$ multiplications and 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*** $\boldsymbol{n^2}$ ***operations***. Down and back up we are solving two triangular systems, $L\boldsymbol{c} = \boldsymbol{b}$ forward and $U\boldsymbol{x} = \boldsymbol{c}$ backward. In back substitution, the last unknown needs only division by the last pivot. The equation above it needs two operations—substituting $\boldsymbol{x}_n$ 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 $\frac{1}{3}n^3$ multiply-subtracts, compared to n^3 for A^{-1}.

2 If A is *banded* so are L and U: by comparison 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 (and for band matrices in Figure 11.1):

Tridiagonal
Bidiagonal
times
bidiagonal

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

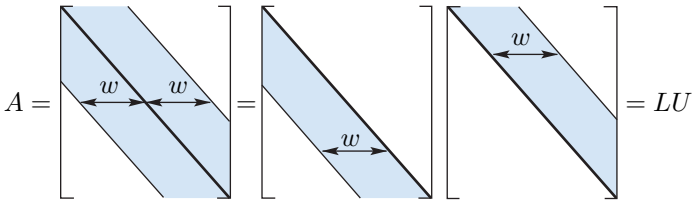


Figure 11.1: $A = L\,U$ for a band matrix. Good zeros in A *stay zero* in L and U.

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 of $A\boldsymbol{x} = \boldsymbol{b}$, 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 $L\boldsymbol{c} = \boldsymbol{b}$ and $U\boldsymbol{x} = \boldsymbol{c}$ 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.*

In $AA^{-1} = I$, the jth column of A^{-1} solves $A\boldsymbol{x}_j = j$th 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:

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

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.

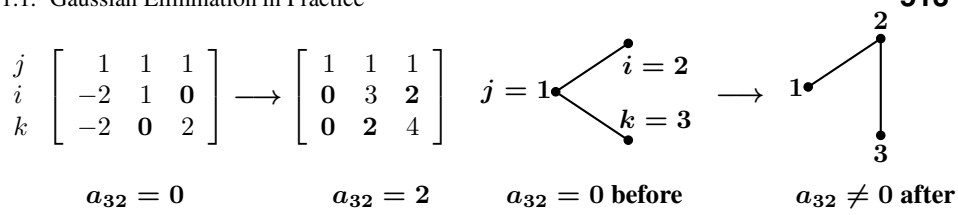
Caution To judge a numerical algorithm, it is **not enough** to count the operations. Beyond "flop counting" is a study of stability (Householder wins) and the flow of data.

Reordering Sparse Matrices

For band matrices with constant width w, the row ordering is optimal. 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 "**a**pproximate **mi**nimum **d**egree" 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 find the absolute minimum of nonzeros in L and U is an NP-hard problem, much too expensive, and **amd** is a good compromise.

Fill-in is famous when each point on a square grid is connected to its four nearest neighbors. It is impossible to number all the gridpoints 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.



We only need the *positions* of the nonzeros, not their exact values. Think of the graph of nonzeros: *Node i is connected to node j if $a_{ij} \neq 0$.* Watch to see how elimination can create nonzeros (new edges), which we are trying to avoid.

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 goal of **colamd** and **symamd** is a better ordering (permutation P) that reduces fill-in for AP and $P^{\mathrm{T}}AP$—*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 from elimination, 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*** $I - 2\boldsymbol{u}\boldsymbol{u}^{\mathrm{T}}$ 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 in the 1-2 plane

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

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 $\cos\theta$:

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

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 L and Q are not the same.

Householder reflections are faster than rotations 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

$$H_1 = I - 2\boldsymbol{u}_1\boldsymbol{u}_1^{\mathrm{T}} \qquad H_1\,\boldsymbol{a}_1 = \begin{bmatrix} \|\boldsymbol{a}_1\| \\ 0 \\ \cdot \\ 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -\|\boldsymbol{a}_1\| \\ 0 \\ \cdot \\ 0 \end{bmatrix} = \boldsymbol{r}_1. \tag{4}$$

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, we have $n-k$ available choices in the unit vector $\boldsymbol{u}_k$. This leads to $n-k$ zeros in $\boldsymbol{r}_k$. *We just store the $\boldsymbol{u}$'s and $\boldsymbol{r}$'s to know the final Q and R:*

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

This is how LAPACK improves on 19th century Gram-Schmidt. Q is *exactly* orthogonal.

Section 11.3 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 $A_1, A_2, A_3, \dots$ 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 11.1

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

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 $PA = LU$:

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

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:

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

(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 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

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

15 Create the 10 by 10 second difference matrix $K = \textbf{toeplitz}([2\ -1\ \textbf{zeros}(1,8)])$. Permute rows and columns randomly by $KK = K(\textbf{randperm}(10), \textbf{randperm}(10))$. Factor by $[L,U] = \textbf{lu}(K)$ and $[LL,UU] = \textbf{lu}(KK)$, and count nonzeros by $\textbf{nnz}(L)$ and $\textbf{nnz}(LL)$. 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 $1, \dots, 10$ to $1, 3, 5, 7, 9, 2, 4, 6, 8, 10$:

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

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 $B = \textbf{sparse}(A)$, after which $\textbf{find}(B)$ will list the nonzero entries and $\boldsymbol{\ell}\textbf{u}(B)$ 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:

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

When those equations are shown by nearly parallel long sticks, a small shake gives a big jump in the crossing point (x, y). Errors $\boldsymbol{e}$ and $\boldsymbol{E}$ are amplified by 10000.