Lecture 20. Gaussian Elimination

Gaussian elimination is undoubtedly familiar to the reader. It is the simplest way to solve linear systems of equations by hand, and also the standard method for solving them on computers. We first describe Gaussian elimination in its pure form, and then, in the next lecture, add the feature of row pivoting that is essential to stability.

LU Factorization

Gaussian elimination transforms a full linear system into an upper-triangular one by applying simple linear transformations on the left. In this respect it is analogous to Householder triangularization for computing QR factorizations. The difference is that the transformations applied in Gaussian elimination are not unitary.

Let $A \in \mathbb{C}^{m \times m}$ be a square matrix. (The algorithm can also be applied to rectangular matrices, but as this is rarely done in practice, we shall confine our attention to the square case.) The idea is to transform A into an $m \times m$ upper-triangular matrix U by introducing zeros below the diagonal, first in column 1, then in column 2, and so on—just as in Householder triangularization. This is done by subtracting multiples of each row from subsequent rows. This "elimination" process is equivalent to multiplying A by a sequence of lower-triangular matrices L_k on the left:

$$\underbrace{L_{m-1} \cdots L_2 L_1}_{L^{-1}} A = U. \tag{20.1}$$

Setting $L = L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1}$ gives $A = LU$. Thus we obtain an *LU factorization* of A,

$$A = LU, \tag{20.2}$$

where U is upper-triangular and L is lower-triangular. It turns out that L is *unit lower-triangular*, which means that all of its diagonal entries are equal to 1.

For example, suppose we start with a 4×4 matrix. The algorithm proceeds in three steps (compare (10.1)):

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}} \xrightarrow{L_1} \underset{L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} & \mathbf{\times} \\ \mathbf{0} & \mathbf{\times} & \mathbf{\times} & \mathbf{\times} \end{bmatrix}} \xrightarrow{L_2} \underset{L_2L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & \mathbf{0} & \mathbf{\times} & \mathbf{\times} \\ & \mathbf{0} & \mathbf{\times} & \mathbf{\times} \end{bmatrix}} \xrightarrow{L_3} \underset{L_3L_2L_1A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & \mathbf{0} & \mathbf{\times} \end{bmatrix}}.$$

(As in Lecture 10, boldfacing indicates entries just operated upon, and blank entries are zero.) The kth transformation L_k introduces zeros below the diagonal in column k by subtracting multiples of row k from rows $k+1, \ldots, m$. Since the first $k-1$ entries of row k are already zero, this operation does not destroy any zeros previously introduced.

Gaussian elimination thus augments our taxonomy of algorithms for factoring a matrix:

Gram–Schmidt: $A = QR$ by triangular orthogonalization,
Householder: $A = QR$ by orthogonal triangularization,
Gaussian elimination: $A = LU$ by triangular triangularization.

Example

In discussing the details, it will help to have a numerical example on the table. Suppose we start with the 4×4 matrix

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}. \tag{20.3}$$

(The entries of A are anything but random; they were chosen to give a simple LU factorization.) The first step of Gaussian elimination looks like this:

$$L_1A = \begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{bmatrix}.$$

In words, we have subtracted twice the first row from the second, four times the first row from the third, and three times the first row from the fourth. The second step looks like this:

$$L_2L_1A = \begin{bmatrix} 1 & & & \\ & 1 & & \\ -3 & 1 & & \\ -4 & & 1 & \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{bmatrix}.$$

This time we have subtracted three times the second row from the third and four times the second row from the fourth. Finally, in the third step we subtract the third row from the fourth:

$$L_3L_2L_1A = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & -1 & 1 & \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{bmatrix} = U.$$

Now, to exhibit the full factorization $A = LU$, we need to compute the product $L = L_1^{-1}L_2^{-1}L_3^{-1}$. Perhaps surprisingly, this turns out to be a triviality. The inverse of L_1 is just L_1 itself, but with each entry below the diagonal negated:

$$\begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & & 1 & \\ 3 & & & 1 \end{bmatrix}. \tag{20.4}$$

Similarly, the inverses of L_2 and L_3 are obtained by negating their subdiagonal entries. Finally, the product $L_1^{-1}L_2^{-1}L_3^{-1}$ is just the unit lower-triangular matrix with the nonzero subdiagonal entries of L_1^{-1}, L_2^{-1}, and L_3^{-1} inserted in the appropriate places. All together, we have

$$\underset{A}{\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}} = \underset{L}{\begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{bmatrix}} \underset{U}{\begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{bmatrix}}. \tag{20.5}$$

General Formulas and Two Strokes of Luck

Here are the general formulas for an $m \times m$ matrix. Suppose x_k denotes the kth column of the matrix at the beginning of step k. Then the transformation

L_k must be chosen so that

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1,k} \\ \vdots \\ x_{mk} \end{bmatrix} \quad \overset{L_k}{\longrightarrow} \quad L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

To do this we wish to subtract ℓ_{jk} times row k from row j, where ℓ_{jk} is the *multiplier*

$$\ell_{jk} = \frac{x_{jk}}{x_{kk}} \qquad (k < j \le m). \tag{20.6}$$

The matrix L_k takes the form

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -\ell_{mk} & & 1 \end{bmatrix},$$

with the nonzero subdiagonal entries situated in column k. This is analogous to (10.2) for Householder triangularization.

In the numerical example above, we noted two strokes of luck: that L_k can be inverted by negating its subdiagonal entries (20.4), and that L can be formed by collecting the entries ℓ_{jk} in the appropriate places (20.5). We can explain these bits of good fortune as follows. Let us define

$$\ell_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \ell_{k+1,k} \\ \vdots \\ \ell_{m,k} \end{bmatrix}.$$

Then L_k can be written $L_k = I - \ell_k e_k^*$, where e_k is, as usual, the column vector with 1 in position k and 0 elsewhere. The sparsity pattern of ℓ_k implies $e_k^* \ell_k = 0$, and therefore $(I - \ell_k e_k^*)(I + \ell_k e_k^*) = I - \ell_k e_k^* \ell_k e_k^* = I$. In other words, the inverse of L_k is $I + \ell_k e_k^*$, as in (20.4).

For the second stroke of luck we argue as follows. Consider, for example, the product $L_k^{-1} L_{k+1}^{-1}$. From the sparsity pattern of ℓ_{k+1}, we have $e_k^* \ell_{k+1} = 0$, and therefore

$$L_k^{-1} L_{k+1}^{-1} = (I + \ell_k e_k^*)(I + \ell_{k+1} e_{k+1}^*) = I + \ell_k e_k^* + \ell_{k+1} e_{k+1}^*.$$

Thus $L_k^{-1}L_{k+1}^{-1}$ is just the unit lower-triangular matrix with the entries of both L_k^{-1} and L_{k+1}^{-1} inserted in their usual places below the diagonal. When we take the product of all of these matrices to form L, we have the same convenient property everywhere below the diagonal:

$$L = L_1^{-1}L_2^{-1}\cdots L_{m-1}^{-1} = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{m1} & \ell_{m2} & \cdots & \ell_{m,m-1} & 1 \end{bmatrix}. \tag{20.7}$$

Though we did not mention it in Lecture 8, the sparsity considerations that led to (20.7) also appeared in the interpretation (8.10) of the modified Gram–Schmidt process as a succession of right-multiplications by triangular matrices R_k.

In practical Gaussian elimination, the matrices L_k are never formed and multiplied explicitly. The multipliers ℓ_{jk} are computed and stored directly into L, and the transformations L_k are then applied implicitly.

Algorithm 20.1. Gaussian Elimination without Pivoting

$U = A$, $L = I$
for $k = 1$ **to** $m-1$
  for $j = k+1$ **to** m
    $\ell_{jk} = u_{jk}/u_{kk}$
    $u_{j,k:m} = u_{j,k:m} - \ell_{jk}u_{k,k:m}$

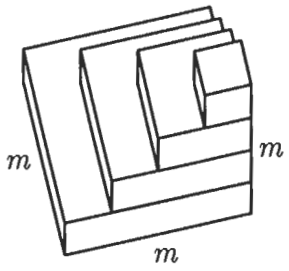
(Three matrices A, L, U are not really needed; to minimize memory use on the computer, both L and U can be written into the same array as A.) See Exercise 20.4 for an alternative "outer product" formulation of Gaussian elimination, involving one explicit loop rather than two.

Operation Count

As usual, the asymptotic operation count of this algorithm can be derived geometrically. The work is dominated by the vector operation in the inner loop, $u_{j,k:m} = u_{j,k:m} - \ell_{jk}u_{k,k:m}$, which executes one scalar-vector multiplication and one vector subtraction. If $l = m - k + 1$ denotes the length of the row vectors being manipulated, the number of flops is $2l$: two flops per entry.

For each value of k, the inner loop is repeated for rows $k+1, \ldots, m$. The

work involved corresponds to one layer of the following solid:



This is the same figure we displayed in Lecture 10 to represent the work done in Householder triangularization (assuming $m = n$). There, however, each unit cube represented four flops rather than two. As before, the solid converges as $m \to \infty$ to a pyramid, with volume $\frac{1}{3}m^3$. At two flops per unit of volume, this adds up to

$$\text{Work for Gaussian elimination:} \quad \sim \frac{2}{3}m^3 \text{ flops.} \tag{20.8}$$

Solution of $Ax = b$ by LU Factorization

If A is factored into L and U, a system of equations $Ax = b$ is reduced to the form $LUx = b$. Thus it can be solved by solving two triangular systems: first $Ly = b$ for the unknown y (forward substitution), then $Rx = y$ for the unknown x (back substitution). The first step requires $\sim \frac{2}{3}m^3$ flops, and the second and third each require $\sim m^2$ flops. The total work is $\sim \frac{2}{3}m^3$ flops, half the figure of $\sim \frac{4}{3}m^3$ flops (10.9) for a solution by Householder triangularization (Algorithm 16.1).

Why is Gaussian elimination usually used rather than QR factorization to solve square systems of equations? The factor of 2 is certainly one reason. Also important, however, may be the historical fact that the elimination idea has been known for centuries, whereas QR factorization of matrices did not come along until after the invention of computers. To supplant Gaussian elimination as the method of choice, QR factorization would have to have had a compelling advantage.

Instability of Gaussian Elimination without Pivoting

Unfortunately, Gaussian elimination as presented so far is unusable for solving general linear systems, for it is not backward stable. The instability is related to another, more obvious difficulty. For certain matrices, Gaussian elimination fails entirely, because it attempts division by zero.

For example, consider

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

This matrix has full rank and is well-conditioned, with $\kappa(A) = (3+\sqrt{5})/2 \approx 2.618$ in the 2-norm. Nevertheless, Gaussian elimination fails at the first step.

A slight perturbation of the same matrix reveals the more general problem. Suppose we apply Gaussian elimination to

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}. \tag{20.9}$$

Now the process does not fail. Instead, 10^{20} times the first row is subtracted from the second row, and the following factors are produced:

$$L = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1-10^{20} \end{bmatrix}.$$

However, suppose these computations are performed in floating point arithmetic with $\epsilon_{\text{machine}} \approx 10^{-16}$. The number $1 - 10^{20}$ will not be represented exactly; it will be rounded to the nearest floating point number. For simplicity, imagine that this is exactly -10^{20}. Then the floating point matrices produced by the algorithm will be

$$\tilde{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \qquad \tilde{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{bmatrix}.$$

This degree of rounding might seem tolerable at first. After all, the matrix $\tilde{U}$ is close to the correct U relative to $\|U\|$. However, the problem becomes apparent when we compute the product $\tilde{L}\tilde{U}$:

$$\tilde{L}\tilde{U} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix}.$$

This matrix is not at all close to A, for the 1 in the $(2,2)$ position has been replaced by 0. If we now solve the system $\tilde{L}\tilde{U}x = b$, the result will be nothing like the solution to $Ax = b$. For example, with $b = (1,0)^*$ we get $\tilde{x} = (0,1)^*$, whereas the correct solution is $x \approx (-1,1)^*$.

A careful consideration of what has occurred in this example reveals the following. Gaussian elimination has computed the LU factorization stably: $\tilde{L}$ and $\tilde{U}$ are close to the exact factors for a matrix close to A (in fact, A itself). Yet it has not solved $Ax = b$ stably. The explanation is that the LU factorization, though stable, was *not backward stable*. As a rule, if one step of an algorithm is a stable but not backward stable algorithm for solving a subproblem, the stability of the overall calculation may be in jeopardy.

In fact, for general $m \times m$ matrices A, the situation is worse than this. Gaussian elimination without pivoting is neither backward stable nor stable as a general algorithm for LU factorization. Additionally, the triangular matrices it generates have condition numbers that may be arbitrarily greater than those of A itself, leading to additional sources of instability in the forward and back substitution phases of the solution of $Ax = b$.

Exercises

20.1. Let $A \in \mathbb{C}^{m\times m}$ be nonsingular. Show that A has an LU factorization if and only if for each k with $1 \le k \le m$, the upper-left $k \times k$ block $A_{1:k,1:k}$ is nonsingular. (Hint: The row operations of Gaussian elimination leave the determinants $\det(A_{1:k,1:k})$ unchanged.) Prove that this LU factorization is unique.

20.2. Suppose $A \in \mathbb{C}^{m\times m}$ satisfies the condition of Exercise 20.1 and is banded with bandwidth $2p+1$, i.e., $a_{ij} = 0$ for $|i-j| > p$. What can you say about the sparsity patterns of the factors L and U of A?

20.3. Suppose an $m\times m$ matrix A is written in the block form $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where A_{11} is $n \times n$ and A_{22} is $(m-n) \times (m-n)$.
Assume that A satisfies the condition of Exercise 20.1.

(a) Verify the formula

$$\begin{bmatrix} I & \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}$$

for "elimination" of the block A_{21}. The matrix $A_{22} - A_{21}A_{11}^{-1}A_{12}$ is known as the *Schur complement* of A_{11} in A.

(b) Suppose A_{21} is eliminated row by row by means of n steps of Gaussian elimination. Show that the bottom-right $(m-n) \times (m-n)$ block of the result is again $A_{22} - A_{21}A_{11}^{-1}A_{12}$.

20.4. Like most of the algorithms in this book, Gaussian elimination involves a triply nested loop. In Algorithm 20.1, there are two explicit **for** loops, and the third loop is implicit in the vectors $u_{j,k:m}$ and $u_{k,k:m}$. Rewrite this algorithm with just one explicit **for** loop indexed by k. Inside this loop, U will be updated at each step by a certain rank-one outer product. This "outer product" form of Gaussian elimination may be a better starting point than Algorithm 20.1 if one wants to optimize computer performance.

20.5. We have seen that Gaussian elimination yields a factorization $A = LU$, where L has ones on the diagonal but U does not. Describe at a high level the factorization that results if this process is varied in the following ways:

(a) Elimination by columns from left to right, rather than by rows from top to bottom, so that A is made lower-triangular.

(b) Gaussian elimination applied after a preliminary scaling of the columns of A by a diagonal matrix D. What form does a system $Ax = b$ take under this rescaling? Is it the equations or the unknowns that are rescaled by D?

(c) Gaussian elimination carried further, so that after A (assumed nonsingular) is brought to upper-triangular form, additional column operations are carried out so that this upper-triangular matrix is made diagonal.