

§6.1 naïve Gaussian Elimination

Our goal is the automatic solution of systems of linear equations:

$$\begin{array}{cccccc}
 a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + \cdots + a_{1n}x_n & = & b_1 \\
 a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + \cdots + a_{2n}x_n & = & b_2 \\
 a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + \cdots + a_{3n}x_n & = & b_3 \\
 \vdots & & \vdots & & \vdots & \ddots & \vdots & \vdots \\
 a_{n1}x_1 & + & a_{n2}x_2 & + & a_{n3}x_3 & + \cdots + a_{nn}x_n & = & b_n
 \end{array}$$

In these equations, the a_{ij} and b_i are given real numbers. We also write this as

$$A\mathbf{x} = \mathbf{b},$$

where A is a matrix, whose element in the i^{th} row and j^{th} column is a_{ij} , and \mathbf{b} is a column vector, whose i^{th} entry is b_i .

This gives the easier way of writing this equation:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} \tag{1}$$

Elementary Row Operations

You may remember that one way to solve linear equations is by applying *elementary row operations* to a given equation of the system. For example, if we are trying to solve the given system of equations, they should have the same solution as the following system:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa a_{i1} & \kappa a_{i2} & \kappa a_{i3} & \cdots & \kappa a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \kappa b_i \\ \vdots \\ b_n \end{bmatrix}$$

where κ is some given number which is *not zero*. It suffices to solve this system of linear equations, as it has the same solution(s) as our original system. Multiplying a row of the system by a nonzero constant is one of the elementary row operations.

The second elementary row operation is to replace a row by the sum of that row and a constant times another. Thus, for example, the following system of equations has the same solution as the original system:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{(i-1)1} & a_{(i-1)2} & a_{(i-1)3} & \cdots & a_{(i-1)n} \\ a_{i1} + \beta a_{j1} & a_{i2} + \beta a_{j2} & a_{i3} + \beta a_{j3} & \cdots & a_{in} + \beta a_{jn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(i-1)} \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{(i-1)} \\ b_i + \beta b_j \\ \vdots \\ b_n \end{bmatrix}$$

We have replaced the i^{th} row by the i^{th} row plus β times the j^{th} row.

The third elementary row operation is to switch rows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_3 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

We have here switched the second and third rows. The purpose of this e.r.o. is mainly to make things look nice.

Note that none of the e.r.o.'s change the structure of the solution vector \mathbf{x} . For this reason, it is customary to drop the solution vector entirely and to write the matrix \mathbf{A} and the vector \mathbf{b} together in *augmented form*:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array} \right)$$

The idea of Gaussian Elimination is to use the elementary row operations to put a system into *upper triangular form* then use *back substitution*. We'll give an example here:

Example 1. Solve the set of linear equations:

$$\begin{aligned} x_1 + x_2 - x_3 &= 2 \\ 2x_1 - x_2 + 3x_3 &= 5 \\ 3x_1 + 2x_2 - 2x_3 &= 5 \end{aligned}$$

We start by rewriting in the augmented form:

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 2 & -1 & 3 & 5 \\ 3 & 2 & -2 & 5 \end{array} \right)$$

We add -2 times the first row to the second, and -3 times the first row to the third to get:

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & -3 & 5 & 1 \\ 0 & -1 & 1 & -1 \end{array} \right)$$

We now add -3 times the third row to the second row to get:

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & 0 & 2 & 4 \\ 0 & -1 & 1 & -1 \end{array} \right)$$

(We did it this way to avoid getting fractions.) Then we interchange the last two rows:

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 2 & 4 \end{array} \right)$$

The matrix is now in upper triangular form: there are no nonzero entries below the diagonal. This corresponds to the set of equations:

$$\begin{aligned} x_1 + x_2 - x_3 &= 2 \\ -x_2 + x_3 &= -1 \\ 2x_3 &= 4 \end{aligned}$$

We now solve this by back substitution. Because the matrix is in upper triangular form, we can solve x_3 by looking only at the last equation; namely $x_3 = 2$. However, once x_3 is known, the second equation involves only one unknown, x_2 , and can be solved only by $x_2 = 3$. Then the first equation has only one unknown, and is solved by $x_1 = 1$.

All sorts of funny things can happen when you attempt Gaussian Elimination: it may turn out that your system has no solution, or has a single solution (as above), or an infinite number of solutions. We should expect that an algorithm for automatic solution of systems of equations should detect these problems.

Algorithm Terminology

The method outlined above is fine for solving small systems. We should like to devise an algorithm for doing the same thing which can be applied to large systems of equations. The algorithm will take the system (in augmented form):

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array} \right)$$

The algorithm then selects the first row as the *pivot equation* or *pivot row*, and the first element of the first row, a_{11} is the *pivot element*. The algorithm then *pivots* on the pivot element to get the system:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} & b'_n \end{array} \right)$$

Where

$$\left. \begin{array}{l} a'_{ij} = a_{ij} - \left(\frac{a_{i1}}{a_{11}} \right) a_{1j} \\ b'_i = b_i - \left(\frac{a_{i1}}{a_{11}} \right) b_1 \end{array} \right\} \quad (2 \leq i \leq n, 1 \leq j \leq n)$$

Effectively we are carrying out the e.r.o. of replacing the i^{th} row by the i^{th} row minus $\left(\frac{a_{i1}}{a_{11}} \right)$ times the first row. The quantity $\left(\frac{a_{i1}}{a_{11}} \right)$ is the *multiplier* for the i^{th} row.

Hereafter the algorithm will not alter the first row or first column of the system. Thus, the algorithm could be written recursively. By pivoting on the second row, the algorithm then generates the system:

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & a''_{33} & \cdots & a''_{3n} & b''_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a''_{n3} & \cdots & a''_{nn} & b''_n \end{array} \right)$$

In this case

$$\left. \begin{array}{l} a''_{ij} = a'_{ij} - \left(\frac{a'_{i2}}{a'_{22}} \right) a'_{2j} \\ b''_i = b'_i - \left(\frac{a'_{i2}}{a'_{22}} \right) b'_2 \end{array} \right\} \quad (3 \leq i \leq n, 1 \leq j \leq n)$$

Algorithm Problems

This chapter is called naïve Gaussian Elimination because a real algorithm is a bit more complicated. The algorithm we have outlined is far too rigid—it always chooses to pivot on the k^{th} row during the k^{th} step. This would be bad if the pivot element were zero; in this case all the multipliers $\frac{a_{ik}}{a_{kk}}$ are not defined.

Bad things can happen if a_{kk} is merely small instead of zero. Consider the following example:

Example 2. Solve the system of equations given by the augmented form:

$$\left(\begin{array}{cc|c} 0.0003 & 1.566 & 1.569 \\ 0.3454 & -2.436 & 1.018 \end{array} \right)$$

Note that the exact solution of this system is $x_1 = 10$, $x_2 = 1$. Suppose, however, that the algorithm uses only 4 significant figures for its calculations. The algorithm, naïvely, pivots on the first equation. The multiplier for the second row is

$$\frac{0.3454}{0.0003} = 1151.3\bar{3},$$

which will be rounded to 1151 by the algorithm.

The second entry in the matrix is replaced by

$$-2.436 - (1151)(1.566) = -2.436 - 1802 = -1804,$$

where the arithmetic is rounded to four significant figures each time. Similarly, the second vector entry becomes:

$$1.018 - (1151)(1.569) = 1.018 - 1806 = -1805,$$

so the system is in the form

$$\left(\begin{array}{cc|c} 0.0003 & 1.566 & 1.569 \\ & 0 & -1805 \end{array} \right)$$

When the algorithm attempts back substitution, the value

$$x_2 = \frac{-1805}{-1804} \approx 1.0000554324$$

is rounded to 1.001. This is not so far off. However, it now computes

$$x_1 = (1.569 - 1.566 \cdot 1.001) / 0.0003 = (1.569 - 1.568) / 0.0003 = (0.001) / 0.0003 = 3.3\bar{3},$$

which the computer rounds to 3.333. This is far from the right answer. Looks like there was some subtractive cancelling in the last part which made things bad.

A Lamé OpCount

At this point in the class it became obvious that the four pages of material I had prepared did not equal 50 minutes of class time. The problem is that much of these four pages consist of matrices, which take up a lot of space. So I tried to analyze the operation count for Gaussian Elimination. Seems like a simple enough exercise, right?

We consider how many operations are required to reduce

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array} \right)$$

to

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} & b'_n \end{array} \right)$$

First off, you compute the multiplier for each row. Then in each row you do n multiplies and n adds. This gives a total of $(n-1) + (n-1)n$ multiplies (counting in the computing of the multiplier in each of the $(n-1)$ rows) and $(n-1)n$ adds. In total this is at least n^2 and less than $2n^2$ operations.

Consider that we have to do this recursively on the lower right subsystem, which is an $(n-1)$ by $(n-1)$ system. Thus we perform fewer than $2(n-1)^2$ operations. In total, then, we use around

$$2 [1^2 + 2^2 + \dots + (n-1)^2 + n^2]$$

operations. At this point, things really get goofy because I am obviously not prepared and have forgotten the exact form of the answer to this one. My second guess, however, is the correct one, *i.e.*, that this series is

$$2\frac{1}{6}(n)(n+1)(2n+1),$$

so we perform fewer than $\frac{2n^3}{3}$ operations, fewer than the n^3 that I quoted for matrix inverse, but only by a constant. So Gaussian Elimination is not *so* great.