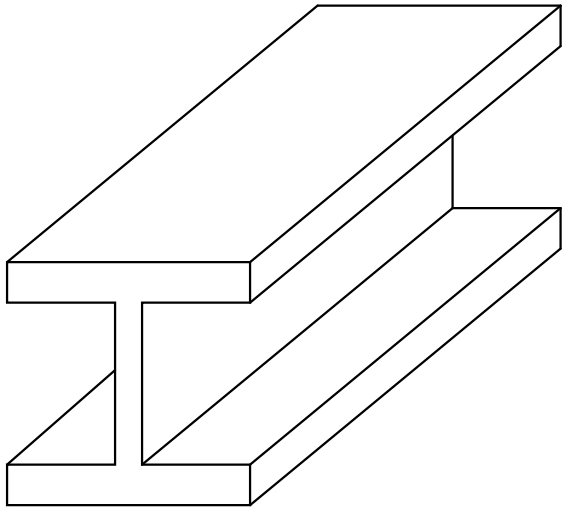
CEE384—Numerical Methods

Naive Gauss Elimination

Solving Linear Systems of Equations



Arizona State University

Brian Chevalier

Updated: October 21, 2019

Contents

1 Naive Gauss Elimination

Naive Gauss Elimination is a linear algebra method to solve matrix equations of the following form:

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

This method is similar to a method called Reduced Row Echelon Form that you may be familiar with from previous courses.

$$\mathbf{A} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{2}$$

Note: the indexing used here is zero-indexing. That is, numbering begins at zero, not at one like typical linear algebra. That is because almost all programming languages use zero-indexing.

1.1 Forward Elimination

Forward elimination is the process by which a matrix is converted into an upper triangular matrix, which is where all elements below the diagonal are zero.

Let's take a look at the first step in the process. We want to make element $A_{1,0}$ equal to zero such as in Equation 3

$$\mathbf{A} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{3}$$

To accomplish this, we take the row with the element we want to eliminate (row 1), and subtract row 0 by some scaled amount that will guarantee $A_{1,0}$ will become zero.

$$A_{1,:} = A_{1,:} - \frac{A_{1,0}}{A_{0,0}} A_{0,:} \tag{4}$$

Note: the notation $A_{1,:}$ reads as "row one, all of the columns". Similarly, $A_{:,0}$ would read as "all of the rows of the zero column". In linear algebra classes this may be written as "R_1", however the notation used here makes it easier to jump into programming the logic later.

We are now left with a zero in position $A_{1,0}$.

$$\mathbf{A} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ 0 & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{5}$$

Next, eliminate $A_{2,0}$ with the same method as before.

$$A_{2,:} = A_{2,:} - \frac{A_{2,0}}{A_{0,0}} A_{0,:} \tag{6}$$

For one last time, let's eliminate $A_{2,1}$.

$$\mathbf{A} = \begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ 0 & A_{1,1} & A_{1,2} & A_{1,3} \\ 0 & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \tag{7}$$

The equation applied is:

$$A_{2,:} = A_{2,:} - \frac{A_{2,1}}{A_{1,1}} A_{1,:} \tag{8}$$

Matching the Pattern. With just three iterations a clear pattern emerges, as shown in Table 2. Putting the expressions in a table along with the current row and column number allows us to match the indicies in the expression as a function of the current row, i, and current column, j. Equation 9 summarizes this expression for every row and column.

Table 1: Matching the Pattern

Row (i)	Col. (j)	Expression
1	0	$A_{1,:} = A_{1,:} - \frac{A_{1,0}}{A_{0,0}} A_{0,:}$
2	0	$A_{2,:} = A_{2,:} - \frac{A_{2,0}}{A_{0,0}} A_{0,:}$
2	1	$A_{2,:} = A_{2,:} - \frac{A_{2,1}}{A_{1,1}} A_{1,:}$

Establishing the Bounds We also need to establish the bounds where this expression is valid (i.e. which rows and columns should we apply this to?) The expression was first applied to row 1 (skipping row 0), and continues to less than n, where n is the number of rows in the matrix (in this case 4). So in our case i would vary from and include 1 and go to but not include 4. Next, the columns are dependent on the current row. We will look at rows 0 up to but not including the i column.

$$A_{i,:} -= \frac{A_{i,j}}{A_{j,j}} A_{j,:} \quad (1 \leq i < n) \quad (0 \leq j < i) \tag{9}$$

All of the changes made to each row of $\mathbf{A}$ must also be applied to each row of $\mathbf{b}$. This pattern is the same before, but modifications are made to the i row of $\mathbf{b}$. This is given in Equation 10.

$$b_i -= \frac{A_{i,j}}{A_{j,j}} b_j \quad (1 \leq i < n) \quad (0 \leq j < i) \tag{10}$$

1.2 Backward Substitution

After completing the process of forward elimination we are left with a matrix equation that looks like the following:

$$\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ 0 & A_{1,1} & A_{1,2} & A_{1,3} \\ 0 & 0 & A_{2,2} & A_{2,3} \\ 0 & 0 & 0 & A_{3,3} \end{bmatrix} \begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix} \tag{11}$$

To find the elements of $\mathbf{x}$ we must work backward starting with the last row and rearrange each equation to solve for x_i.

Matching the Pattern. There are two patterns within this expression. The b term in the numerator is always b_i, and the term in the denominator is always $A_{i,i}$. In the numerator there is also a summation term that grows with each iteration. This term is summing the A terms in the current row with each x term that matches its respective column. Equation 12 summarizes the pattern for backward substitution.

Table 2: Matching the Pattern

Row (i)	Expression
3	$x_3 = \dfrac{b_3}{A_{3,3}}$
2	$x_2 = \dfrac{b_2 - [A_{2,3}x_3]}{A_{2,2}}$
1	$x_1 = \dfrac{b_1 - [A_{1,2}x_2 + A_{1,3}x_3]}{A_{1,1}}$
0	$x_0 = \dfrac{b_0 - [A_{0,1}x_1 + A_{0,2}x_2 + A_{0,3}x_3]}{A_{0,0}}$

Establishing the Bounds. The bounds of this expression range from row 0 up to but not including the number of rows (4). This is given below.

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} A_{i,j}x_j}{A_{i,i}} \quad (0 \leq i < n) \tag{12}$$

Note: if you are implementing this in a programming language with a zero-based index you should reverse the value range. Reversing with a built in function is the easiest way, otherwise start with $n-1$ decrementing by 1, down to but not including -1.

1.3 Limitations

1. There may be instances during the process of forward elimination where the order of the rows in the matrix A require division by zero (i.e. when $A_{0,0}$ is zero). To avoid this problem we use a method called "partial pivoting". This means rearranging the rows such that division by zero errors are not encountered.

2. There may be instances where division by very small (almost zero) values occur. This will cause greater numerical errors that could be significant. We can use "full pivoting" to arrange the rows such division always occurs with the greatest possible denominator to minimize this error.

3. There are times we may want to analyze a system of equations with many **b** vectors, as is the case when a structure is subjected to many load cases. Using Gauss Elimination would require redoing work each time a new load case should be analyzed.

References

[1] Chapra, Steven C. and Canale, Raymond P., *Numerical Methods for Engineers*, 7th ed. McGraw Hill Education, 2015.