1.2.1 Reduction of $A\underline{x} = \underline{b}$ to triangular form by Gaussian Elimination

We now with to develop an automatic procedure, an algorith, for obtaining a solution to the set of N linear algebraic equations:

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\
&\qquad\vdots \quad \vdots \\
&\qquad\vdots \quad \vdots \\
&a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n
\end{aligned} \qquad \textbf{(1.2.1-1)}$$

Note that we can select any two rows, say #j and #k, and add the two equations to obtain another one that is equally valid.

$$\begin{array}{c}
a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jN}x_N = b_N \\
+ (a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kN}x_N = b_k) \\
\hline
(a_{j1} + a_{k1})x_1 + (a_{j2} + a_{k2})x_2 + \ldots + (a_{jN} + a_{kN})x_N = (b_j + b_k)
\end{array}$$

(1.2.1-2)

If the equation for row #j is satisfied, and the equation obtained by summing rows j and k **(1.2.1-2)** is satisfied, then it automatically follows that the equation of row k must be satisfied.

We are therefore free to replace in our system of equations the equation
$a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kN}x_N = b_k$ by
$(a_{j1} + a_{k1})x_1 + (a_{j2} + a_{k2})x_2 + \ldots + (a_{jN} + a_{kN})x_N = (b_j + b_k)$ with <u>no effect</u> on the solution $\underline{x}$.

Similarly, we can take any row, say #j, multiply it by a non-zero scalar c, to obtain an equation

$$ca_{j1}x_1 + ca_{j2}x_2 + \ldots + ca_{jN}x_N = cb_j \quad \textbf{(1.2.1-3)}$$

that we can substitute for the original without effecting the solution.

In general, for the linear system

$$\begin{array}{c}
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1 \\
\vdots \\
a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jN}x_N = b_j \;\leftarrow \text{row \#j} \\
\vdots \\
a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kN}x_N = b_k \;\leftarrow \text{row \#k} \\
\vdots \\
a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N = b_N
\end{array}$$

We can choose any $j \in [1,N]$, $k \in [1,N]$ and any scalar $c \neq 0$ to form the following linear system with exactly the same solution as **(1.2.1-4)**.

$$\begin{array}{c}
a_{11}x_1 + \ldots + a_{1N}x_N = b_1 \\
\vdots \\
a_{j1}x_1 + \ldots + a_{jN}x_N = b_j \;\leftarrow \text{row \#j} \\
\vdots \\
(ca_{j1} + a_{k1})x_1 + \ldots + (ca_{jN} + a_{kN})x_N = (cb_j + b_k) \;\leftarrow \text{row \#k} \\
\vdots \\
a_{N1}x_1 + \ldots + a_{NN}x_N = b_N \quad \textbf{(1.2.1-5)}
\end{array}$$

The process of converting **(1.2.1-4)** into the equivalent system **(1.2.1-5)** is known as an elementary row operation.

We develop in this section an algorithm for solving a system of linear equations by performing a sequence of these elementary row operations until the system is of a form that is easy to solve.

We will wish to use matrix/vector notation $A\underline{x} = \underline{b}$, so we write the system as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ : & : & & \\ a_{j1} & a_{j2} & \dots & a_{jN} \\ : & : & & \\ a_{k1} & a_{k2} & \dots & a_{kN} \\ : & : & & \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ : \\ x_j \\ : \\ x_k \\ : \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ : \\ b_j \\ : \\ b_k \\ : \\ b_N \end{bmatrix} \qquad \textbf{(1.2.1-6)}$$

After a row operation, we have the equivalent system $A'\underline{x} = \underline{b}'$

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{j1} & a_{j2} & \dots & a_{jN} \\ : & : & & \\ (ca_{j1} + a_{k1}) & (ca_{j2} + a_{k2}) & \dots & (ca_{jN} + a_{kN}) \\ : & : & & \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ : \\ x_j \\ : \\ x_k \\ : \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ : \\ b_j \\ : \\ (cb_j + b_k) \\ : \\ b_N \end{bmatrix} \qquad \textbf{(1.2.1-7)}$$

Since we must change both A and $\underline{b}$ for every elementary row operation, it is common to perform these operations on the augmented matrix (A, $\underline{b}$)

$$(A, \underline{b}) = \underbrace{\begin{bmatrix} a_{11} & \dots & a_{1N} & b_1 \\ : & & & \\ a_{j1} & \dots & a_{jN} & b_j \\ : & & & \\ a_{k1} & \dots & a_{kN} & b_k \\ : & & & \\ a_{N1} & \dots & a_{NN} & b_N \end{bmatrix}}_{N \text{ x } (N+1) \text{ Matrix}} \qquad \textbf{(1.2.1-8)}$$

After our elementary row operation, the system is written as

$$(A', \underline{b}') = \underbrace{\begin{bmatrix} a_{11} & \dots & a_{1N} & b_1 \\ : & & & \\ a_{j1} & \dots & a_{jN} & b_j \\ : & & & \\ (ca_{j1} + a_{k1}) & \dots & (ca_{jN} + a_{kN}) & (cb_j + b_k) \\ : & & & \\ a_{N1} & \dots & a_{NN} & b_N \end{bmatrix}}_{N \text{ x } (N+1) \text{ Matrix}} \qquad \textbf{(1.2.1-9)}$$

By applying the elementary row operation to the augmented matrix, we automatically change both the left and right hand sides of $A\underline{x} = \underline{b}$ as needed to ensure that the solution $\underline{x}$ does not change

We will now build a systematic approach to solving $A\underline{x} = \underline{b}$ based opon a sequence of these row operations.

We now present the standard approach to solving $A\underline{x} = \underline{b}$, the
<u>Method of Gaussian Elimination.</u>

First, we start with the original augmented matrix $(A, \underline{b})$ of the system,

$$(A, \underline{b}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{j1} & a_{22} & a_{23} & \dots & a_{jN} & b_j \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{k1} & a_{32} & a_{33} & \dots & a_{kN} & b_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{NN} & b_N \end{bmatrix} \qquad \textbf{(1.2.1-10)}$$

As long as $a_{11} \neq 0$ (in section 1.2.3 we show how we can ensure that this is the case for systems with a unique solution), we can define the finite scalar

$$\lambda_{21} = \frac{a_{21}}{a_{11}} \quad \textbf{(1.2.1-11)}$$

We now perform a row operation where we replace the 2nd row by the equation

$$\begin{array}{c} -\lambda_{21}(a_{11}x_1 + a_{12}x_2 + \dots + a_{1N}x_N = b_1) \\ + \quad (a_{21}x_1 + a_{22}x_2 + \dots + a_{2N}x_N = b_2) \\ \hline (a_{21} - \lambda_{21}a_{11})x_1 + (a_{22} - \lambda_{21}a_{12})x_2 + \dots + (a_{2N} - \lambda_{21}a_{1N})x_N = (b_2 - \lambda_{21}b_1) \end{array}$$

(1.2.1-12)

We see that the coefficient multiplying x_1 in this equation is

$$(a_{21} - \lambda_{21}a_{11}) = a_{21} - \left(\frac{a_{21}}{a_{11}}\right)a_{11} = a_{21} - a_{21} = 0 \qquad \textbf{(1.2.1-13)}$$

If we write the augmented matrix obtained from this row operation to place a zero in the (2,1) position – row #2 column #1 – as

$(A^{(2,1)}, \underline{b}^{(2,1)}) =$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1N} & b_1 \\ 0 & (a_{22} - \lambda_{21}a_{12}) & (a_{23} - \lambda_{21}a_{13}) & \dots & (a_{2N} - \lambda_{21}a_{1N}) & (b_2 - \lambda_{21}b_1) \\ a_{31} & a_{32} & a_{33} & \dots & a_{3N} & b_3 \\ : & : & : & : & : & : \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{NN} & b_N \end{bmatrix} \quad \textbf{(1.2.1-14)}$$

Again, the important point is that the linear system $A^{(2,1)}\ \underline{x} = \underline{b}$ has the same solution $\underline{x}$ as the original system $A\underline{x} = \underline{b}$.

As we develop this method, let us consider a particular system; for example, the set of equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 4 \\ 2x_1 + x_2 + 3x_3 &= 7 \\ 3x_1 + x_2 + 6x_3 &= 2 \quad \textbf{(1.2.1-15)} \end{aligned}$$

The original augmented matrix for this system is

$$(A, \underline{b}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 2 & 1 & 3 & 7 \\ 3 & 1 & 6 & 2 \end{bmatrix} \quad \textbf{(1.2.1-16)}$$

Since $a_{11} \neq 0$, we can define $\lambda_{21} = \dfrac{a_{21}}{a_{11}} = \dfrac{2}{1} = 2$ **(1.2.1-17)**

We now perform the row operation **(1.2.1-10) → (1.2.1-14)** to obtain

$$(A^{(2,1)}, \underline{b}^{(2,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 2-(2)(1) & 1-(2)(1) & 3-(2)(1) & 7-(2)(4) \\ 3 & 1 & 6 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 3 & 1 & 6 & 2 \end{bmatrix} \quad \textbf{(1.2.1-18)}$$

We now wish to continue this process to place a zero in the (3,1) position of the augmented matrix.
If $a_{2j}^{(2,1)} = a_{2j} - \lambda_{21} a_{1j}, b_2^{(2,1)} = b_2 - \lambda_{21} b_1$ **(1.2.1-19)**

We write the augmented matrix following the first row operation as

$$(A^{(2,1)}, \underline{b}^{(2,1)}) =$$
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \cdots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} & b_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} & b_N \end{bmatrix} \quad \textbf{(1.2.1-20)}$$

If $a_{11} \neq 0$, we now define the finite scalar $\lambda_{31} = \frac{a_{31}}{a_{11}}$ **(1.2.1-21)**

And perform the following row operation, noting

$$a_{31} - \lambda_{31} a_{11} = a_{31} - \left(\frac{a_{31}}{a_{11}}\right) a_{11} = a_{31} - a_{31} = 0 \quad \textbf{(1.2.1-22)}$$

$(A^{(3,1)}, \underline{b}^{(3,1)}) =$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \cdots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ 0 & a_{32}^{(3,1)} & a_{33}^{(3,1)} & \cdots & a_{3N}^{(3,1)} & b_3^{(3,1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} & b_N \end{bmatrix} \quad \textbf{(1.2.1-23)}$$

Where

$$a_{3j}^{(3,1)} = a_{3j} - \lambda_{31} a_{1j}, \; b_3^{(3,1)} = b_3 - \lambda_{31} b_1 \quad \textbf{(1.2.1-24)}$$

For our system with $(A^{(2,1)}, \underline{b}^{(2,1)})$ given by **(1.2.1-18)**,

$$\lambda_{31} = \frac{a_{31}}{a_{11}} = \frac{3}{1} = 3 \quad \textbf{(1.2.1-25)}$$

and

$$(A^{(3,1)}, \underline{b}^{(3,1)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 3-(3)(1) & 1-(3)(1) & 6-(3)(1) & 2-(3)(4) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & -2 & 3 & -10 \end{bmatrix} \quad \textbf{(1.2.1-26)}$$

In general, for N > 3, we continue this process of row operations until all elements of the 1st column are zero except a_N. The augmented system matrix at this stage of the calculation will be:

$$(A^{(N,1)}, \underline{b}^{(N,1)}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \ldots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ 0 & a_{32}^{(3,1)} & a_{33}^{(3,1)} & \ldots & a_{3N}^{(3,1)} & b_3^{(3,1)} \\ : & : & : & : & : & \\ 0 & a_{N2}^{(N,1)} & a_{N3}^{(N,1)} & \ldots & a_{NN}^{(N,1)} & b_N^{(N,1)} \end{bmatrix} \quad \textbf{(1.2.1-27)}$$

We now move to the 2nd column and perform row operations to place all zeros below the (2,2) position.

First, if $a_{22}^{(2,1)} \neq 0$ we define $\lambda_{32} = \dfrac{a_{32}^{(3,1)}}{a_{22}^{(2,1)}}$ **(1.2.1-28)** and perform the row operation

$$a_{3j}^{(3,2)} = a_{3j}^{(3,1)} - \lambda_{32} a_{2j}^{(2,1)}, \quad b_3^{(3,2)} = b_3^{(3,1)} - \lambda_{31} b_1 \quad \textbf{(1.2.1-29)}$$

so that $a_{32}^{(3,2)} = 0$

$$(A^{(3,2)}, \underline{b}^{(3,2)}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \ldots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & a_{33}^{(3,2)} & \ldots & a_{3N}^{(3,2)} & b_3^{(3,2)} \\ : & : & : & : & : & \\ 0 & a_{N2}^{(N,1)} & a_{N3}^{(N,1)} & \ldots & a_{NN}^{(N,1)} & b_N^{(N,1)} \end{bmatrix} \quad \textbf{(1.2.1-30)}$$

For our system with $(A^{(3,1)}, \underline{b}^{(3,1)})$ given by **(1.2.1-26)**,

$$\lambda_{32} = \frac{a_{32}^{(3,1)}}{a_{22}^{(2,1)}} = \frac{(-2)}{(-1)} = 2 \qquad \textbf{(1.2.1-31)}$$

so that

$$(A^{(3,2)}, \underline{b}^{(3,2)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & (-2)-(2)(-1) & (3)-(2)(1) & (-10)-(2)(-1) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & -8 \end{bmatrix} \qquad \textbf{(1.2.1-32)}$$

In general for N > 3, we continue this process with another N - 3 row operations $(A^{(3,2)}, \underline{b}^{(3,2)}) \to (A^{(4,2)}, \underline{b}^{(4,2)}) \to \ldots \to (A^{(N,2)}, \underline{b}^{(N,2)})$ to place all zeros in the 2nd column.

$$(A^{(N,2)}, \underline{b}^{(N,2)}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & \ldots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & a_{33}^{(3,2)} & \ldots & a_{3N}^{(3,2)} & b_3^{(3,2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & a_{N3}^{(N,2)} & \ldots & a_{NN}^{(N,2)} & b_N^{(N,2)} \end{bmatrix} \textbf{(1.2.1-33)}$$

We then move to the 3rd column and perform row operations to place all zeros below the (3,3) position then in column #4, we put all zeros below (4,4), etc.

When we are finished, the system augmented matrix is of the following upper triangular form

$$(A^{(N,N\text{-}1)}, \underline{b}^{(N,N\text{-}1)}) = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1N} & b_1 \\ 0 & a_{22}^{(2,1)} & a_{23}^{(2,1)} & a_{24}^{(2,1)} & \dots & a_{2N}^{(2,1)} & b_2^{(2,1)} \\ 0 & 0 & a_{33}^{(3,2)} & a_{34}^{(3,2)} & \dots & a_{3N}^{(3,2)} & b_3^{(3,2)} \\ 0 & 0 & 0 & a_{44}^{(4,3)} & \dots & a_{4N}^{(4,3)} & b_4^{(4,3)} \\ : & : & : & : & : & : & : \\ 0 & 0 & 0 & 0 & \dots & a_{NN}^{(N,N\text{-}1)} & b_N^{(N,N\text{-}1)} \end{bmatrix} \qquad \textbf{(1.2.1-34)}$$

For our example system with N =3, the final augmented matrix is

$$(A^{(3,2)}, \underline{b}^{(3,2)}) = \begin{bmatrix} 1 & 1 & 1 & 4 \\ 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & -8 \end{bmatrix} \qquad \textbf{(1.2.1-32, repeated)}$$

By “upper triangular” we mean that the only non-zero elements are found on or “above” the principal diagonal.

$$\begin{bmatrix} a_{11} & & & \\ & \backslash & & \\ & a_{22} & & \\ & & \backslash \quad \Rightarrow \text{"above} & \\ \text{"below} \Leftarrow & & a_{33} \quad \text{the diagonal"} & \\ \text{the diagonal"} & & \quad \backslash & \\ & & \quad a_{NN} & \end{bmatrix}$$

the “principal diagonal”

We now write out the equations of the system defined by this modified matrix, $A^{(N,N\text{-}1)}\underline{x} = \underline{b}^{(N,N-1)}$

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1N}x_N = b_1\\
&a_{22}^{(2,1)}x_2 + a_{23}^{(2,1)}x_3 + \ldots + a_{2N}^{(2,1)}x_N = b_2^{(2,1)}\\
&a_{33}^{(3,2)}x_3 + \ldots + a_{3N}^{(3,2)}x_N = b_3^{(3,2)}\\
&\qquad\qquad\vdots\\
&\qquad\qquad\vdots\\
&a_{N-1,N-1}^{(N-1,N-2)}x_{N-1} + a_{N-1,N}^{(N-1,N-2)}x_N = b_{N-1}^{(N-1,N-2)}\\
&a_{NN}^{(N,N-1)}x_N = b_N^{(N,N-1)}
\end{aligned} \qquad \textbf{(1.2.1-35)}$$

For our system with N = 3, **(1.2.1-32)** gives the equations

$$\begin{aligned}
x_1 + x_2 + x_3 &= 4\\
-x_2 + x_3 &= -1\\
x_3 &= -8
\end{aligned} \qquad \textbf{(1.2.1-36)}$$

We see that we can immediately solve for the last unknown from

$$x_N = \frac{b_N^{(N,N-1)}}{a_{NN}^{(N,N-1)}} \qquad \textbf{(1.2.1-36)}$$

For **(1.2.1-36)**, we have $x_3 = -8/1 = -8$ **(1.2.1-38)**

Then, we can work backwards, next solving for x_{N-1},

$$a_{N-1,N-1}^{(N-1,N-2)}x_{N-1} + a_{N-1,N}^{(N-1,N-2)}x_N = b_{N-1}^{(N-1,N-2)} \Rightarrow x_{N-1} = \frac{[b_{N-1}^{(N-1,N-2)} - a_{N-1,N}^{(N-1,N-2)}x_N]}{a_{N-1,N-1}^{(N-1,N-2)}} \qquad \textbf{(1.2.1-39)}$$

For **(1.2.1-36)**,

$$x_2 = \frac{[-1-(1)(-8)]}{(-1)} = -7 \qquad \textbf{(1.2.1-40)}$$

Next we solve for x_{N-2},

$$x_{N-2} = \frac{[b_{N-2}^{(N-2,N-3)} - a_{N-2,N-2}^{(N-2,N-3)} x_{N-1} - a_{N-2,N}^{(N-2,N-3)} x_N}{a_{N-2,N-2}^{(N-2,N-3)}} \qquad \textbf{(1.2.1-41)}$$

For **(1.2.1-36)**

$$x_1 = \frac{[4-(1)(-7)-(1)(-8)]}{1} = 19 \qquad \textbf{(1.2.1-42)}$$

Therefore, the solution to the set of equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 4 \\ 2x_1 + x_2 + 3x_3 &= 7 \qquad \textbf{(1.2.1-15, repeated)} \\ 3x_1 + x_2 + 6x_3 &= 2 \end{aligned}$$

is

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 19 \\ -7 \\ -8 \end{bmatrix} \qquad \textbf{(1.2.1-43)}$$

While we have found one solution to the set of equations **(1.2.1-15)** it remains to be proven in section 1.3 that this is the only solution.

The process of solving for the unknowns by starting at the last position, and working backwards with the equation set generated by the upper triangular matrix is known as <u>backward substitution</u>.

We therefore have the following algorithm to transform our system to upper triangular form using Gaussian elimination:

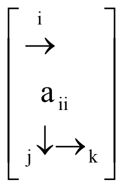
> allocate N^2 memory locations to store A, and N each for $\underline{x}$ and $\underline{b}$

```
For i = 1, 2, …, N-1      % iterate over columns from left → right
        >if $a_{ii} = 0$, STOP to avoid division by zero

    For j = i+1, i+2, …, N   % iterate over rows below diagonal
        > $\lambda \leftarrow \frac{a_{ji}}{a_{ii}}$

        For k = I, I+1, …, N  % iterate in row j from the ith column to right
            ➢ $a_{jk} \leftarrow a_{jk} - \lambda a_{ik}$
        end

        >$b_j \leftarrow b_j - \lambda b_i$
    end
end
```



We see that the basic computational unit of this algorithm is d ← a + b*c
Comprised of two "floating point operations", an addition and a multiplication, plus a memory assignment in d denoted as ←.

Each of these operations is equivalent, by most people's definition, to 2 FLOP's (FLOP = "floating point operation", +, -, , / between two numbers in memory). I will call the combined operation d ← a + b*c a "computational unit", or CU, 1CU = 2 FLOP's.

We see that since there are three nested loops running over all or a part of [1,N], that the number of CU's, or FLOP's, scales as N^3. Gaussian elimination using this algorithm is very slow for large N!

```
% perform backward substitution
>Allocate N memory locations for the components of x

For I = N, N-1, … 1 % iterate over rows from bottom
    >sum = 0

    For j = I+1, I+2, …, N
        >sum ← sum + $a_{ij}$*$x_j$
    end
    >$x_i$ ← $\frac{[b_i - \text{sum}]}{a_{ii}}$
end
```

As there are only two nested loops, the number of calculations required to perform backward substitution scales as N^2.

So, in the limit of large N (i.e. many unknowns) it takes far more time (N^3) to perform the elimination row operations necessary to put the system in upper triangular form than it does ($N^2 << N^3$) to perform backward substitution.

Let us now determine the exact number of FLOP's, "floating point operations", required to perform the elimination, υ_{elim}, and the backward substitution, υ_{sub}.

First, consider the Gaussian elimination step.

Let v_1 = # of FLOP's required to put all zeros in the 1st column

To put a zero at (2,1) to go from **(1.2.1-10)** to **(1.2.1-14)**, we see on page 1.2.1-6 that we need to perform a row operation involving 2(N+1) FLOP's, $d \leftarrow a + b*c$ = 2 FLOP's

$$\begin{array}{ll} & 1\text{ FLOP} - \text{calc. } \lambda_{21} \\ & 2 \text{ x } (N-1) \text{ FLOP's} - \text{calc. } a_{22}^{(2,1)}, a_{23}^{(2,1)}, \ldots, a_{2N}^{(2,1)} \\ & 2 \text{ x } 1 \text{ FLOP} - \text{calc. } b_2^{(2,1)} \\ \cline{2-2} \sim & 2\,(N+1)\text{ FLOP's for the row operation} \end{array}$$

To place all zeros in the remainder of column 1, we need to perform another (N-2) row operations to place zeros at (3,1), (4,1) … (N,1).
Each row operation requires 2(N+1) FLOP's.

So, the total number of FLOP's required to place all zeros below the diagonal in column1 is

$$v_1 = \underbrace{(N-1)}_{\text{\# of row operations}}\underbrace{2(N+1)}_{\text{\# of FLOP's per row operation}} \quad \textbf{(1.2.1-44)}$$

Next to move from $(A^{(N,1)}, \underline{b}^{(N,1)})$ **(1.2.1-27)** to $(A^{(3,2)}, \underline{b}^{(3,2)})$ **(1.2.1-30)**
We need to perform a row operation involving one FLOP less than those in column 1 – so each column 2 row operation involves 2N FLOP's. We need to perform now only (N-2) row operations, so

$$v_2 = \underbrace{(N-2)}_{\text{\# of row operations}}\underbrace{(2N)}_{\text{\# of FLOP's per row operation}} \quad \textbf{(1.2.1-45)}$$

In general, to place zeros below the diagonal in column #I will require (N – i) # of row operations, each involving 2(N + 2 – i) FLOP's. The total number of FLOP's required for column #I is therefore

$$v_i = \underbrace{(N - i)}_{\text{\# of row operations}}\underbrace{(N + 2 - i)2}_{\text{\# of FLOP's per row operation}} \quad \textbf{(1.2.1-46)}$$

The total number of FLOP's required to perform the Gaussian elimination is therefore

$$V_{elim} = \sum_{i=1}^{N-1} v_i = \sum_{i=1}^{N-1} (N-i)(N+2-i)2 \underset{\substack{\uparrow \\ \text{For large N}}}{\approx} 2\int_1^{N-1} (N-x)(N+2-x)dx \quad \textbf{(1.2.1-47)}$$

For N very large, $N\text{-}1 \approx N, N + 2 - x \approx N - x$, so

$$v_{elim} \approx 2\int_0^N (N-x)^2\,dx = 2\int_0^N (N^2 - 2xN + x^2)dx$$

$$\approx 2[N^2\int_0^N dx - 2N\int_0^N xdx + \int_0^N x^2dx] = 2[N^2N - 2N(\frac{1}{2}N^2) + \frac{1}{3}N^3]$$

So, in the limit of large N,

$$v_{elim}\ \frac{2}{3}N^3 \quad \textbf{(1.2.1-48)}$$

For backward substitution, inspection of the algorithm on page 1.2.1-15 shows that $v_{sub.}$, the total number of required FLOP's is

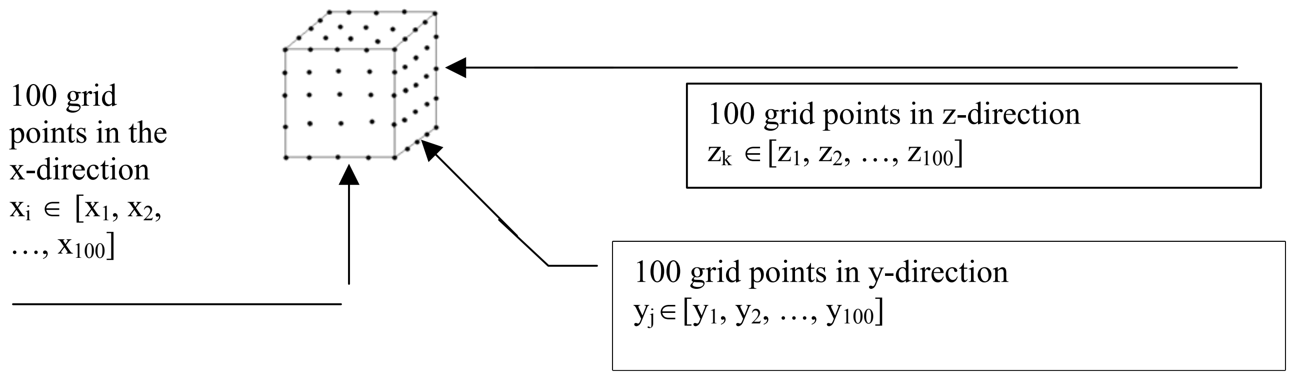
$$V_{sub.} = \sum_{i=1}^{N}(N - i + 1) \approx 2\int_1^N (N - x + 1)dx \approx 2\int_0^N (N-x)dx = 2[N^2 - \frac{1}{2}N^2] = N^2$$

$$\textbf{(1.2.1-49)}$$

Therefore, for large systems of equations, where the number of calculations required to solve the system is a concern, $v_{elim} >> v_{sub}$, so the CPU time required to perform backward substitution is trivially small.

Such scaling issues are a large concern, since in this course we will often solve large systems of linear equations.

For example, let us say that we wish to solve a partial differential equation (e.g. the diffusion equation $\nabla^2 c = 0$) on a 3-D domain. We place a grid of points (x_i, y_j, z_k) throughout the computational domain,


100 grid points in the x-direction $x_i \in [x_1, x_2, \ldots, x_{100}]$
100 grid points in z-direction $z_k \in [z_1, z_2, \ldots, z_{100}]$
100 grid points in y-direction $y_j \in [y_1, y_2, \ldots, y_{100}]$

The total number of grid points (x_i, y_j, z_k) in the 3-D mesh is $10^2 \times 10^2 \times 10^2 = 10^6$

If we develop a method to solve a linear system of algebraic equations for the concentration at each point, we have a system of 10^6 equations for 10^6 unknowns.

For Gaussian elimination, look at # FLOP's and # of memory locations required to solve a system of N equations

N	# FLOP's ($\frac{2}{3}N^3$)	# of memory locations to store A (N^2)
$10^2 = 100$	$\approx \frac{2}{3} \times (10^2)^3 \approx \frac{2}{3} \times 10^6$	$(10^2)^2 = 10{,}000$
$10^4 = 10{,}000$	$\approx \frac{2}{3} \times (10^4)^3 \approx \frac{2}{3} \times 10^{12}$	$(10^4)^2 = 10^8 = 100{,}000{,}000$
$10^6 = 1{,}000{,}000$	$\approx \frac{2}{3} \times (10^6)^3 \approx \frac{2}{3} \times 10^{18}$	$(10^6)^2 = 10^{12} =$ 1,000,000,000,000

For a system of 10^6 unknowns we would need 10^{12} locations in memory to store A – not possible.

Even if we could store A, then even on a tera-FLOP supercomputer (10^{12} FLOP's per second performance), the CPU time required to perform the Gaussian elimination is

$$\text{CPU time} \approx \frac{10^{18}\,\text{FLOP's}}{10^{12}\,\text{FLOP's/sec.}} \approx (10^6\,\text{s})(\frac{1\text{min}}{60\text{s}})(\frac{1\text{hr}}{60\text{min}})(\frac{1\text{day}}{24\text{hr}}) \approx 12\text{days!!}$$

Even though Gaussian elimination appears useful (as we have developed it) for small systems, we will need to develop other methods later to solve the large systems that arise when solving partial differential equations and other challenging problems.

Note, even for small systems, some question's remain:

- What do we do if a_{11}=0, or another a_{ji} when we calculate the λ factors during Gaussian elimination.
- We have found a way to obtain one solution, if no problems with division by zero. Are there any other solutions?