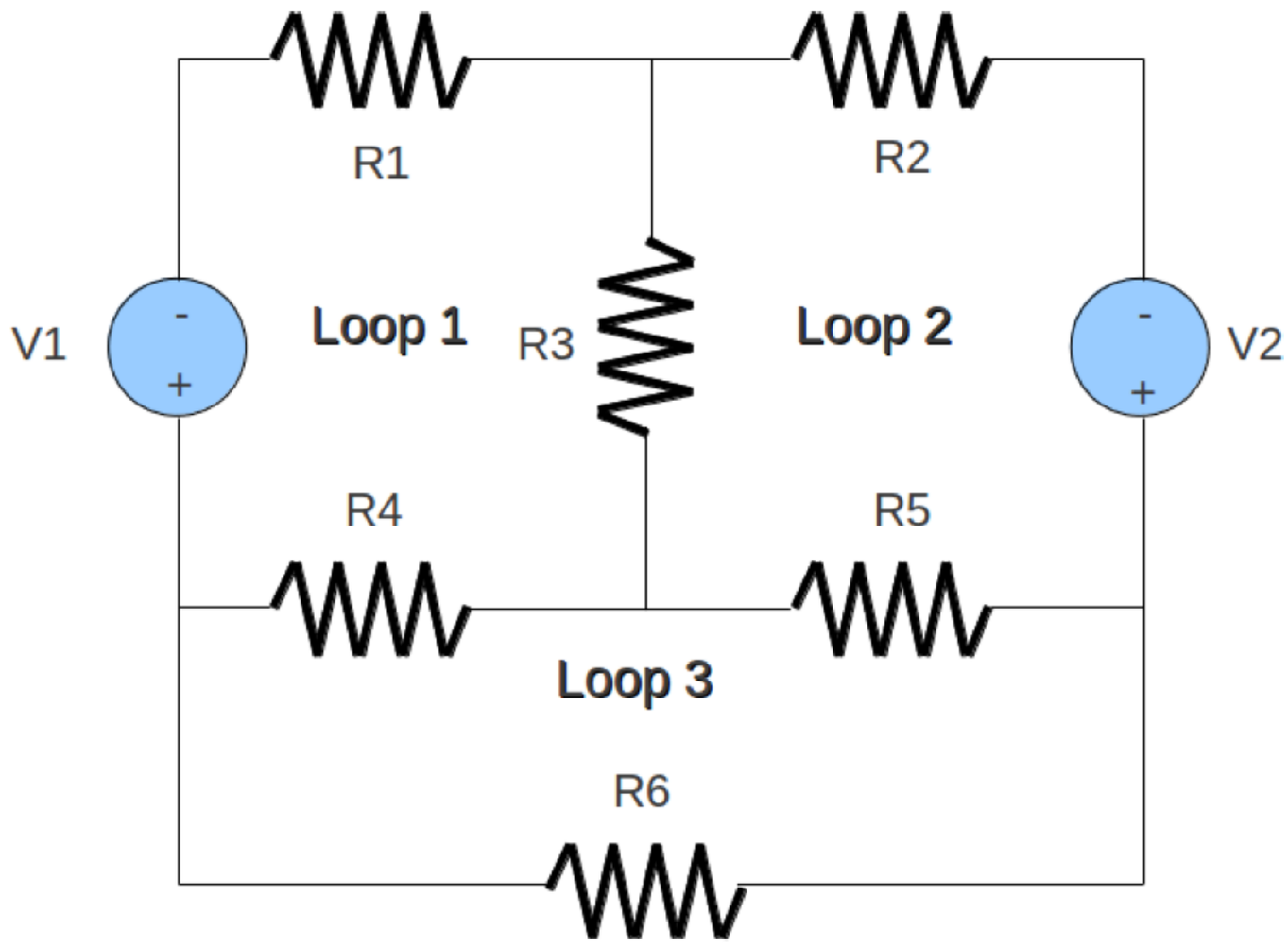
*ACM40290: Numerical Algorithms*

# Numerical Linear Algebra

Dr Barry Wardell
School of Mathematics and Statistics
University College Dublin

Ohm's Law: Voltage drop due to a current $i$ through a resistor $R$ is $V = iR$

Kirchoff's Law: The net voltage drop in a closed loop is zero



# Example: Electric Circuits

Let $i_j$ denote the current in "loop $j$"

Then, we obtain the linear system:

$$\begin{bmatrix} (R_1 + R_3 + R_4) & R_3 & R_4 \\ R_3 & (R_2 + R_3 + R_5) & -R_5 \\ R_4 & -R_5 & (R_4 + R_5 + R_6) \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ 0 \end{bmatrix}$$

Circuit simulators solve large linear systems of this type

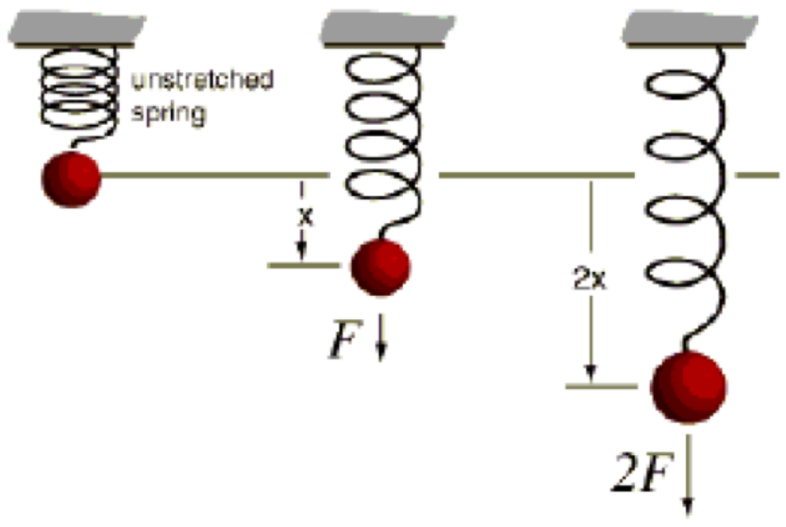
# Example: Structural Analysis

Common in structural analysis to use a linear relationship between force and displacement, Hooke's Law

Simplest case is the Hookean spring law

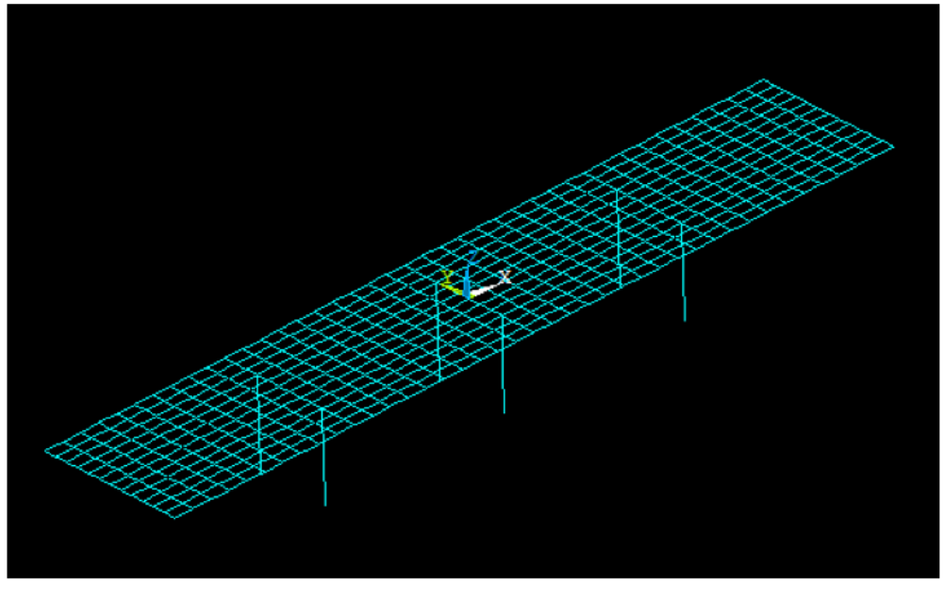
$$F = kx,$$

- $k$: spring constant (stiffness)
- $F$: applied load
- $x$: spring extension



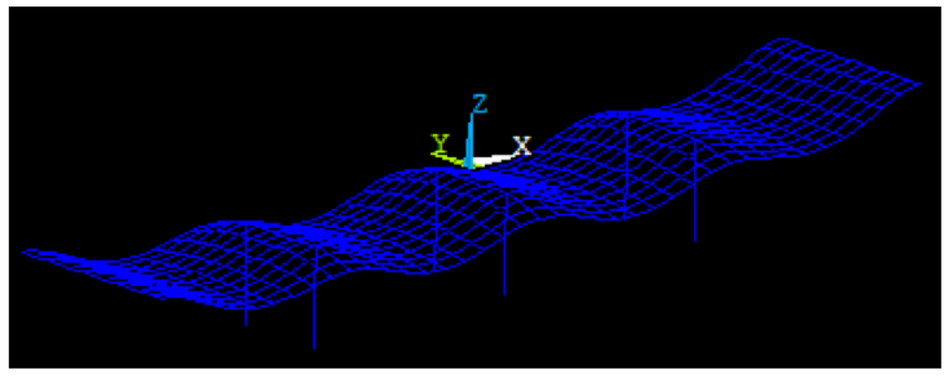
# Example: Structural Analysis

Solving the linear system yields the displacement ($x$), hence we can simulate structural deflection under applied loads ($F$)



Unloaded structure

$Kx=F$




Loaded structure

# Example: Economics

Leontief awarded Nobel Prize in Economics in 1973 for developing linear input/output model for production/consumption of goods

Consider an "economy" in which $n$ goods are produced and consumed

- $A \in \mathbb{R}^{n \times n}$: $a_{ij}$ represents amount of good $j$ required to produce 1 unit of good $i$
- $x \in \mathbb{R}^n$: $x_i$ is number of units of good $i$ produced
- $d \in \mathbb{R}^n$: $d_i$ is consumer demand for good $i$

In general $a_{ii} = 0$, and $A$ may or may not be sparse

The total amount of $x_i$ produced is given by the sum of consumer demand ($d_i$) and the amount of $x_i$ required to produce each $x_j$

$$x_i = \underbrace{a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n}_{\text{production of other goods}} + d_i$$

Hence $x = Ax + d$ or,

$$(\mathrm{I} - A)x = d$$

Solve for $x$ to determine the required amount of production of each good

If we consider many goods (e.g. an "entire economy"), then we get a large linear system

Matrix computations arise all over the place!

Numerical Linear Algebra algorithms provide us with a toolbox for performing these computations in an efficient and stable manner

In most cases, can use these tools as "black boxes", e.g. use "backslash" in Matlab to solve $Ax = b$ for you

But it's important to understand what the linear algebra "black boxes" do:

- Pick the right algorithm for a given situation (e.g. exploit structure in a problem: symmetry, bandedness, etc)
- Understand how/when the "black box" can fail

we will focus on linear systems $Ax = b$ for $A \in \mathbb{R}^{n \times n}$ and $b, x \in \mathbb{R}^n$

Recall that it is often helpful to think of matrix multiplication as a linear combination of the columns of $A$, where $x_j$ are the weights

That is, we have $b = Ax = \sum_{j=1}^{n} x_j a_{(:,j)}$ where $a_{(:,j)} \in \mathbb{R}^n$ is the $j^{\text{th}}$ column of $A$ and $x_j$ are scalars

# Preliminaries

This can be displayed schematically as

$$\begin{bmatrix} \\ b \\ \\ \end{bmatrix} = \left[\begin{array}{c|c|c|c} & & & \\ a_{(:,1)} & a_{(:,2)} & \cdots & a_{(:,n)} \\ & & & \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$= x_1 \begin{bmatrix} \\ a_{(:,1)} \\ \\ \end{bmatrix} + \cdots + x_n \begin{bmatrix} \\ a_{(:,n)} \\ \\ \end{bmatrix}$$

We therefore can interpret $Ax = b$ as: "$x$ is the vector of coefficients of the expansion of $b$ in the basis of columns of $A$"

E.g. from "linear combination of columns" view we immediately see that $Ax = b$ has a solution if

$$b \in \mathrm{span}\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$$

(this holds even if $A$ isn't square)

Let us write $\mathrm{image}(A) \equiv \mathrm{span}\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$

## Existence and Uniqueness:

Solution $x \in \mathbb{R}^n$ exists if $b \in \text{image}(A)$

If solution $x$ exists and the set $\{a_{(:,1)}, a_{(:,2)}, \cdots, a_{(:,n)}\}$ is linearly independent, then $x$ is unique.

If solution $x$ exists and $\exists z \neq 0$ such that $Az = 0$, then also $A(x + \gamma z) = b$ for any $\gamma \in \mathbb{R}$, hence infinitely many solutions

If $b \notin \text{image}(A)$ then $Ax = b$ has no solution

The inverse map $A^{-1}: \mathbb{R}^n \to \mathbb{R}^n$ is well-defined if and only if $Ax = b$ has unique solution for all $b \in \mathbb{R}^n$

Unique matrix $A^{-1} \in \mathbb{R}^{n \times n}$ such that $AA^{-1} = A^{-1}A = \mathrm{I}$ exists if any of the following equivalent conditions are satisfied:

- $\det(A) \neq 0$
- $\mathrm{rank}(A) = n$
- For any $z \neq 0$, $Az \neq 0$ (null space of $A$ is $\{0\}$)

$A$ is non-singular if $A^{-1}$ exists, and then $x = A^{-1}b \in \mathbb{R}^n$

$A$ is singular if $A^{-1}$ does not exist

# DIRECT METHODS FOR SOLVING $Ax = b$

Direct methods for solving $Ax = b$ apply elementary matrix operations to $A$ and $b$ which gives a transformed problem $T(Ax = b) = A'x' = b'$ which is easily solved for $x'$. The solution of the original problem $x$ is then found as $x = T^{-1}(x')$. This last step is not always necessary.

## Upper and Lower Triangular Systems

If $Ax = b$ has the following special form (upper triangular)

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

then these equations may by easily solved by **Back-substitution**, i.e., solve for $x_i$ in the $i$th equation, given $x_{i+1}, \ldots, x_n$. Substitute $x_i, x_{i+1}, \ldots, x_n$ in the $i-1$st equation and solve for $x_{i-1}$.

To get $x_{n-1}$ :

$$x_{n-1} = \frac{b_{n-1} - a_{n-1,n}\, x_n}{a_{n-1,n-1}}$$

**algorithm** *BackSubst* $(a, b, n, x)$

$x_n := b_n / a_{nn}$
**for** $i := n-1$ **downto** $1$ **do**
  sum $:= 0.0$
  **for** $j := i+1$ **to** $n$ **do**
    sum $:=$ sum $+ a_{ij} \times x_j$
  **endfor** $j$
  $x_i := (b_i -$ sum$) / a_{ii}$
**endfor** $i$
**endalg** BackSubst

If $Ax = b$ is lower triangular then we solve for the $x_i$'s in reverse order. This is called **Forward Substitution**.

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

The equivalent algorithm is

$$x_2 = \frac{b_2 - a_{21} x_1}{a_{22}}$$

**algorithm** *ForwSubst* $(a, b, n, x)$

```
x_1 := b_1 / a_11
for i := 2 to n do
    sum := 0.0
    for j := 1 to i - 1 do
        sum := sum + a_ij × x_j
    endfor j
    x_i := (b_i - sum) / a_ii
endfor i
```

**endalg** ForwSubst

**Analysis of Algorithm *BackSubst***

We concentrate on the inner-most loop. This has two floating point operations (flops). Hence the total number of flops is

$$\begin{aligned} S(n) &= \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} 2 = 2 \sum_{k=1}^{n-1} (n-k) = 2 \sum_{k=1}^{n-1} n - 2 \sum_{k=1}^{n-1} k \\ &= 2n(n-1) - n(n-1) = n(n-1) \approx n^2, \quad \text{for large } n. \end{aligned}$$

Hence Back Substitution is $O(n^2)$. Likewise, Forward Substitution is $O(n^2)$

$$1 + 2 + \cdots + n = \frac{n}{2}(n+1)$$

# Solving $Ax = b$

So transforming $Ax = b$ to a triangular system is a sensible goal, but how do we achieve it?

Observation: If we premultiply $Ax = b$ by a nonsingular matrix $M$ then the new system $MAx = Mb$ has the same solution

Hence, want to devise a sequence of matrices $M_1, M_2, \cdots, M_{n-1}$ such that $MA \equiv M_{n-1} \cdots M_1 A \equiv U$ is upper triangular

This process is Gaussian Elimination, and gives the transformed system $Ux = Mb$

question: How should we determine $M_1, M_2, \cdots, M_{n-1}$?

We need to be able to annihilate selected entries of $A$, below the diagonal in order to obtain an upper-triangular matrix

To do this, we use "elementary elimination matrices"

# Elementary Matrix Operations

1. Multiplying row $i$ by a scalar $c$ : $\quad R_i' := cR_i$

   This operation is equivalent to premultiplying a matrix $A$ by the following non singular matrix :

$$E_1 = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & c & & \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$$



where $E_1$ is the identity matrix with $c$ in position $(i, i)$.

2. Adding a scalar multiple of row $j$ to row $i$ : $\quad R_i' := R_i + cR_j$

   This operation is equivalent to premultiplying a matrix $A$ by the following non singular matrix :

$$E_2 = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \vdots & \ddots & \\ & & c & \cdots & 1 \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$$

where $E_2$ is the identity matrix with $c$ in position $(i, j)$

For example, with

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \quad \text{and} \quad E_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ c & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

(3,1)

Row i + c Row j

Row 3 + c Row 1

we get

$$A' = E_2 A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} + c\,a_{11} & a_{32} + c\,a_{12} & a_{33} + c\,a_{13} & a_{34} + c\,a_{14} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

3. Interchanging two rows: $R_i :=: R_j$

   This is equivalent to pre-multiplication by a the matrix $E_3$, where

$$E_3 = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & \cdots & 1 & & \\ & & \vdots & \ddots & \vdots & & \\ & & 1 & \cdots & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}.$$

$E_3$ is the identity matrix $I$ with rows $i$ and $j$ interchanged. This is an example of a **Permutation Matrix**.

# Permutation Matrices

A permutation matrix $P \in \mathbb{Z}^{n \times n}$ is a matrix with exactly one 1 in each row and column. We may interpret a permutation matrix as an identity matrix with its rows (or columns) permuted. For example, consider the permutation $p = [4213]$ and its corresponding permutation matrices $P_r$, $P_c$, and the index matrix $A$ :

$$P_r = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_c = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad \text{and} \quad A = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix}.$$

4, 2, 1, 3

If we pre-multiply $A$ by $P_r$ we get

$$P_r A = \begin{bmatrix} 41 & 42 & 43 & 44 \\ 21 & 22 & 23 & 24 \\ 11 & 12 & 13 & 14 \\ 31 & 32 & 33 & 34 \end{bmatrix}, \quad \text{which is } A \text{ with its rows interchanged.}$$

Notice that the row indices in this matrix occur in the same order as $p = [4213]$.

If we post-multiply $A$ by $P_r$ we get

$$AP_r = \begin{bmatrix} 13 & 12 & 14 & 11 \\ 23 & 22 & 24 & 21 \\ 33 & 32 & 34 & 31 \\ 43 & 42 & 44 & 41 \end{bmatrix}, \quad \text{which is } A \text{ with its columns interchanged.}$$

$AP_c$ — A with columns in permuted order [4, 2, 1, 3]

# GAUSSIAN ELIMINATION

**Example** Consider solving the following set of linear equations

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 6 \\ 2x_1 - 3x_2 + 2x_3 &= 14 \\ 3x_1 + x_2 - x_3 &= -1 \end{aligned}$$

Step 1. Eliminate $x_1$ from the 2nd and 3rd equations by subtracting 2 times the 1st equation from the 2nd, and 3 times the 1st equation from the 3rd:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 6 \\ -7x_2 - 4x_3 &= 2 \\ -5x_2 - 10x_3 &= -20 \end{aligned}$$

Step 2. Eliminate $x_2$ from 3rd equation by subtracting $5/7$ times the 2nd equation from the 3rd:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 6 \\ -7x_2 - 4x_3 &= 2 \\ -\frac{50}{7}x_3 &= -\frac{150}{7} \end{aligned}$$

We can now solve for $x_3$

$$-\frac{50}{7}x_3 = -\frac{150}{7} \quad \text{gives} \quad x_3 = 3.$$

We can now solve for $x_2$

$$-7x_2 - 4 \times 3 = 2 \quad \text{gives} \quad x_2 = -2.$$

We can now solve for $x_1$

$$x_1 + 2 \times (-2) + 3 \times 3 = 6 \quad \text{gives} \quad x_1 = 1.$$

In matrix-vector form these steps are

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 14 \\ -1 \end{bmatrix} \xrightarrow{\text{Step 1}} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & -4 \\ 0 & -5 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ -20 \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & -4 \\ 0 & 0 & -\frac{50}{7} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \\ -\frac{150}{7} \end{bmatrix}$$

We can see that these two elimination steps have transformed the original problem $Ax = b$ to a new problem $Ux = b'$, where $U$ is an upper-triangular matrix. This can now be solved using the *BackSubst* algorithm.

# Derivation of the Gaussian Elimination Algorithm

The Gaussian Elimination Method uses a sequence of elementary matrix operations to transform the square system $Ax = b$ into an upper triangular system $Ux = b'$ which is then solved using back substitution.

The method starts at stage $k = 1$ with

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{bmatrix} = \left[ A^{(1)} \quad | \quad b^{(1)} \right]$$

The coefficients $a_{21}^{(1)}, a_{31}^{(1)}, \ldots, a_{n1}^{(1)}$ below the *pivot element* $a_{11}^{(1)}$ are eliminated (reduced to zero) by subtracting the following multiples of row 1 from rows $2, 3, \ldots, n$ :

$$\frac{a_{21}^{(1)}}{a_{11}^{(1)}}, \frac{a_{31}^{(1)}}{a_{11}^{(1)}}, \ldots, \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} = m_{21}, m_{31}, \ldots, m_{n1}, \quad \text{where} \quad m_{k1} = \frac{a_{k1}^{(1)}}{a_{11}^{(1)}} \quad k = 2, 3, \ldots, n.$$

$$\underbrace{\begin{bmatrix} 1 & & & & & & \\ -M_{21} & 1 & & & & & \\ . & & 1 & & & & \\ . & & & 1 & & & \\ . & & & & 1 & & \\ . & & & & & 1 & \\ -M_{N1} & & & & & & 1 \end{bmatrix}}_{=:M^{(1)}} \begin{bmatrix} A_{11}^{(1)} & . & . & . & . & . & . & . & . & A_{1N}^{(1)} \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ A_{N1}^{(1)} & . & . & . & . & . & . & . & . & A_{NN}^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} A_{11}^{(2)} & . & . & . & . & . & . & . & . & A_{1N}^{(2)} \\ 0 & A_{22}^{(2)} & . & . & . & . & . & . & . & A_{2N}^{(2)} \\ 0 & . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & . & . & . \\ 0 & . & . & . & . & . & . & . & . & . \\ 0 & A_{N2}^{(2)} & . & . & . & . & . & . & . & A_{NN}^{(2)} \end{bmatrix} = A^{(2)} .$$

Stage 1 in GEM same as premultiplying A by matrix $M^{(1)}$.

This gives the transformed system at stage 2

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & & & & \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)} \end{bmatrix} = \left[ A^{(2)} \quad | \quad b^{(2)} \right].$$

Another way to write this (using MATLAB notation) is that the first row of $A^{(2)}$ is just same as the first row of $A^{(1)}$, the first column of $A^{(2)}$ below the diagonal is zero (and doesn't have to be stored) and the remaining bottom right $n-1 \times n-1$ block is obtained by

$$A^{(2)}(2:n, 2:n) = A(2:n, 2:n) - M(2:n, 1) * A(1, 2:n)$$

At stage $k$ we have

$$\left[\begin{array}{cccccc|c}
a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\
0 & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\
\vdots & & \ddots & \vdots & & & \\
0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} & b_k^{(k)} \\
\vdots & & \vdots & & & & \\
0 & \cdots & 0 & a_{nk}^{(k)} & \cdots & a_{nn}^{(k)} & b_n^{(k)}
\end{array}\right] = \left[\begin{array}{c|c} A^{(k)} & b^{(k)} \end{array}\right]$$

The elements $a_{k+1,k}^{(k)}, a_{k+2,k}^{(k)}, \ldots, a_{nk}^{(k)}$ are eliminated by subtracting the following multiples of row $k$ from rows $k+1, k+2, \ldots, n$:

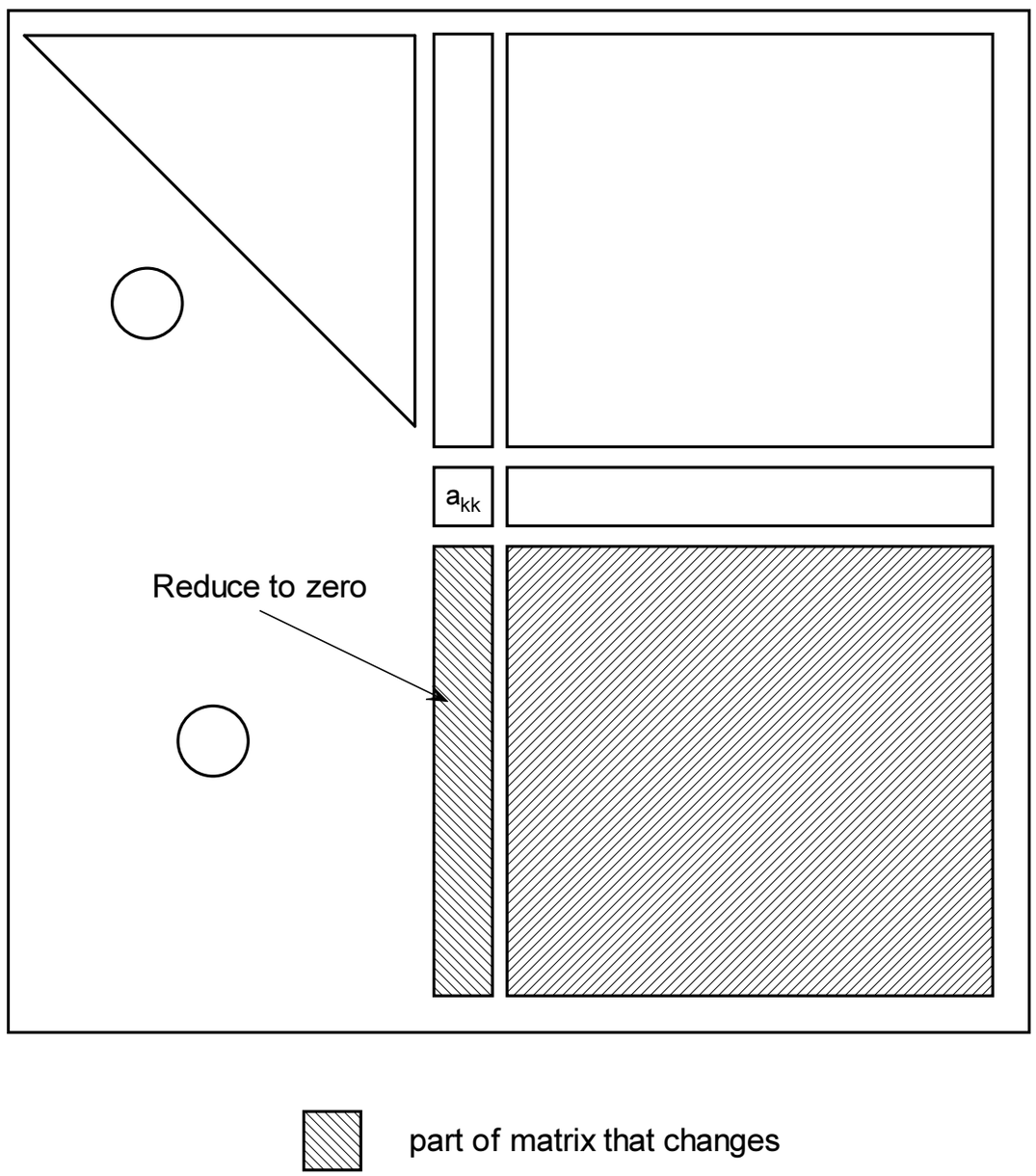
$$\frac{a_{k+1,k}^{(k)}}{a_{kk}^{(k)}}, \frac{a_{k+2,k}^{(k)}}{a_{kk}^{(k)}}, \ldots, \frac{a_{k+n,k}^{(k)}}{a_{kk}^{(k)}} = m_{k+1,k}, m_{k+2,k}, \ldots, m_{n,k}$$

We have in general, assuming $a_{kk}^{(k)} \neq 0$

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad i = k+1, \ldots, n$$

and

$$
\begin{aligned}
a_{ij}^{(k+1)} &= a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, \quad i,j = k+1,\ldots,n \\
b_{i}^{(k+1)} &= b_{i}^{(k)} - m_{ik} b_{r}^{(k)}, \quad i = k+1,\ldots,n.
\end{aligned}
$$



**algorithm** *GaussElim* $(a, b, n)$

Assume that no $a_{kk} = 0$

**for** $k := 1$ **to** $n - 1$ **do**

**for** $i := k + 1$ **to** $n$ **do**

**for** $j := k$ **to** $n$ **do**

$a_{ij} := a_{ij} - \frac{a_{ik}}{a_{kk}} \times a_{kj}$

**endfor** $j$

$b_i := b_i - \frac{a_{ik}}{a_{kk}} \times b_k$

**endfor** $i$

**endfor** $k$

**endalg** *GaussElim*

Two improvements

(1). Do not need to calculate zero elements below the diagonal

(2). Compute $\frac{a_{ik}}{a_{kk}}$ outside j loop as it is constant.

**algorithm** *GaussElim* $(a, b, n)$

Assume that no $a_{kk} = 0$

**for** $k := 1$ **to** $n - 1$ **do**

**for** $i := k + 1$ **to** $n$ **do**

$m_{ik} := a_{ik} / a_{kk}$

**for** $j := k + 1$ **to** $n$ **do**

$a_{ij} := a_{ij} - m_{ik} \times \ a_{kj}$

**endfor** $j$

$b_i := b_i - m_{ik} \times \ b_k$

**endfor** $i$

**endfor** $k$

**endalg** *GaussElim*

This algorithm transforms the original system $Ax = b$ to the following form :

$$Ux = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

This system of equations can now be solved using Back Substitution.

**Analysis of Algorithm *GaussElim***

We concentrate on the operations in the inner-most loop. We see that there are only 2 flops performed. Hence the number of steps performed is

$$\begin{aligned} S(n) &= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \sum_{j=k+1}^{n} 2 = 2 \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (n-k) \\ &= 2 \sum_{k=1}^{n-1} (n-k)^2 = 2[(n-1)^2 + (n-2)^2 + \cdots + 2^2 + 1^2] \\ &= 2n(n-1)(2n-1)/6 \approx \frac{2n^3}{3}, \quad \text{for large} \quad n. \end{aligned}$$

Hence Gaussian Elimination is an $O(n^3)$ process. Note that Matrix Multiplication requires 3 times as much work.

$S_n^2 = 1 + 2^2 + 3^2 \ldots n^2 = \frac{n(n+1)(2n+1)}{6}$

The $O(n^3)$ growth is a very fast growth of complexity. For example if $n = 10^6$ then even on a fast computer with a 1ns time for a floating point operation, $\frac{2}{3}n^3$ operations take 21 years. A lot of modern numerical linear algebra is based on trying to reduce it. For example if a matrix is banded (i.e. only a bounded number of diagonals contain non-zero entries), then the complexity of Gaussian elimination is $0(n^2)$.

**Observations on *GaussElim***

1. Assumes $a_{kk}^{(k)} \neq 0$.
2. $A$ and $b$ are overwritten.
3. The 0's beneath the pivot element are not calculated. They are ignored.
4. An extra matrix is not needed to store the $m_{ik}$'s. They can be stored in place of the zeros.
5. The operations on $b$ can be done separately, once we have stored the $m_{ik}$'s.
6. Because of 5 we may now solve for any $b$ without going through the elimination calculations again.