Brac University
Department of Electrical and Electronic Engineering
EEE282/ECE282 (V3)
Numerical Techniques
Experiment-07: Solution of Simultaneous Linear Equations

Objective

Systems of linear algebraic equations occur often in diverse fields of science and engineering and are an important area of study. In this experiment we will be concerned with the different techniques of finding the solution of a set of n linear algebraic equations in n unknowns.

Concept of linear equations and their solution

A set of linear algebraic equations looks like this:

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots a_{1N}x_N = b_1 \\ &a_{21}x_1 + a_{22}x_2 + \ldots a_{2N}x_N = b_2 \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \\ &a_{M1}x_1 + a_{M2}x_2 + \ldots a_{MN}\ x_N = b_M \end{aligned} \qquad (1)$$

Here the N unknowns x_j , $j = 1, 2, \ldots, N$ are related by M equations. The coefficients a_{ij} with $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$ are known numbers, as are the *right-hand side* quantities b_i, $i = 1, 2, \ldots, M$.

Existence of solution

If $N = M$ then there are as many equations as unknowns, and there is a good chance of solving for a unique solution set of x_j's. Analytically, there can fail to be a unique solution if one or more of the M equations is a linear combination of the others (This condition is called *row degeneracy*), or if all equations contain certain variables only in exactly the same linear combination(This is called *column degeneracy*). (For square matrices, a row degeneracy implies a column degeneracy, and vice versa.) A set of equations that is degenerate is called *singular*.
Numerically, at least two additional things can go wrong:

- While not exact linear combinations of each other, some of the equations may be so close to linearly dependent that rounding errors in the machine renders them linearly dependent at some stage in the solution process. In this case your numerical procedure will fail, and it can tell you that it has failed.
- Accumulated round off errors in the solution process can swamp the true solution. This problem particularly emerges if N is too large. The numerical procedure does not fail algorithmically. However, it returns a set of x's that are wrong, as can be discovered by direct substitution back into the original equations. The closer a set of equations is to being singular, the more likely this is to happen.

Matrices

Equation (1) can be written in matrix form as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \qquad (2)$$

Here the raised dot denotes matrix multiplication, **A** is the matrix of coefficients, **x** is the column vector of unknowns and **b** is the right-hand side written as a column vector,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Finding Solution

There are so many ways to solve this set of equations. Below are some important methods.

(1) Using the backslash and pseudo-inverse operator

In MATLAB, the easiest way to determine whether Ax = b has a solution, and to find such a the solution when it does, is to use the backslash operator. Exactly what **A** \ **b** returns is a bit complicated to describe, but if there is a solution to **A** · **x** = **b**, then **A** \ **b** returns one.
Warnings: (1) **A** \ **b** returns a result in many cases when there is no solution to **A** · **x** = **b**. (2) **A** \ **b** sometimes causes a warning to be issued, even when it returns a solution. This means that you can't just use the backslash operator: you have to check that what it returns is a solution. (In any case, it's just good common sense to check numerical computations as you do them.) In MATLAB this can be done as follows:

Using backslash operator:
x = A\b;

You can also use the pseudo-inverse operator:
x=pinv(A)*b; % it is also guaranteed to solve Ax = b, if Ax = b has a solution. As

with the backslash operator, you have to check the result.

(2) Using Gauss-Jordan Elimination and Pivoting

To illustrate the method let us consider three equations with three unknowns:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \quad \text{(A)}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = a_{24} \quad \text{(B)}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = a_{34} \quad \text{(C)}$$

Here the quantities b_i, $i = 1, 2, \ldots, M$'s are replaced by a_{iN+1}, where i=1,2, ….M for simplicity of understanding the algorithm.

The First Step is to eliminate the first term from Equations (B) and (C). (Dividing (A) by a_{11} and multiplying by a_{21} and subtracting from (B) eliminates x_1 from (B) as shown below)

$$(a_{21} - \frac{a_{11}}{a_{11}}a_{21})x_1 + (a_{22} - \frac{a_{12}}{a_{11}}a_{21})x_2 + (a_{23} - \frac{a_{13}}{a_{11}}a_{21})x_3 = (a_{24} - \frac{a_{14}}{a_{11}}a_{21})$$

Let, $\frac{a_{21}}{a_{11}} = k_2$, then

$$(a_{21} - k_2 a_{11})x_1 + (a_{22} - k_2 a_{12})x_2 + (a_{23} - k_2 a_{13})x_3 = (a_{24} - k_2 a_{14})$$

Similarly multiplying equation (A) by $\frac{a_{31}}{a_{11}} = k_3$ and subtracting from (C), we get

$$(a_{31} - k_3 a_{11})x_1 + (a_{32} - k_3 a_{12})x_2 + (a_{33} - k_3 a_{13})x_3 = (a_{34} - k_3 a_{14})$$

Observe that $(a_{21} - k_2 a_{11})$ and $(a_{31} - k_3 a_{11})$ are both zero.

In the steps above it is assumed that a_{11} is not zero. This case will be considered later in this experiment.

The above elimination procedure is called triangularization.

Algorithm for triangularization n equations in n unknowns:

```
1       for i = 1 to n and j = 1 to (n +1) in steps of 1 do read a_ij endfor
2       for k = 1 to (n −1) in steps of 1 do
3               for i = (k +1) to n in steps of 1 do
4               u ← a_ik / a_kk
5               for j = k to (n +1) in steps of 1 do
6                       a_ij ← a_ij − u a_kj endfor
                endfor
        endfor
```

The reduced equations are:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24}$$

$$a_{32}x_2 + a_{33}x_3 = a_{34}$$

The next step is to eliminate a_{32} from the third equation. This is done by multiplying second equation by

$u = a_{32} / a_{22}$ and subtracting the resulting equation from the third. So, same algorithm can be used.

Finally the equations will take the form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24}$$

$$a_{33}x_3 = a_{34}$$

The above set of equations are said to be in triangular (Upper) form.

From the above upper triangular form of equations, the values of unknowns can be obtained by back substitution as follows:

$$x_3 = a_{34} / a_{33}$$

$$x_2 = (a_{24} - a_{23}x_3) / a_{22}$$

$$x_2 = (a_{14} - a_{12}x_2 - a_{13}x_3) / a_{11}$$

Algorithmically, the back substitution for n unknowns is shown below:

```
1       x_n ← a_n(n+1) / a_nn
2       for i = (n −1) to 1 in step of -1 do
3       sum ← 0
4               for j = (i + 1) to n in steps of 1 do
5               sum ← sum + a_ij x_j endfor
6       x_i ← (a_i(n+1) − sum) / a_ii
        endfor
```

Lab Task 1. Given the simultaneous equations shown below, solve for x_1, x_2, x_3, using Gaussian Elimination.

$$2x_1\ 3x_2\ 5x_3\ 23$$
$$3x_1\ 4x_2\ x_3\ 14$$
$$6x_1\ 7x_2\ 2x_3\ 26$$

Code:

```
clear all
close all
clc

A=[2 3 5; 3 4 1; 6 7 2]; %coefficient matrix
b=[23; 14; 26]; %constant matrix

%here B=augmented upper triangular matrix; x=solutions
[B,x] = naive_gaussian_elimination(A,b)
```

Function generating code for the Naive Gaussian Elimination:

```
function [A,x] = naive_gaussian_elimination(A,b)

    %Form the augmented matrix
    A = [A b];
    [m,n] = size(A);%size of the matrix; m=row, n=column

    for i = 1:m-1%Denotes the number of steps for converting into
upper triangular matrix.
        for j = i+1:m
            A(j,i)/A(i,i)
            A(j,:) = A(j,:) - (A(i,:)/A(i,i)) * A(j,i);
            A
        end
    end

    %Back Substitution
    x = zeros(1,n-1);
    x(n-1) = A(m,n)/A(m,n-1);
    for i = n-2:-1:1
        sum  = 0;
        for j = i+1:n-1
            sum = sum + A(i,j) * x(j);
        end
        x(i) = (A(i,n) - sum)/A(i,i);
    end
end
```

Pivoting

In the triangularization algorithm we have used,

$$u \leftarrow a_{ik} / a_{kk}$$

Here it is assumed that a_{kk} is not zero. If it happens to be zero or nearly zero, the algorithm will lead to no results or meaningless results. If any of the a_{kk} is small it would be necessary to reorder the equations. It is noted that the value of a_{kk} would be modified during the elimination process and there is no way of predicting their values at the start of the procedure. The elements a_{kk} are called pivot elements. In the elimination procedure the pivot should not be zero or a small number. In fact for maximum precision the pivot element should be the largest in absolute value of all the elements below it in its column, i.e. up as the maximum of all a_{mk} where, $m \geq k$ a_{kk} should be picked.

So, during the Gauss elimination, a_{mk} elements should be searched and the equation with the maximum value of a_{mk} should be interchanged with the current position. For example if during elimination we have the following situation:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ 0.3x_2 + 4x_3 &= 5 \\ -8x_2 + 3x_3 &= 6 \end{aligned}$$

As $-8 > 0.3$, 2nd and 3rd equations should be interchanged to yield:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 4 \\ -8x_2 + 3x_3 &= 6 \\ 0.3x_2 + 4x_3 &= 5 \end{aligned}$$

It should be noted that interchange of equations does not affect the solution.
The algorithm for picking the largest element as the pivot and interchanging the equations is called pivotal condensation.

Algorithm for pivotal condensation

```
1       max ← |a_kk|
2       p ← k
3       for m = (k+1) to n in steps of 1 do
4               if (|a_mk| > max) then
5               max ← |a_mk|
6               p ← m
7               endif
        endfor
8       if (p ~= k)
9               for q = k to (n+1) in steps of 1 do
```

10 $temp \leftarrow a_{kq}$

11 $a_{kq} \leftarrow a_{pq}$

12 $a_{pq} \leftarrow temp$

endfor

endif

Lab Task 2. Modify the MATLAB program of Naive Gaussian Elimination to include pivotal condensation and singularity check.

Lab Task 3. Try to solve the following systems of equations (i) Gaussian elimination, (ii) Gaussian elimination with pivoting

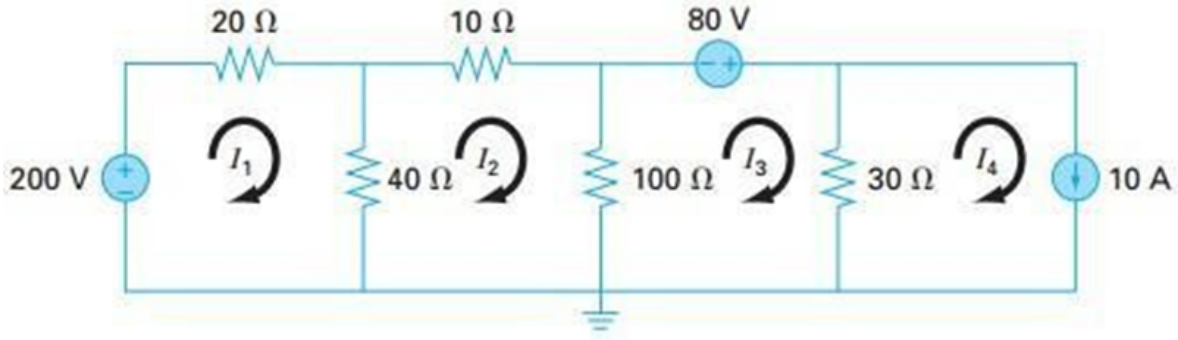
(A)
$$\begin{aligned} 2x_1 + 4x_2 - 6x_3 &= -4 \\ x_1 + 5x_2 + 3x_3 &= 10 \\ x_1 + 3x_2 + 2x_3 &= 5 \end{aligned}$$

(B)
$$\begin{aligned} x_1 + x_2 + 6x_3 &= 7 \\ -x_1 + 2x_2 + 9x_3 &= 2 \\ x_1 - 2x_2 + 3x_3 &= 10 \end{aligned}$$

(C)
$$\begin{aligned} 4x_1 + 8x_2 + 4x_3 &= 8 \\ x_1 + 5x_2 + 4x_3 - 3x_4 &= -4 \\ x_1 + 4x_2 + 7x_3 + 2x_4 &= 10 \\ x_1 + 3x_2 - 2x_4 &= -4 \end{aligned}$$

Practice Problem (Experiment 07)

Problem-1:



Find the equations of the mesh currents using Mesh Analysis first. After doing so, determine the value of the mesh currents using suitable Numerical technique.

Problem-2: Consider the following vectors:

$$\vec{A} = 2\vec{i} - 3\vec{j} + a\vec{k}$$

$$\vec{B} = b\vec{i} + \vec{j} - 4\vec{k}$$

$$\vec{C} = 3\vec{i} + c\vec{j} + 2\vec{k}$$

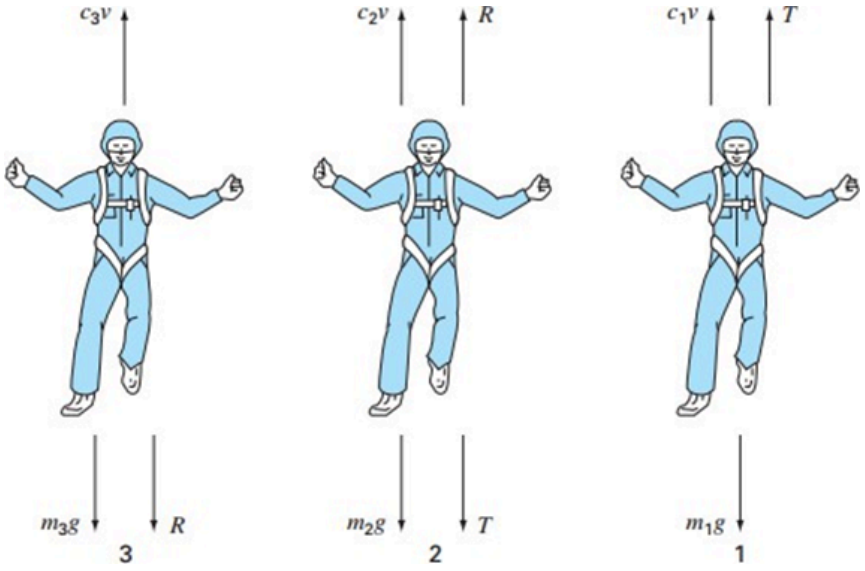
Vector $\vec{A}$ is perpendicular to $\vec{B}$ as well as to $\vec{C}$. It is also given that $\vec{B}.\vec{C} = 2$. Use a suitable Numerical technique to find the values of a, b & c.

Problem-3: Suppose that a team of three parachutists is connected by a weightless cord while free-falling at a velocity of 5 m/s.

Calculate the tension (T & R) in each section of cord and the acceleration (a) of the team, given the following:

Parachutist	Mass, kg	Drag Coefficient, kg/s
1	70	10
2	60	14
3	40	17

[Hint: Consider each of the parachutists according to the following figure:



Applying Newton's 2nd Law, the following equations can be formed:

$$m1g - T - c1v = m1a$$

$$m_2g + T - c_2v - R = m_2a$$

$$m3g - c3v + R = m3a$$

where, $m1$, $m2$, $m3$ are the masses and $c1$, $c2$, $c3$ are the drag coefficients of the three parachutists.