PRACTICE GUIDE

COMPUTATIONAL PHYSICS

BACHELOR OF PHYSICS PROGRAM
DEPARTMENT OF PHYSICS
FACULTY OF SCIENCE AND MATHEMATICS,
UNIVERSITAS DIPONEGORO

LIST OF CONTENTS

MODULE 1
NONLINEAR EQUATIONS

A. Theory:

In general, there are 2 types of ways to find solutions from nonlinear equations with numerical methods, namely bracketing/close method (closed) and open method. The bracketing method looks for the root of the equation between 2 points. Therefore, the bracketing method requires 2 initial guessing points. The Bisection and Regula Falsi method are examples of bracketing methods. In contrast to the closed method, the open method only requires 1 initial guess point. The Newton-Raphson and Secant methods are included in the open method.

1. Bisection Method

The Bisection method or commonly referred to as the middle interval method is one of the methods used to find the roots of nonlinear equations. The method estimates the solution value of a nonlinear equation contained in an interval determined by two upper and lower boundary points.

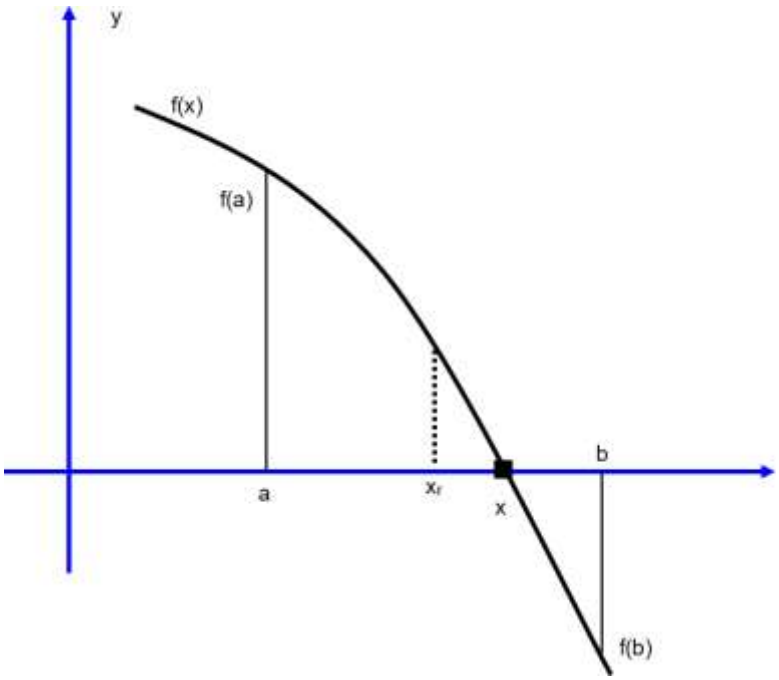


Figure 1. Bisection Method

In an area between a and b or the interval x = [a,b] on the function f(x), the bisection method has a settlement if the following conditions are met, f(a). f(b) < 0. If f(a). f(b) > 0 , or f(a) and f(b) have the same sign (negative-negative or positive-positive) then f(x)

has no solution. Then, if the condition is met, calculate the middle value (xr) of interval [a,b]

$$XR = \frac{a+b}{2} \quad (1)$$

Next, test whether the solution of the equation is between the intervals [a, xr] or [xr, b] by multiplying f(xr) by f(a) or f(b). For example, multiplied by f(a), there are 3 possibilities:

1. $f(xr) \cdot f(a) < 0$, then the solution of the equation is at interval [a, xr]
2. $f(xr) \cdot f(a) > 0$, the solution of the equation does not exist at interval [a, xr]. Unless it is present at intervals [xr, b].
3. $f(xr) \cdot f(a) = 0$, the solution of the equation is xr.

In possibilities 1 and 2, the next step is to update the interval and find the middle value (xr) of the new interval, then test again to find a solution or new interval. This step continues to be repeated until the desired error value is reached.

2. Regula Falsi Method

The Regula Falsi method is a development of the Bisection method in finding the roots of nonlinear equations by utilizing the slope and height difference of the two range boundary points.

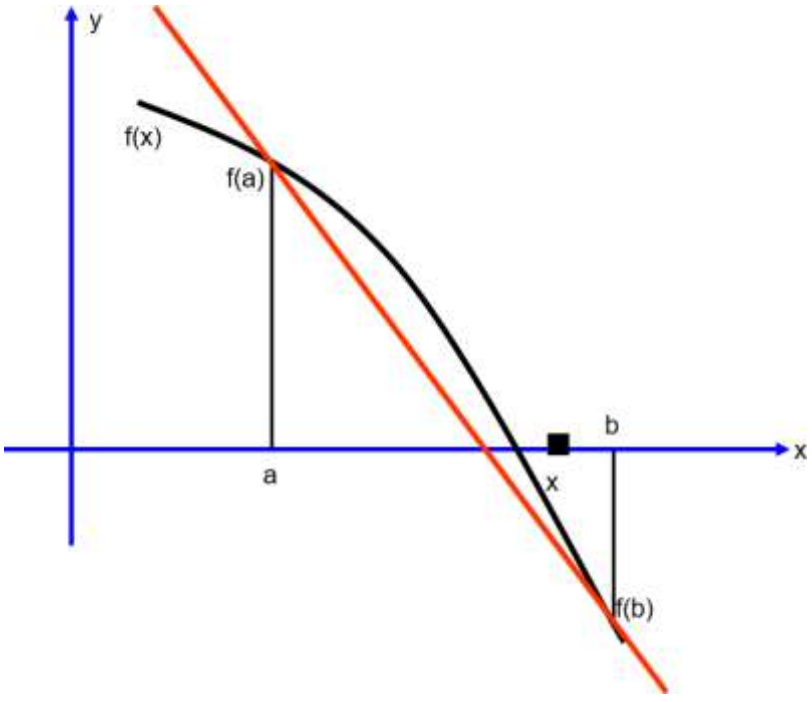


Figure 2. Regula Falsi Method

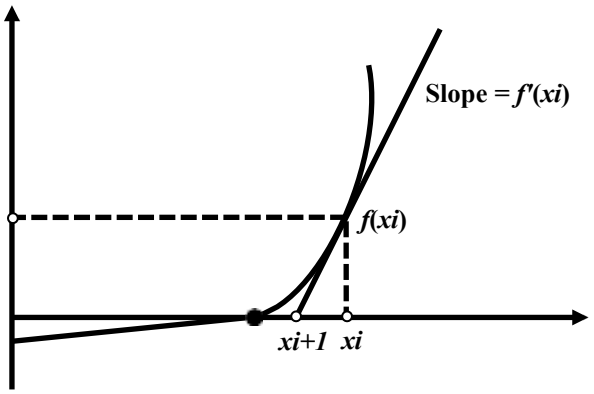
When the points f(a) and f(b) on the graph are drawn, a straight line will intersect the line on the axis x. The intersection point is the value of x, which is the solution of a nonlinear equation with the corrective method of falsi. The calculation method to find x uses the triangular formula with the formula:

$$x = \frac{f(b).A - f(a).b}{f(b) - f(a)} \qquad (2)$$

The next step is as in the Bisection method, where if the multiplication of f(x) and f(a) is less than zero, then the value of b is replaced by the value of x, which is the middle value of the interval a and b (b = x). If not, then a = x. The repeated application of these conditions always results in an estimate that is closer to the true (convergent) value.

3. The Newton-Raphson Method

The Newton-Raphson method or better known as the Newton method is the most widely used method because it has rapid convergence. If the initial guess is xi then a tangent can be drawn from the point [xi, f(xi)] cutting the xi axis which then becomes the root estimate at x_{i+1}.



Based on the geometric estimate of the figure above, the first derivative in xi is equal to the slope:

$$f(x_i) = \frac{f(x_i) - 0}{xi - x_{i+1}} \qquad (3)$$

which can be rearranged into:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad (4)$$

4. Secant Method

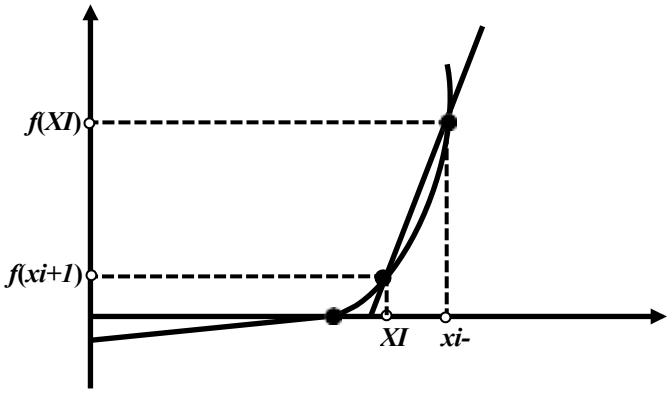
For certain functions whose derivatives are quite difficult to evaluate, to find the roots, you can use the secant method. In the Secant method, a derivative is approached with a finite *difference*, the finite *difference equation*:

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \quad (5)$$

which is then substituted into equation (4) until the following iterative equation is obtained:

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \quad (6)$$

This secant approach requires two initial assessments for x, but f(x) is not required to alternate marks between assessments.



Example :

To find the root of the following equation :

$f(x) = x^2 - 4x + 4$

Can be done by using

a. Newton Rapson's Method

With the first derivative $f'(x) = 2x - 4$, then substituting in equation (4), then:

$$x_{i+1} = x_i - \frac{x^2 - 4x + 4}{2x - 4}$$

With the initial guess of $x_0 = 0$, then the price of x up to a certain iteration can be calculated.

b. Secant Method

From equation (6), an iteration equation can be made:

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

With the initial guesses $x_{-1} = 0$ and $x_0 = 1$ then the price of x up to a certain iteration can be calculated.

B. Objectives :

After participating in this practice, participants are expected to:

1. Understand and master the matlab programming language
2. Understand and master the Bisection, Regula Falsi, Newton Repson and Secant methods

C. Practice Instructions:

In the following equation:

$$f(x) = e^{-x} - 5$$

Create a program to calculate the root of the following equations:

Programs

1. Bisection Method

```
%% Bisection
% the function is: f(x)=e^x-5
clear,clc

%% Input
f=@(x) exp(x) - 5; % fungsi yang akan dicari akar persamaannya
f0=0;              % Nilai awal fungsi untuk looping

% Looping untuk memeriksa apakah terdapat solusi persamaan di antara dua input yang dimasukkan
while f0 >= 0
    xl=input('xl=');  % Lower input
    xu=input('xu=');  % Upper input
    f0=f(xl)*f(xu);   % Perhitungan syarat bisection
    if f0 >= 0        % Jika nilai syarat bisection > 0, gunakan input lain
        disp('Use another input')
    end
end
er=100;                              % nilai awal error
targetError=input('Target error=');  % batas nilai error relatif yang ditoleransi
maxiter=100;                         % batas iterasi maksimal

%% Proses Iterasi
xb=xl;                                          % nilai x 'sebelumnya' untuk menghitung error
fprintf('iteration     xl      xu      root       error\n')   % Header tabel iterasi

for i=1:maxiter              % iterasi dijalankan sebanyak maxiter
    xr=(xl+xu)/2;            % hitung xr
    if f(xl)*f(xr) < 0       % jika syarat < 0, maka ubah nilai xu menjadi xr pada iterasi selanjutnya
        xu=xr;
    elseif f(xl)*f(xr) > 0   % jika syarat > 0, maka ubah nilai xl menjadi xr pada iterasi selanjutnya
        xl=xr;
    else                     % jika syarat = 0, maka xr adalah akar persamaan
        fprintf('The root is %f\n',xr)
        break
    end
    er = abs((xr-xb)/xr)*100;   % hitung nilai error
    xb=xr;                      % definisikan xr sebagai xb untuk iterasi selanjutnya
    fprintf('%2d          %4.2f    %4.2f    %6.4f      %6.4f%%\n',i,xl,xu,xr,er) % tabel iterasi
    if er < targetError
        fprintf('The root is %f\n',xr)
        break
    end
end
```

2. Regula Falsi

```
%% Regula-Falsi
% the function is: f(x)=e^x-5
clear,clc

%% Input
f=@(x) exp(x) - 5; % fungsi yang akan dicari akar persamaannya
f0=0;              % Nilai awal fungsi untuk looping

% Looping untuk memeriksa apakah terdapat solusi persamaan di antara dua input yang dimasukkan
while f0 >= 0
    xl=input('xl=');  % Lower input
    xu=input('xu=');  % Upper input
    f0=f(xl)*f(xu);   % Perhitungan syarat bisection
    if f0 >= 0        % Jika nilai syarat bisection > 0, gunakan input lain
        disp('Use another input')
    end
end
er=100;                              % nilai awal error
targetError=input('Target error=');  % batas nilai error relatif yang ditoleransi
maxiter=100;                         % batas iterasi maksimal
xGraph=zeros(maxiter,1);
%% Proses Iterasi
xb=xl;
fprintf('iteration     xl      xu      root        error\n')

for i=1:maxiter
    xr=xu-(f(xu)*(xl-xu))/(f(xl)-f(xu));
    xGraph(i)=xr;
    xl=xr;
    er = abs((xr-xb)/xr)*100;
    xb=xr;
    fprintf('%2d           %4.2f    %4.2f    %6.4f      %6.4f%%\n',i,xl,xu,xr,er)
    if er < targetError
        break
    end
end
```

3. Newton Repson

```
%% Newton-Raphson
% the function is: f(x)=e^x-5
clear,clc

%% Input
f=@(x) exp(x) - 5; % fungsi yang akan dicari akar persamaannya
df=@(x) exp(x);    % turunan pertama

x=input('x=');                       % nilai tebakan
targetError=input('Target Error=');  % batas error
maxiter=1000;                        % batas maksimal iterasi
er=100;                              % nilai awal error


%% Proses Iterasi
fprintf('iteration     xb       xNew         error\n')  % header tabel iterasi
xBefore=x;
for i=1:maxiter
    xNew=xBefore-f(xBefore)/df(xBefore);        % hitung nilai x 'baru' dengan persamaan Newton-Raphson
    er=abs((xBefore-xNew)/xNew)*100;            % hitung nilai error
    fprintf('%2d           %5.2f    %5.2f       %5.2f%% \n',i,xBefore,xNew,er) %tabel iterasi


    if er > targetError     % jika nilai error masih lebih besar dari target,
        xBefore=xNew;       % definisikan xNew sebagai xBefore untuk iterasi selanjutnya.
    else                    % jika nilai error < target
        fprintf('\nThe root is %f\n',xNew)
        break               % hentikan proses iterasi
    end
end
```

4. Secant

```
%% Secant
% the function is: f(x)=e^x-5
clear,clc

%% Input
f=@(x) exp(x) - 5; % fungsi yang akan dicari akar persamaannya
df=@(x) exp(x);    % turunan pertama

xBefore=input('x=');                    % nilai tebakan
delta=input('delta=');
xAfter=xBefore+delta;
targetError=input('Target Error=');  % batas error
maxiter=1000;                           % batas maksimal iterasi
er=100;                                 % nilai awal error


%% Proses Iterasi
fprintf('iteration     xb       xNew         error\n')  % header tabel iterasi

for i=1:maxiter
    xNew=xBefore-f(xBefore)*(xBefore-xAfter)/(f(xBefore)-f(xAfter));        % hitung nilai x 'baru' dengan persamaan Secan
    er=abs((xAfter-xNew)/xNew)*100;                % hitung nilai error
    fprintf('%2d          %5.2f    %5.2f       %5.2f%% \n',i,xBefore,xNew,er) %tabel iterasi


    if er > targetError     % jika nilai error masih lebih besar dari target,
        xBefore=xAfter;
        xAfter=xNew;       % definisikan xNew sebagai xBefore untuk iterasi selanjutnya.
    else                    % jika nilai error < target
        fprintf('\nThe root is %f\n',xNew)
        break               % hentikan proses iterasi
    end
end
```

Assignment :

An aircraft with a mass of 200 tons is about to land on an airport runway. If the aircraft requires a minimum landing distance of 2 km, and the coefficient of friction between the aircraft wheels and the runway is assumed to be 0.6, determine the landing velocity of the aircraft so that it stops exactly after traveling 2 km along the runway.

$$g = 9.81 \quad m/s^2$$

The equation for finding the root can be derived from the following relation:

$$W = \Delta E_{kinetic}$$
$$f_{friction} \cdot s = E_{kinetic\ \ final} - E_{kinetic\ \ initial}$$
$$\mu N s = \frac{1}{2} m v_{final}^2 - \frac{1}{2} m v_{initial}^2$$

At the final condition, the aircraft stops, therefore:

$$v_{final} = 0 \quad m/s$$
$$\mu(-mg)s = -\frac{1}{2} m v_{initial}^2$$
$$-\frac{1}{2} v_{initial}^2 + \mu g s = 0$$

Thus, the equation whose root is to be found as a function of $v_{initial}$ is:

$$f(v_{initial}) = -\frac{1}{2} v_{initial}^2 + \mu g s$$

Tasks

1. Create a flowchart of the computational procedure**.**
2. Compute the root of the equation using 2 out of the 4 numerical methods that have been studied.
3. Compare the computational results with the exact analytical solution, and explain the comparison in the discussion section.

MODULE II
LINEAR EQUATIONS

A. Theory

1a. Gauss Elimination

Review a set consisting of a number of n linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \vdots \quad \vdots \quad & \quad \vdots \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned} \tag{1}$$

The equation can be written in the form of a matrix equation

$$AX = B \tag{2}$$

With A expressing nxn coefficients, while X and B are 1xn column matrices.

Example:

Linear equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 + 2x_2 + 4x_3 &= -1 \\ x_1 + 3x_2 + 9x_3 &= 1 \end{aligned} \tag{3}$$

which can be brought into the form of a matrix:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}. \tag{4}$$

These linear equations can be solved easily if the coefficient matrix A can be brought into the form of a triangular matrix

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{13} \end{pmatrix}. \tag{5}$$

A method to bring a common matrix to a triangular shape is to use the Gauss elimination method which will be described as follows:

Take for example the matrix equation AX = B on the eq. (3)

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ 1 & 3 & 9 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}=\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} \quad (6)$$

Furthermore, the augmented equation of the matrix can be written as

$$[A\,|\,B]=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & -1 & -1 \\ 1 & 3 & 9 & 1 \end{bmatrix} \quad (7)$$

with the 4th column (or N+1) being matrix B. For example, the first row is used as a pivot. Notice the first column. It appears that the ratio of the 2nd line to the first line, $m_{21}=\frac{a_{21}}{a_{11}}=1$. For the ratio of the 3rd line to the 1st row, $m_{31}=\frac{a_{31}}{a_{11}}=1$.

The next step is the second-row component subtracted by (m_{21} × first-line component). Also, the third-row component is subtracted by (m_{31} × the first-row component. Obtained a new matrix in the form of

$$[A\,|\,B]=\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & -2 & -2 \\ 0 & 2 & 10 & 2 \end{bmatrix} \quad (8)$$

Next, using the second row in the equation matrix (8) as a pivot, the 2nd column is reviewed. It appears that the ratio of the 3rd line to the 2nd line is, $m_{32}=\frac{a_{32}}{a_{22}}=2$. Then the third row component is subtracted by (m_{32} × line component) second) until the form of a matrix is obtained

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & -2 & -2 \\ 0 & 0 & 10 & 6 \end{bmatrix} \quad (9)$$

which can be written as a triangular matrix equation $\boldsymbol{UX} = \boldsymbol{Y}$ as

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ -2 \\ 6 \end{bmatrix}$$

1b. Back Substitution

For a form of equation, the matrix ***UX = Y*** with the matrix of coefficient U is a triangular matrix

$$\begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1N} \\ & U_{22} & \cdots & U_{2N} \\ & & \ddots & \vdots \\ & & & U_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{bmatrix} \tag{11}$$

The solution for XN can be easily formulated as

$$x_N = \frac{Y_N}{U_{NN}}. \tag{12}$$

Meanwhile, for x_{N-1} it can be obtained by

$$x_{N-1} = \frac{Y_{N-1} - U_{N-1,N} x_N}{U_{N-1,N-1}} \tag{13}$$

with x_N obtained from the press. (12). Furthermore, easily x_{N-2} to x_1 can be easily derived.

More generally, the form of the solution can be formulated as

$$x_k = \frac{Y_k - \sum_{j=k+1}^{N} U_{kj} x_j}{U_{kk}} \tag{14}$$

Example:

From the eq. (10) :

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 6 \end{bmatrix}$$

||⌊0 0 10 ||||⌊x3 ⌋| ||⌊ 6 ⌋|

Using formulation on the press. (12) can be easily obtained:

$$x_3 = \frac{Y_3}{U_{33}} = \frac{6}{10} = 0.6. \tag{15}$$

Next, using the eq. (13) as well as results in the eq. (15) obtained

$$x_2 = \frac{Y_2 - U_{2,3}x_3}{U_{2,2}} = \frac{-2-(-2x0,6)}{1} = 0,8 \qquad (16)$$

Thus, x_1 can be easily obtained.

2. The Gauss-Seidel Method

The Gauss-Seidel method is a method for solving the solution of several equations. This method is the most commonly used iteration method. For the set n equation :

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = c_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = c_2$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = c_n$$

If the diagonal elements are not zero, then it can be solved by:

$$x_1 = \frac{c_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n}{a_{11}}$$

$$x_2 = \frac{c_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}}$$

$$\vdots \qquad \vdots$$

$$x_n = \frac{c_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}}{a_{nn}}$$

The completion of the iteration starts by giving an initial guess at x prices, the easiest guess is to give a guess at each variable x1 to x_n all with zero. Then this new value is substituted for the next equation along with the previous zero guesses. After all the xn values are obtained, the confluence of the guess value is sought by using the relative error criteria:

$$\left|\frac{x_i^j - x_i^{j-1}}{x_i^j}\right| \times 100\% < \varepsilon_s$$

For all i's, where J and J-1 are current and previous iterations. Example of a solution with the Gauss-Seidel method of equations:

$$7x_1 - 0,1x_2 + 5x_3 = 26$$
$$6x_1 - 12x_2 + 0,8x_3 = -12$$
$$0,9x_1 - 0,4x_2 + 14x_3 = 84$$

i.e. for each equation is rearranged

$$x_1 = \frac{26 + 0{,}1x_2 - 5x_3}{7} \quad (17)$$

$$x_2 = \frac{-12 - 6x_1 - 0{,}8x_3}{12} \quad (18)$$

$$x_3 = \frac{84 - 0{,}9x_1 + 0{,}4x_3}{14} \quad (19)$$

First iteration

With the initial guess of zero for x_2 and x_3 on equation (17) then x_1 is obtained

$$x_1 = \frac{26}{7} = 3{,}714$$

The result of x_1 above with the guess $x_3 = 0$ is subscribed to the equation (18)

$$x_2 = \frac{-12 - 6(3{,}714)}{12} = -2{,}857$$

The results of x_1 and x_2 are subscribed to equations (19)

$$x_3 = \frac{84 - 0.9(3{,}714) + 0.4(-2{,}857)}{14} = 5{,}679$$

Second iteration

Substituting x_1,x_2 and x_3 from previous iterations

$$x_1 = \frac{26 - 0.1(2{,}857) - 5(5{,}679)}{7} = -0{,}383 \qquad \text{Error} = 1068.82\%$$

$$x_2 = \frac{-12 - 6(-0.383) - 0.8(5{,}679)}{112} = -1{,}187 \qquad \text{Error} = 140.71\%$$

$$x_3 = \frac{84 - 0.9(-0.383) + 0.4(-1.187)}{14} = 5.991 \qquad \text{Error} = 5.19\%$$

Third iteration

x_1 = -0.582 Error = 34.10%

x_2 = -1,208 Error = 1.72%

x_3 = 6,002 Error = 0.20%

and so on until it is achieved at the desired price of equality

B. Purpose

After participating in this practice, participants are expected to:

1. Determine the solution of an equation by determining its variables using the Gauss and Gauss-Seidel elimination methods.
2. Knowing the accuracy of the calculation by means of analytical calculations.

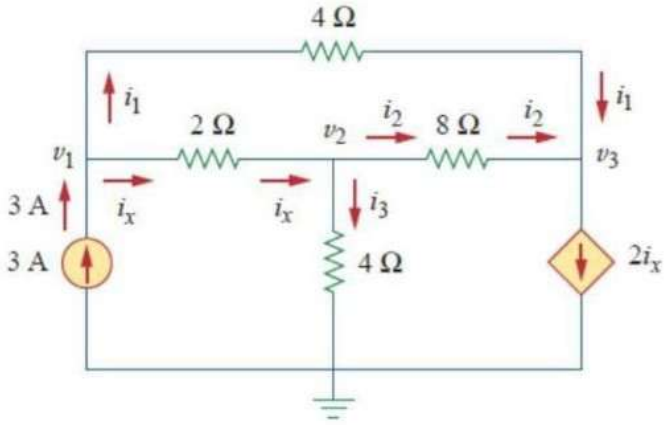
C. Practice Instructions

1. In the following equation:

Create a program to solve this with the following Gauss Elimination and back substitution methods:

```
% Gauss Back-Substitotion Method for Linear Equation
% A X = B
%% Input Matrix A and B
N=input('input the ordo=') ;%Ordo Matrix
%Define Matrix A
A=zeros(N,N); %Make All Value in Matrix A (NxN) to zero
for i=1:N
    fprintf("row %d\n",i)
    for l=1:N
        A(i,l)=input("input into the matrix=");
    end
end

%Define Matrix B
B=zeros(N,1);%Make All Value in Matrix B (Nx1) to zero
for i=1:N
    B(i,1)=input("enter the result of the equation=");
end
B
% Matrix A and Matrix B C=[A B];
%% Gauss Algorithm for the Matrix C
for p=1:N-1
    fprintf("\nLine %d", p)
    C
   for k=p+1:N;
       fprintf("\nColumn %d", k)
       %Elimination process for column p
       m=C(k,p)/C(p,p);
       fprintf("\npivot line %d against line %d = %.2f", k,p, m)
       C(k,p:N+1)=C(k,p:N+1)-m*C(p,p:N+1);
       fprintf("\nAfter operation, the %d line is minus %.2f the %d line", k, p, m)
       C
   end
end
%% Back Subtitution Algorithm
%Sprad the Matrix C become two matix (NxN) and (Nx)
A=C(1:N, 1:N);
B=C(1:N, N+1);

%Iteration for Back Subtitution
fprintf("\nback substitution")
X=zeros(N,1);
for k=N:-1:1
   X(k)=(B(k)-A(k,k+1:N)*X(k+1:N))/A(k,k);
end
X
```

D. Task



Equation 1

$$3v_1 - 2v_2 - v_3 = 12$$

Equation 2

$$-4v_1 + 7v_2 - v_3 = 0$$

Equation 3

$$2v_1 - 3v_2 + v_3 = 0$$

Determine the values of V1, V2, and V3**.**

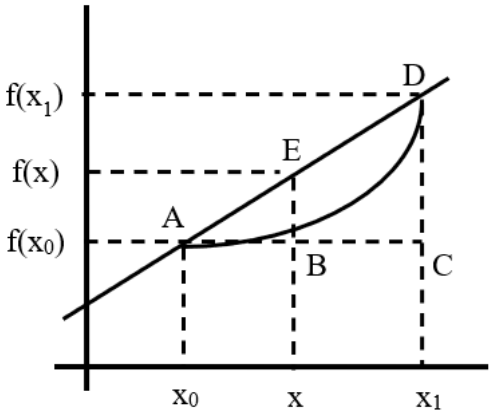
- Create a flowchart of the computational procedure**.**
- Compute the solution of the system of equations **(**V1, V2, V3**)** using the Gauss method and Back Substitution**.**
- Show the program running results along with the matrix results at each operation step in MATLAB/Octave**.**

MODULE III
INTERPOLATION

A. Theory

1. 1-order interpolation (Linear)

An 1-order or Linear interpolation approaches a function f(x) with a straight line equation.



Review the two points of the data pair $\{x_0, x_1; f(x_0), f(x_1)\}$. From the picture next to the triangle ABE and ACD gives a relationship:

$$\frac{AB}{AC} = \frac{BE}{CD}$$

or

$$\frac{x - x_0}{x_1 - x_0} = \frac{f(x) - f(x_0)}{f(x_1) - f(x_0)}$$

or

$$f(x) = f(x_0) + \frac{(f(x_1) - f(x_0))}{x_1 - x_0}(x - x_0)$$

$$= \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0} + \frac{(f(x_1) - f(x_0)}{x_1 - x_0} x$$

which can be written in polynomial form

$$P1(x) \equiv f(x) = A0 + A_1 x$$

with

$$A_0 = \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0}$$

$$A_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

2. Polynomial order N

Polynomials of the order N, $P_N(x)$, can be uniquely searched through (N+1) data points. If the data is obtained through N data points, the interpolation function of the polynomial order N-1 can be arranged. By using the Lagrange interpolation function, an interpolation of a polynomial of order N can be solved.

The N-order Lagrange interpolation function is defined as:

$$g(x) = \sum_{i=1}^{N+1} f_i \left[\frac{\prod_{n=1}^{N}{}'(x - x_n)}{\prod_{n=1}^{N}{}'(x_i - x_n)} \right]$$

$$= \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_1-x_2)(x_1-x_3)\cdots(x_1-x_N)} f_1 + \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_2-x_1)(x_2-x_3)\cdots(x_2-x_N)} f_2 + \cdots$$

$$+ \frac{(x-x_1)(x-x_2)\cdots(x-x_N)}{(x_{N+1}-x_1)(x_{N+1}-x_2)\cdots(x_{N+1}-x_N)} f_{N+1}$$

B. Purpose

After participating in this practice, participants are expected to:

1. Determining the price of a linear equation by interpolation of order 1 and order N
2. Knowing the accuracy of the calculation compared to the usual calculation method.

C. Practice Instructions

1. Polynomial order 1

The following data was obtained:

M (kg)	12	16
F (N)	40	50

If you want to know the price of *the F* force at a mass of 14 kg, then

$$A_0 = \frac{16.40 - 12.50}{16 - 12} = 10$$

$$A_1 = \frac{50 - 40}{16 - 12} = 2.5$$

So that

$F(M) = 4 + 2.5\ M$

At $M = 14$ is obtained $F(14) = 10 + 2.5\ .\ 14 = 45$

The matlab program is as follows:

```
%Coding Polynomial Order 1
% Mass    12 16 sb x
% F(style) 40 50 sb y
% Finding force on M = 14
m1=input('m1='); % enter value x1
m2=input('m2='); % mass value 2
F1=input('f1=');%force value at mass 1
F2=input('f2=');%force value at mass 1
m=input('m='); % value F to be searched with value x A0 =
((m2*F1)-(m1*F2))/(m2-m1) %formula looking for A0
```

```
A1 = (F2-F1)/(m2-m1)% search formula
A1 Fm = A0+(A1*m)% polynomial formula
FM
```

2. Polynomial order 2

Example: Review the following three pairs:

t(s)	0	20	22
and (m)	-1900	100	80

What is the price *of y* at t = 19 s ?

By using the Lagrange interpolation function for N=2, then

```
%N-Order Interpolation
N= 2;
t(1)=0;
t(2)=20;
t(3)=22;
y(1)=-1900;
y(2)=100;
y(3)=80;
tx=19;
yt=0;
%Algoritm
for j=1:N+1
  r1=1;
  r2=1;
  for k=1:N+1
    if j~=k
    r1=r1*(tx-t(k));
    r2=r2*(t(j)-t(k));
    end
  end
  yt=yt+r1*y(j)/r2;
End
YT
```

D. Duties

INTERPOLATION ASSIGNMENT

The table value of the graph of the relationship between time (x) and acceleration f(x) is known as follows:

No	x	f(x)
1	0	0
2	1	1
3	2	2.83
4	3	4.43
5	4	5.66
6	5	6.43
7	6	6.64
8	7	6.23
9	8	5.23
10	9	3.66
11	10	1.66
12	11	-0.57
13	12	-2.57
14	13	-3.93
15	14	-4.57
16	15	-4.43
17	16	-3.57
18	17	-2.07
19	18	-0.07
20	19	2.03

Determine the value of f(x) at a given value of x**.**

- Use first-order and second-order interpolation, where x is obtained from the last two digits of the student ID number (NIM) reversed in decimal form (for example, if the digits are 35, then x = 3.5 and x = 5.3).
- Create an interactive MATLAB script and provide explanations/comments within the code.
- Show the program running results and explain the difference between the results obtained using first-order and second-order interpolation.

MODULE IV
DERIVATIVES

A. Theory

With the Taylor series, a function *f*(*x*) can be guessed approximately through the price *f(x0)* and derivatives of *the f(x)* function expressed as:

$$f(x) = f(x_0) + xf'(x) + \frac{x^2}{2!}f''(x) + \frac{x^3}{3!}f'''(x) + \ldots$$

If a function *f*(*x*) is described as shown in figure 1 below, and the function is divided into segments on the x-axis where the distance between segments is *h*, then the relationship can be shown:

$$f_1 - f_{-1} = 2hf' + \frac{2h^3}{6}f''' + O(h)$$

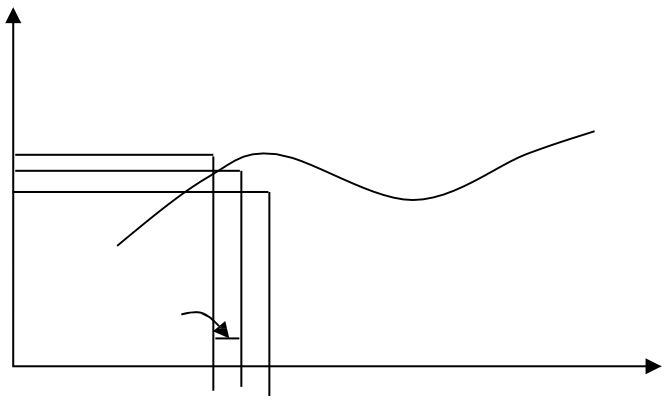


Figure 1. The function *f*(*x*) is divided into segments So that it can be formulated that:

- First Derivative

With the centralized approach method, the following are obtained:

$$f' = \frac{f_1 - f_{-1}}{2h}$$

Or by using double precision for the calculated variables obtained

$$f' = \frac{1}{12h}[f_{-2} - 8f_{-1} + 8f_1 - f_2]$$

- Second Derivative

$$f'' = \frac{f_1 - 2f_0 + f_{-1}}{h^2}$$

or

$$f'' = \frac{1}{12h^2}[-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2]$$

- Third Derivative

$$f''' = \frac{-f_{-2} + 3f_0 - 3f_1 + f_2}{h^3}$$

or

$$f''' = \frac{-f_{-2} + 2f_{-1} - 2f_1 + f_2}{2h^3}$$

- Fourth Derivative

$$f'''' = \frac{f_{-2} - 4f_{-1} + 6f_0 - 4f_1 + f_2}{h^4}$$

B. Purpose

After participating in this practice, participants are expected to:

1. Solve the derivative of an equation f(x) from the first derivative to the fourth derivative numerically.
2. Knowing the accuracy of the decline in an analytical way

C. Practice Instructions

The similarities are known from:

$f(x) = 2x^3 + 4x^2 + 6$

The above equation is derived from x, at $x = 3$, precisely calculable that

$f'(x) = (6x^2 + 8x)|_{x=3} = 78$

If h = 0.1 is selected, then edit the program to complete the first derivative of The equation $f(x)$ to x is as follows:

```
%% Initialization of Functions and Parameters
% Create Function
fx = @(x) 10 * x^5 + 18 * x^4 + 28 * x^3 + 7 * x^2 + 6;

% Parameter Initialization
x0 = 3; % Point at which derivatives are
evaluated exactly = 6792; % Exact derivative
value at point x0 nilai_h = 0; % Initial value
for h
delta_h = 0.1; % Increment added to h value in each iteration num_iter = 10; %
Number of iterations

%% Near Centralized Difference fprintf('Near Centralized Difference\n');
```

```
% Create an array to store h values, derivative results, and errors of centralized
difference h_terpusat = zeros(1, num_iter);
hasil_turunan_terpusat = zeros(1, num_iter);
error_terpusat = zeros(1, num_iter);

% Calculates the derivative value and error for the variation of the value h plus
delta_h for i = 1:num_iter
    h_terpusat(i) = nilai_h + i * delta_h;
    diff = (fx(x0 + h_terpusat(i)) - fx(x0 - h_terpusat(i))) / (2 * h_terpusat(i));
    error_terpusat(i) = abs((diff - exact) / exact) * 100;
    hasil_turunan_terpusat(i) = diff;
end

% Displays a table of results near centralized differences
fprintf('----------------------------------------------------------------------------------------------------\n');
fprintf('    h            Derivative Values        Error\n');
fprintf('----------------------------------------------------------------------------------------------------\n');
for i = 1:num_iter
    fprintf('%10.6f    %15.6f    %15.6f\n', h_terpusat(i), hasil_turunan_terpusat(i),
error_terpusat(i));
end
fprintf('----------------------------------------------------------------------------------------------------\n');

% Display a graph of derived values near the centralized
difference of the figure; % Create a new figure
plot(h_terpusat, hasil_turunan_terpusat, 'b-'); % Plot of title derivative value
('Approximate Centralized Difference of Order 1 - Numerical Derivative Value'); %
Graphic title xlabel('h'); % x axis label
ylabel('Derivative Value'); % Axis label
and grid on; % Displays grid

% Display error graph near the centered difference
of the figure; % Create a new figure
plot(h_terpusat, error_terpusat, 'r--'); % Plot of title derivative values
('Near the Centralized Difference of Order 1 - Error Graph'); % Graphic
title xlabel('h'); % x axis label
ylabel('Error'); % Axis label and
grid on; % Displays grid

%% Near Forward Difference fprintf('Near
Forward Difference\n');

% Create an array to store the h-value, derivative result, and error of the forward
difference h_maju = zeros(1, num_iter);
hasil_turunan_maju = zeros(1, num_iter);
error_maju = zeros(1, num_iter);

% Calculates the derivative value and error for the variation of the value h plus
delta_h for i = 1:num_iter
    h_maju(i) = nilai_h + i * delta_h;
    diff = (fx(x0 + h_maju(i)) - fx(x0)) / h_maju(i);
    error_maju(i) = abs((diff - exact) / exact) * 100;
    hasil_turunan_maju(i) = diff;
end

% Displays the result table near the forward difference
fprintf('----------------------------------------------------------------------------------------------------\n');
fprintf('    h            Derivative Values        Error\n');
fprintf('----------------------------------------------------------------------------------------------------\n');
for i = 1:num_iter
    fprintf('%10.6f    %15.6f    %15.6f\n', h_maju(i), hasil_turunan_maju(i), error_maju(i));
end
```

```
fprintf('-------------------------------------------------------------------------------------------\n');

% Display a graph of derived values near the forward
difference of the figure; % Create a new figure
plot(h_maju, hasil_turunan_maju, 'b-'); % Derivative value plot of the forward difference
approximation method
title('Near the Forward Difference of Order 1 - Numerical Derivative Values'); %
Graphic title xlabel('h'); % x axis label
ylabel('Derivative Value'); % Axis label
and grid on; % Displays grid

% Display error graph near the forward difference
figure; % Create a new figure
plot(h_maju, error_maju, 'r--'); % Plot error from  the title forward
difference method ('Forward Difference Near Order 1 - Error Graph'); % Graphic
title xlabel('h'); % x axis label
ylabel('Error'); % Axis label and
grid on; % Displays grid

%% Near Backward Difference
fprintf('Near Backward Difference\n');

% Create an array to store h values, derivative results, and errors of centralized
difference h_mundur = zeros(1, num_iter);
hasil_turunan_mundur = zeros(1, num_iter);
error_mundur = zeros(1, num_iter);

% Calculates the derivative value and error for the variation of the value h plus
delta_h for i = 1:num_iter
    h_mundur(i) = nilai_h + i * delta_h;
    diff = (fx(x0) - fx(x0 - h_mundur(i))) / h_mundur(i);
    error_mundur(i) = abs((diff - exact) / exact) * 100;
    hasil_turunan_mundur(i) = diff;
end

% Show results table near the drawback difference
fprintf('-------------------------------------------------------------------------------------------\n');
fprintf('    h            Derivative Values        Error\n');
fprintf('-------------------------------------------------------------------------------------------\n');
for i = 1:num_iter
    fprintf('%10.6f    %15.6f    %15.6f\n', h_mundur(i), hasil_turunan_mundur(i),
error_mundur(i));
end
fprintf('-------------------------------------------------------------------------------------------\n');

% Display a graph of derived values near the deviation of
the figure; % Create a new figure
plot(h_mundur, hasil_turunan_mundur, 'b-'); % Plot of derived values from the regression
approximation method
title('Approximate Backward Difference Order 1 - Numerical Derivative Value'); % Graphic
title xlabel('h'); % x axis label
ylabel('Derivative Value'); % Axis label
and grid on; % Displays grid

% Display error graph near the deviation difference
figure ; % Create a new figure
plot(h_mundur, error_mundur, 'r--'); % Plot error from the title reverse difference
approximation method ('Approximate Reverse Difference Order 1 - Error Graph'); % Chart
title
xlabel('h'); % x-axis label
ylabel('Error'); % Axis label and
grid on; % Displays grid

%% Plot graph of derivative values of all methods in one graph
```

```
figure; % Create a new figure
hold on; % Starts depicting data without deleting existing ones
plot(h_terpusat, hasil_turunan_terpusat, 'b-', 'DisplayName', 'Near Centralized Difference');
% Plot of derived values from the centralized difference approximation method
plot(h_maju, hasil_turunan_maju, 'r-', 'DisplayName', 'Forward Difference Approach'); %
Derivative value plot of the forward difference approximation method
plot(h_mundur, hasil_turunan_mundur, 'g-', 'DisplayName', 'Near Backward Difference'); %
Derivative value plot of the title reverse approximation
method ('Derivative Value Graph - All Methods'); % Graphic
title xlabel('h'); % x axis label
ylabel('Derivative Value'); % Y-axis label
legend('Location', 'best'); % Displays legends in the best location of
the grid on; % Displays grid
hold off; % Ends data depiction and activates normal depiction mode

%% Plot graph error all methods in one graph
figure; % Create a new figure
hold on; % Starts depicting data without deleting existing ones
plot(h_terpusat, error_terpusat, 'b-', 'DisplayName', 'Near Centralized Difference'); %
Error plot of the method near centralized difference
plot(h_maju, error_maju, 'r-', 'DisplayName', 'Forward Difference Approach'); % Plot error
of the forward difference approximation method
plot(h_mundur, error_mundur, 'g-', 'DisplayName', 'Near Backward Difference'); % Plot error
of the reverse difference approximation method
title('Error Graph - All Methods'); % Graphic title
xlabel('h'); % x axis label
ylabel('Error'); % Y-axis label
legend('Location', 'best'); % Displays legends in the best location of
the grid on; % Displays grid
hold off; % Ends data depiction and activates normal depiction mode
```

D. Assignments

You are studying the motion of a particle moving in a medium. The motion of the particle is described by a quadratic function representing the particle's position as a function of time:

$$x(t) = A \cdot t^3 + B \cdot t^2 - C \cdot t + D$$

Where:

A = (Last digit of Student ID × Second-to-last digit of Student ID) + Second-to-last digit of Student ID
B = Last digit of Student ID + Second-to-last digit of Student ID
C = Last digit of Student ID
D = Any arbitrary value

Example: If the Student ID (NIM) is 24040121130084, then 8 is the second-to-last digit and 4 is the last digit.

Task

Solve the following problems using MATLAB.

1. Calculate the particle velocity $v(t)$and its error using central difference, forward difference, and backward difference approximations at
   $t = |\texttt{Last digit of Student ID} - \texttt{Second-to-last digit of Student ID}|$

Perform iterations for h = 0.1 up to h = 1.
After that, plot the derivative graph and the error graph of all methods in a single plot.
Then, compare the three methods and provide an explanation.

2. Calculate the particle acceleration $a(t)$and its error using central, forward, and backward difference approximations at
   $t = |\texttt{Last digit of Student ID} - \texttt{Second-to-last digit of Student ID}|$

Use h = 0.1.
Compare the three methods and provide an explanation.

3. Solve this problem analytically and compare the result with the numerical computation obtained from your MATLAB code.

MODULE V
ORDINARY DIFFERENTIAL EQUATION (ODE) 1st ORDE

A. Theory

The common form of ordinary differential equations (Ordinary Differential Equations = ODE) of the first order is

$$\frac{dy}{dt} = f(t, y(t))$$

with order refers to the derivative degree in the left-hand quarter ($\frac{dy}{dt}$), while The function $f(x,y(t))$ in the right quarter can be arbitrary.

The solution of the first-order differential equation uses several methods, including:

a. Euler Method

By taking the initial condition $y(x_0) = y_0$, then done with the formula:

$y_{n+1} = y_n + hf(x_n,y_n)$

where h is the distance between the segments, the number of N segments

The main Program of this method is

```
for i = 1:N
y = y + f(i*h,y)
end
```

b. The Taylor Series Method

This method is in the form of an equation:

$y_{n+1} = y_n + h*f(i*h,y) + h^2 [\partial f/\partial x + f \partial f/\partial y]/2$

The main program of this method is

```
for i = 1 : N
y: = y + h*fxy(i*h,y)+ h^2(dfx(y)+ f(i*h,y)*dfy(i*h)+)/2
end
```

c. Runge Kutta Method

1. order I : $y_{n+1}=y_n + hf(x_n,y_n)$

(same as Euler's method)

2. Order II: $y_{n+1}=y_n + hf(x_n+1/2h, y_n + 1/2k)$

where $k = h\,f(t_n, y_n)$

The main program of this method is

```
for i= 1 : N
k= h * f(i*h,y)
y=y+h*f((i*h)+0.5*h,y+0.5*k);
end
```

3. Order III : $y_{n+1}=y_n + \frac{1}{6}(k_1+4k_2+k_3)$

   where $k_1 = h\,f(x_n, y_n)$

   $k_2 = H\,f(x_n+1/2h, y_n+1/2k_1)$

   $k_3 = h\,f(x_{n+h}, y_n - k_1+2k_2)$

   The main program of this method is

```
for i= 1 : N
k1= h * f(i*h,y);
k2= h * f((i*h)+0.5*h,y+0.5*k1);
k3= h * f((i*h)+h,y-k1+2*k2);
y=y+h*(k1+4*k2+k3)/6;
end
```

4. Order IV : $y_{n+1}=y_n + \frac{1}{6}(k_1+2k_2+2k_3+k_4)$

   where $k_1 = h\,f(x_n, y_n)$

   $k_2 = h\,f(x_n+1/2h, y_n+1/2k_1)$

   $k_3 = h\,f(x_n+1/2h, y_n+1/2k_2)$

   $k_4 = h\,f(x_n+h, y_n + k_3)$

   The main program of this method is

```
for i= 1 : N do
begin
k1= h * f(i*h,y);
k2= h * f((i*h)+0.5*h,y+0.5*k1);
k3= h * f((i*h)+0.5*h,y+0.5*k2);
k4= h * f((i*h)+h,y+k3);
y=y+h*(k1+2k2+2*k3+k4)/6;
end;
```

B. Purpose

After participating in this practice, participants are expected to

1. Solve first-order differential equations with Euler, Taylor Series, Runge Kutta Orders I,II,III & IV methods
2. Knowing the accuracy of each method by comparing the results of its integration with an analytical way

C. Practice Instructions

Example :

$$\frac{dy}{dx} = -xy$$

with the initial condition $y(0)=0$, then the completion of the program for the Taylor Series method

```
% For example, we have the form of differential equation as follows:
% dy/dx + xy = 0
% The solution of the differential equation is y as a function x [ y(x) ]
% With the analytical method, the solution of the above differential equation can be
% is obtained by integrating the equation

% For numerical methods, there are several methods that can be used to
% get a differential equation solution:
% 1. With the Taylor series
% 2. With Euler's method
% 3. Runge-Kutta Method

% The three methods have the same principle, which is to carry out the process
% iteration to obtain a value y at the next point based on the value
% x and y at the previous point, then the values of x & y that have been
The % obtained will be plotted on the graph until it forms a curve

% In the previous equation, to obtain a PD solution we need
% is an initial coordinate as the starting point. For example, x0 = -3; y0 = 0.
%    i   |   x   |   y
% ---------------------------------------
%    0   |  -3   |   0
%    1   | x0+h  |  ...
%    2   | x1+h  |  ...
%    3   | x2+h  |  ...
%  etc.  |  ...  |  ...

%% ==============================Taylor Row==============================
% Taylor's series equation:
% y(x_setelah) = y(x_sebelum) + h * y'(x_sebelum) + h^2/2 + y''(x_sblm) ...
% dy/dx + x*y = 0
% dy/dx = y'(x) = -x*y
% y''(x) = -y - x^2*y
clear,clc
df=@(x,y) -x*y;
df2=@(x,y) -y + y*x^2;

% Input
a = -3;   % Interval Lower Limit
b = 3;    % Upper Limit Interval h = 0.1; % step
```

```
y = 0.01;
```

```
N = round((b-a)/h);            % total
iterations xx=zeros(N,1);
         yy=zeros(N,1);

         for i = 1:N % Loop to perform numerical integration
             x = a+i*h;
             y = y + df(x,y)*h + df2(x,y)*(h^2)/2;
             %fprintf('%.4f    %.4f \n',x,y)
             yy(i) = y; % Stores y-value on each iteration
             xx(i) = a + i*h; % Stores x values on each iteration
         end

         figure;
         plot(xx, yy, 'r--'); % Create a numeric solution plot
         title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
         xlabel('x'); % Add x-axis labels
         ylabel('y'); % Add y-axis labels

         %% ============================Euler's Method==============================
         % Euler method equation ≈ Taylor series
         % y(x_setelah) = y(x_sebelum) + h * f(x_sebelum,y_sebelum)
         % dy/dx + x*y = 0
         % dy/dx = y'(x) = -x*y

         clear,clc
         f=@(x,y) -x*y;

         % Input
         a = -3;   % Interval Lower Limit
         b = 3;    % Upper Limit
         Interval h = 0.1; % step
         y = 0.01;
         N = round((b-a)/h);   % total
         iterations xx=zeros(N,1);
         yy=zeros(N,1);

         for i = 1:N % Loop to perform numerical integration
             x = a+i*h;
             y = y + h*f(x,y);
             fprintf('%.4f    %.4f \n',x,y)
             yy(i) = y; % Stores the value y on each iteration xx
             (i) = x; % Stores x values on each iteration
         End
         Figure(2)
         plot(xx, yy, 'r--'); % Create a numeric solution plot
         title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
         xlabel('x'); % Add x-axis labels
         ylabel('y'); % Add y-axis labels


         %% ============================Runge-Kutta==================================
         % The Runge-Kutta order method has the following order:
         % 1. Order 1: y(x_[i+1]) = y(x_i) + k1
         %    where k1 = h*f(x_i, y_i)
         %    Runge-Kutta order 1 = Euler's method
         %
         % 2. Order 2: y(x_[i+1]) = y(x_i) + (1/2)*(k1+k2)
         %    where k1 = h*f(x_i,y_sblm)
         %           k2 = h*f(x_i + h, y_i + k1)
         %
         % 3. Order 3: y(x_[i+1]) = y(x_i) + (1/6)*(k1 + 4*k2 + k3)
         %    where k1 = h*f(x_i,y_i)
         %           k2 = h*f(x_i + h/2, y_i + k1/2)
```

```
%           K3 = h*f(x_i + h, y_i - k1 + 2*k2)
%
% 4. Order 4: y(x_[i+1]) = y_i + (k1 + 2*k2 + 2*k3 + k4)
%    where k1 = h*f(x_i,y_i)
%           k2 = h*f(x_i + h/2, y_i + k1/2)
%           K3 = h*f(x_i + h/2, y_i + k2/2)
%           k4 = h*f(x_i + h, y_i + k3)

%% ============================== RK Order 1 ================================
% dy/dx + x*y = 0
% dy/dx = y'(x) = -x*y

clear,clc
f=@(x,y) -x*y;

% Input
a = -3;   % Interval Lower Limit
b = 3;    % Upper Limit
Interval h = 0.0001; % step
y = 0.01;
N = round((b-a)/h);   % total
iterations xx=zeros(N,1);
yy=zeros(N,1);

for i = 1:N % Loop to perform numerical integration
    x = a+i*h;
    k1 = h*f(x,y);
    y = y + k1;
    fprintf('%.4f    %.4f \n',x,y)
    yy(i) = y; % Stores the value y on each iteration xx
    (i) = x; % Stores x values on each iteration
End
Figure(3)
plot(xx, yy, 'r--'); % Create a numeric solution plot
title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Add y-axis labels

%% ============================== RK Order 2 ================================
% dy/dx + x*y = 0
% dy/dx = y'(x) = -x*y

clear,clc
f=@(x,y) -x*y;

% Input
a = -3;   % Interval Lower Limit
b = 3;    % Upper Limit
Interval h = 0.0001; % step
y = 0.01;
N = round((b-a)/h);   % total
iterations xx=zeros(N,1);
yy=zeros(N,1);

for i = 1:N % Loop to perform numerical integration
    x = a+i*h;
    k1 = h*f(x,y);
    k2 = h*f(x + h, y +k1);
    y = y + (1/2)*(k1 + k2);
    fprintf('%.4f    %.4f \n',x,y)
    yy(i) = y; % Stores the value y on each iteration xx
    (i) = x; % Stores x values on each iteration
end
```

```
Figure(1)
plot(xx, yy, 'r--'); % Create a numeric solution plot
title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Add y-axis labels

%% ============================== RK Order 3 ================================
% dy/dx + x*y = 0
% dy/dx = y'(x) = -x*y

clear,clc
f=@(x,y) -x*y;

% Input
a = -3;   % Interval Lower Limit
b = 3;    % Upper Limit
Interval h = 0.0001; % step
y = 0.01;
N = round((b-a)/h);   % total
iterations xx=zeros(N,1);
yy=zeros(N,1);

for i = 1:N % Loop to perform numerical integration
    x = a+i*h;
    k1 = h*f(x,y);
    k2 = h*f(x + 0.5*h, y + 0.5*k1);
    k3 = h*f(x + h, y - k1 + 2*k2);
    y = y + (1/6)*(k1 + 4*k2 + k3);
    fprintf('%.4f    %.4f \n',x,y)
    yy(i) = y; % Stores the value y on each iteration xx
    (i) = x; % Stores x values on each iteration
End
Figure(1)
plot(xx, yy, 'r--'); % Create a numeric solution plot
title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Add y-axis labels

%% ============================== RK Order 4 ================================
clear,clc
f=@(x,y) -x*y;

% Input
a = -3;   % Interval Lower Limit
b = 3;    % Upper Limit
Interval h = 0.0001; % step
y = 0.01;
N = round((b-a)/h);   % total
iterations xx=zeros(N,1);
yy=zeros(N,1);

for i = 1:N % Loop to perform numerical integration
    x = a+i*h;
    k1 = h*f(x,y);
    k2 = h*f(x + 0.5*h, y + 0.5*k1);
    k3 = h*f(x + 0.5*h, y + 0.5*k2);
    k4 = h*f(x + h, y + k3);
    y = y + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
    fprintf('%.4f    %.4f \n',x,y)
    yy(i) = y; % Stores the value y on each iteration xx
    (i) = x; % Stores x values on each iteration
End
Figure(1)
```

```
plot(xx, yy, 'r--'); % Create a numeric solution plot
title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Add y-axis labels

%%Plot All Method

yy_taylor = zeros(N,1);
yy_euler = zeros(N,1);
yy_runge = zeros(N,1);
yy_eksak = zeros(N,1);
fungsi_eksak = @(x) exp(-x^2/2);

%Count y
= 0.01;
for i = 1:N % Loop to perform numerical integration
    x = a+i*h;
    y_taylor = y + df(x,y)*h + df2(x,y)*(h^2)/2; yy_taylor(i)
    = y; % Stores the value of y on each iteration y_euler =
    y + h*f(x,y);
    yy_euler(i) = y_euler;
    k1 = h*f(x,y);
    k2 = h*f(x + 0.5*h, y + 0.5*k1);
    k3 = h*f(x + h, y - k1 + 2*k2);
    y_runge = y + (1/6)*(k1 + 4*k2 + k3);
    yy_runge(i) = y_runge;
    y_eksak = fungsi_eksak(x);
    yy_eksak(i) = y_eksak;
    xx(i) = x; % Stores x values on each iteration
end

figure;
hold on;
title('Differential Equation Solution Graph dy/dx = -xy'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Add y-axis labels

% Taylor's series equation:
plot(xx, yy_taylor, 'r--', 'DisplayName','Taylor'); % Create a numeric solution plot

% Euler equation:
plot(xx, yy_euler, 'b--','DisplayName','Euler'); % Create a numeric solution plot

% Runge Kutta Equation Order 4:
plot(xx, yy_runge, 'k--', 'DisplayName','Runge'); % Create a numeric solution plot

% Exact Solution:
plot(xx, yy_eksak, 'g--','DisplayName','Exact'); % Create a numeric solution plot

legend('Location','best');
grid on;
hold off;
```

D. Assignment

Given the equation of air drag force on a freely falling object:

$$\frac{dv}{dt} + g - \frac{c}{m}\left(v + \alpha\left(\frac{v}{v_{max}}\right)^{\beta}\right) = 0$$

Where:

- α= the last two digits of the Student ID (NIM)
- β= 4 if the Student ID is even, and 3 if the Student ID is odd
- $v_{max} = 46.0$
- $g = 9.8$
- $c = 14$
- $m = 60$
- step size $(h) = 2$
- lower interval limit = 0
- upper interval limit = 40
- $x_0 = 0$
- $y(x_0) = y_0 = 0$

Solve the equation above using the Taylor series method, Euler method, and the fourth-order Runge–Kutta method (RK4)**.**

Compare the graphs obtained from the three methods.

MODULE VI
ORDINARY DIFFERENTIAL EQUATION (ODE) 2nd ORDE

A. Theory :

Differential equations of order 2 are often found in simulation cases or simulations in the fields of engineering and science. For example, the equation

$$mx''(t) + cx'(t) + kx(t) = g(t)$$

Which represents a mechanical system with a spring having a spring constant *k* returns the shift position of a mass *m*. Damping is assumed to be proportional to velocity, as well as an external force *g*(t).

Ordinary differential equations (ODEs) of order 2 can be reduced to first-order ODEs. Consider an ODE order 2 in the form of

$$x''(t) = f(t, x(t), x'(t))$$

On the initial condition that $x(t_0) = x_0$ and $x'(t_0) = y_0$. The above equation can be reformulated as a system consisting of two first-order differential equations. When to use the equation

$$x'(t) = y(t)$$

then $x''(t) = y'(t)$ and the second-order differential equation becomes a system of equations

$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = f(t, x, y)$$

where $x(t_0) = x_0$ and $y(t_0) = y_0$.

Furthermore, methods such as Runge-Kutta can be used to solve the system of two differential equations of order one above. The method will result in two sequential $\{x_k\}$ and $\{y_k\}$. The first sequential is the solution of the second-order differential equation.

The main coding for the Runge Kutta method is order N = 4 for the system of differential equations of order one:

In the above coding using the feval command. This means that you have to create an m-file that contains the function to be analyzed. Then save the file (e.g. : F.m). When you evaluate at a value x, then: feval('F',x).

B. Objectives

After participating in this practice, participants are expected to:

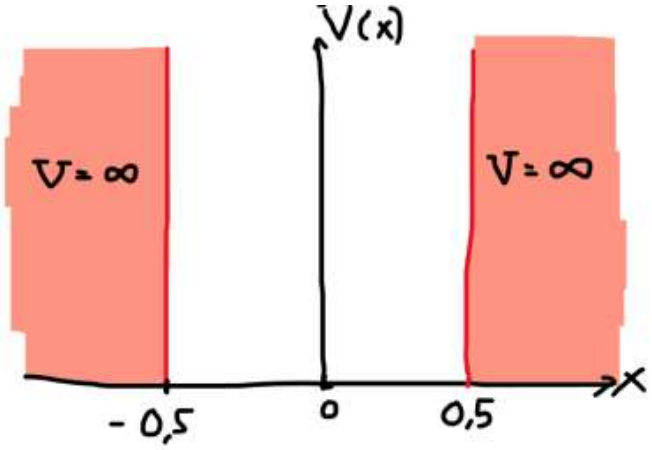
1. Understand and master the matlab programming language
2. Understand and master the 4th order Runge Kutta method to solve 2nd order differential equations numerically.

C. Practice Instructions

```
%% Runge Kutta Order 4
f=@(t,x,v) v;% v = dx/dt
g=@(t,x,v) -(2/10)*x -(3/10)*v;

a =0;
b = 50;
h = 0.1;
x= 10;
v = 0;
num = round((b-a)/h); tt_rk=zeros(num,1);
xx_rk=tt_rk; vv_rk=tt_rk; for i = 1:num
    t = a+i*h;
    p1 = h*f(t,x, v);
    q1 = h*g(t,x, v);
    p2 = h*f(t + 0.5*h, x + 0.5*p1, v + 0.5*q1);
    q2 = h*g(t + 0.5*h, x + 0.5*p1, v + 0.5*q1);
    p3 = h*f(t + 0.5*h, x + 0.5*p2, v + 0.5*q2);
    q3 = h*g(t + 0.5*h, x + 0.5*p2, v + 0.5*q2);
    p4 = h*f(t+h, x+p3, v+q3);
    q4 = h*g(t+h, x+p3, v+q3);
    x = x + (1/6)*(p1 + 2*p2 + 2*p3 + p4);
    fprintf('%.4f    %.4f \n',t,x)
    v = v + (1/6)*(q1 + 2*q2 + 2*q3 + q4);
    fprintf('%.4f    %.4f \n',t,v)
    vv_rk(i) = v; % Stores the value of y on each iteration
    xx_rk(i) = x; % Stores the value y on each iteration
    tt_rk(i) = t; % Stores x values on each iteration
end
%%
Figure(1)
Hold On
plot(tt, xx, 'r--');
plot(tt, vv, 'b--'); %
title('');
%% ============================== RK Order 4 ================================

f=@(z) z;
g=@(y,z) - 2*y - 3*z;

% Input
a = 0;   % Interval Lower Limit
b = 20;    % Upper Limit
Interval h = 0.01; % step
% x0=0; y0=0;
z0=0; y=0; z=10;
Num=round(abs((b-a)/h));
for i = 1:Num % Loop to perform numerical integration
    x = a+i*h;
    p1 = h*f(x,y,z);   % k1
    q1 = h*g(x,y,z);   % l1
    p2 = h*f(x+0.5*h,y+0.5*p1,z+0.5*q1);  % K2 Q2
    = H*g(X+0.5*H,Y+0.5*P1,Z+0.5*Q1);  % L2 P3 =
    H*F(X+0.5*H,Y+0.5*P2,Z+0.5*Q2);   % k3
```

```
    q3 = h*g(x+0.5*h,y+0.5*p2,z+0.5*q2);     % l3
    p4 = h*f(x+h,y+p3,z+q3);     % k4
    q4 = h*g(x+h,y+p3,z+q3);     % l4
    y = y + (1/6)*(p1 + 2*p2 + 2*p3 + p4);
    z = z + (1/6)*(q1 + 2*q2 + 2*q3 + q4);
    %fprintf('%.4f    %.4f \n',x,y)
    yy(i) = y; % Stores the value y on each iteration zz
    (i) = z;
    xx(i) = x; % Stores x values on each iteration
end


figure
hold on
plot(xx, yy, 'm--'); % Create a plot of the
numerical solution plot(xx,zz,'b--')
title('Differential Equation Solution Graph dy/dx = -xy; RK order 4'); % plot title
xlabel('x'); % Add x-axis labels
ylabel('y'); % Added y-axis label
xlim([0 20])

%% Euler
%f=@(t,x,v) v;
%g =@(t,x,v) -(2/10)*x -(3/10)*v;

% Input
a = 0;   % Interval Lower Limit
b = 50;    % Upper Limit
Interval h = 0.001; % step
x = 10;
v=0;
N=round(abs((b-a)/h)) ;   % total
iterations tt_e=zeros(N,1);
xx_e=zeros(N,1);
vv_e=zeros(N,1);

for i = 1:N % Loop to perform numerical integration
    t = a+i*h;
    x = x + h*f(t,x,v);
    v = v + h*g(t,x,v);
    fprintf('%.4f    %.4f %.4f\n' ,t, x, v)
    tt_e(i) = t; % Stores the value of y on each iteration
    xx_e(i)  =  x;  %  Stores  x  value  on  each  iteration
    vv_e(i) = v;
end
%%
Figure(1)
Hold On
plot(tt_e, xx_e, 'r--'); % Create a numerical solution plot
plot(tt_rk, xx_rk, 'b--');
title('Position chart of the Differential Equation of order 2'); % plot
title xlabel('t'); % Add x-axis labels
ylabel('x'); % Add y-axis labels

Figure(2)
Hold On
plot(tt_e, vv_e, 'r--'); % Create a plot of the numerical
solution plot(tt_rk, vv_rk, 'b--');
title('Velocity graph of order 2 differential equations'); % plot title
xlabel('t'); % Add x-axis labels
ylabel('v'); % Add y-axis labels
```

D. Assignment



There is an infinite potential well (infinite square well) as shown in the figure above. Determine the wave function $\psi(x)$and the probability of finding a particle $|\psi(x)|^2$inside the potential well.

The one-dimensional Schrödinger equation is given by:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x)$$

Assume that:

$$\frac{\hbar^2}{2m} = 1$$

Computational Boundaries
Lower bound of the interval = −0.5
Upper bound of the interval = 0.5

Step size h=

1. Conditions:

- Use the last 5 digits of the Student ID (NIM)
- Multiply the last 5 digits of the Student ID by 4.135×10^{-7}
  (example: $30044 \times 4.135 \times 10^{-7}$)

  Energy E=

2. Conditions:

- Use the last 2 digits of the Student ID
- If the last two digits are 01, 02, 03, ..., they must be reversed to become 10, 20, 30, ...
- Multiply the last two digits of the Student ID by 1.6
  (example: 44×1.6)

Initial conditions:

$$\psi(x_0) = \psi_0 = 0$$
$$\psi'(x_0) = \psi'_0 = 0.1$$