University of Biskra

Faculty of sciences and technology

Department of industrial chemistry

2nd year process engineering

Dr Khaled ATHMANI

khaled.athmani@univ-biskra.dz

PW numerical methods

2024/2025

Contents

Contents

1 Introduction

This course, designed for second-year process engineering students, focuses on applying numerical methods to solve engineering problems using MATLAB. The series of practical works (PWs) introduces students to key numerical techniques and their implementation in MATLAB, building essential computational skills for tackling complex engineering challenges.

1. **Course Structure:**

- **PW01 - Numerical Methods for Solving Nonlinear Equations**: This session covers the bisection method and Newton-Raphson method, providing students with techniques to find roots of nonlinear equations.
- **PW02 - Interpolation**: Students will learn how to apply Newton and Lagrange interpolation methods, allowing them to estimate unknown values within a range of known data points.
- **PW03 - Numerical Integration Methods**: This session focuses on different integration techniques such as the midpoint, Simpson's, and trapezoidal rules, helping students approximate the integral of a function.
- **PW04 - Resolution of Differential Equations**: Students will be introduced to the Euler and Runge-Kutta methods for solving ordinary differential equations, key tools in modeling dynamic engineering systems.
- **PW05 - Numerical Solution of Linear Equation Systems**: This session explores the Jacobi method, an iterative approach for solving systems of linear equations, essential for various engineering applications.

The aim of this course is to provide students with the knowledge and practical skills necessary to solve mathematical problems encountered in process engineering using numerical methods. By the end of the course, students will be able to:

2. Implement numerical methods for solving nonlinear equations and interpolate data.
3. Apply various numerical integration techniques to approximate solutions.
4. Solve ordinary differential equations using Euler and Runge-Kutta methods.
5. Use iterative techniques to solve systems of linear equations.

This course will enable students to efficiently use MATLAB to apply numerical methods to real-world engineering problems, enhancing their problem-solving capabilities in their academic and professional pursuits.

2nd year PE **University of Biskra**

PW numerical methods **2024/2025**

2 PW 01- Numerical methods for solving nonlinear equations

2.1 Objectives:

Apply the methods to solve various types of nonlinear equations.

Develop MATLAB code for the bisection and Newton-Raphson methods.

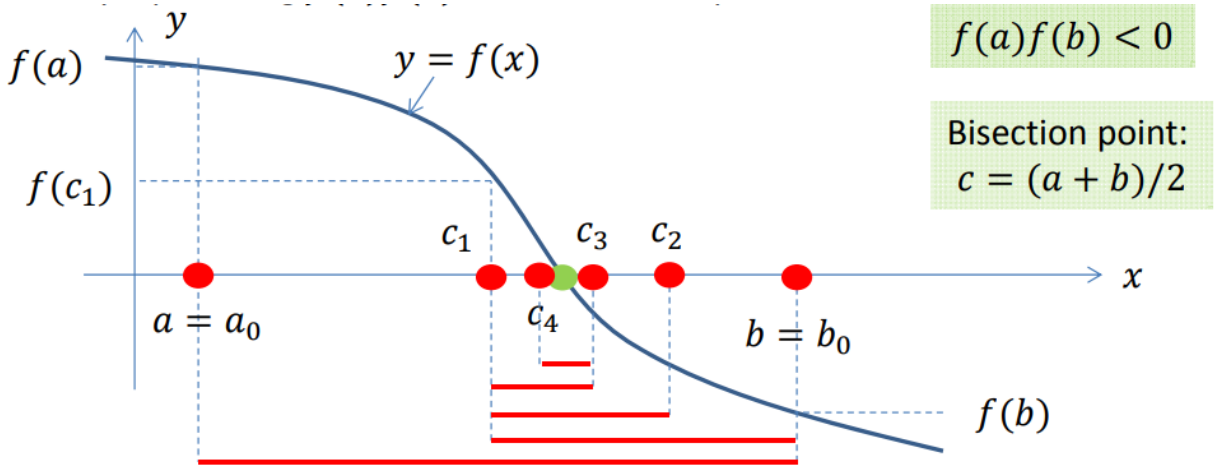
2.2 Bisection method

2.2.1 Bisection method (key idea)

Let's assume that we localize a single root in an interval (a, b) and $f(x)$ changes sign in the root. If the interval (a, b) contains one root of the equation, then

$$f(a)f(b) < 0.$$

Let's iteratively shorten the interval by bisections until the root will be localized in the sufficiently short interval. For every bisection at the central point $c = (a + b)/2$, we replace either a or b by c providing $f(a)f(b) < 0$ after the replacement.



2.2.2 Bisection algorithm:

1. Define the function **f(x)**

2. Take the interval **[a b]** where **f(a)*f(b) < 0**

3. Choose pre-specified tolerable error **E**.

4. Find middle point **c = (a + b)/2**

5. Calculate **f(a)f(b)**

    a. if **f(a)*f(c) < 0** then **b=c**

    b. **if f(a)*f(c) > 0** then **a= c**

    c. if **f(a)f(c) = 0** then go to (7)

6. If **abs(b-a)> E** then go to (4) otherwise go to (7)

7. Display **c** as root.

2.2.3 Exercise:

Write a MATLAB script for solving the nonlinear equation $f(x) = 0$ using the bisection method on the interval [1, 2] with an accuracy of $E = 10^{-3}$.

$$f(x) = x^3 + x^2 - 3\,x - 3$$

Display the estimated roots and the number of iterations used to achieve the solution.

2.2.4 Solution

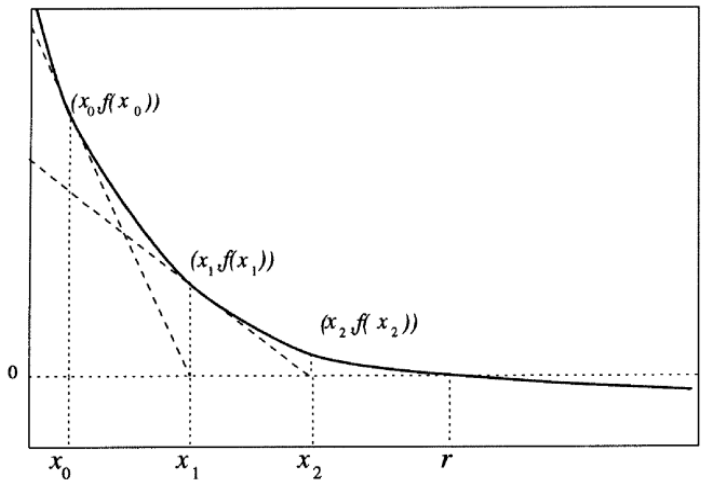
```
%************************************************************
% Bisection method                                Dr Khaled Athmani
% A code for solving a nonlinear equation using bisection method
% PW01 2^ed year process engineering university of Biskra 2024
% f(x)= x^3 + x^2 -3 *x -3
%************************************************************
%************************************************************
clear All     % To clear all variables from the current workspace
close All     % To close all open files
clc           % Clears all the text from the Command Window

f=@(x) x^3 + x^2 -3*x -3;    % function f(x)

a=1;       % the left bound in the interval[a,  b]
b=2;       % the right bound in the interval[a,  b]
E= 1e-3;   % Error   1*10^(-3)
k=0;

disp('====================== RESULTS ======================')

while abs(b-a) > E
   xm=(a+b)/2;     %xm is the midel point of the interval[a,  b]

   if f(xm)== 0.0     % solved the equation exactly
       disp('xm is the exact solution')
       break         % jumps out of the for loop
   elseif f(a)*f(xm)<0
       b=xm;

   else     %f(a)*f(xm)>0
       a=xm ;

   end

    k=k+1;     % k is the number of iterations
    fprintf(' a= %8.6f b= %8.6f \n',a,b);

end
disp('-----------------------------------------------------')
fprintf(' x= %8.6f  \n',xm);
fprintf(' Error= %8.6f  \n',abs(b-a) );
fprintf(' Number of iterations %d  \n',k);
disp('-----------------------------------------------------')
```

2.3 Newton-Raphson method

The Newton method is applied to solve an equation of the form $f(x) = 0$. Given an approximation x_0 of the root, we construct the tangent line to the curve with equation $y = f(x)$ at the point with abscissa x_0. This line intersects the horizontal axis at x_1. We then construct a new tangent at this abscissa, and its intersection with the x-axis gives us x_2. This process is iterated until convergence.

Consider a differentiable function f:[a,b]→R and a point $x_0 \in [a, b]$ We can define a recursive sequence as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



2.3.1 Algorithm for Newton Raphson Method

1. Define the function f(x)
2. Define first derivative of f(x) as df(x)
3. Input initial guess (x_0), tolerable error E
4. Calculate $x_1 = x_0 - \frac{f(x_0)}{df(x_0)}$
5. if $\boldsymbol{f(x_1) = 0}$ then go to (7)
6. If abs(x1-x0) > E then set $x_0 = x_1$ and go to (4) otherwise go to (7)
7. Print root as x1

2.3.2 Exercise:

-Write a MATLAB script for solving the nonlinear equation $f(x) = 0$ using the Newton Raphson method on the interval [1, 2] with an accuracy of $E = 10^{-3}$.

$$f(x) = x^3 + x^2 - 3\,x - 3$$

-Display the estimated roots and the number of iterations used to achieve the solution.

-Compare the number of iterations obtained with the Newton-Raphson method to those obtained by the bisection method in the previous exercise. Determine which method required fewer iterations.

2.4 Solution

```
%%
%*****************************************************************
%  The Newton method                  Dr Khaled Athmani
%  A code for solving a nonlinear equation using the Newton
method
% PW 01  2 year process engineering university of Biskra
2022/2023
%  f(x)= x^3 + x^2 -3 *x -3
%*****************************************************************
%*****************************************************************

clear All     %  To clear all variables from the current
workspace
close All     %  To close all open files
clc           %  clears all the text from the Command Window

f=@(x) x^3 + x^2 -3 *x -3;% x^3 + x^2 -3 *x -3;     % function
f(x)
df=@(x) 3*x^2 + 2*x -3; % df(x)/dx
p=1e-3;    % Error
x0=1;

Er=abs(x0-f(x0)/df(x0)-x0);  % abslute error

k=0;
disp('==================== RESULTS ======================')
while Er > p

     x1= x0-f(x0)/df(x0);      % x_(i+1)=g(_i)
```

```
      Er=abs(x1-x0); % abslute error
      x0=x1;
      k=k+1;          % number of iterations
      fprintf(' x= %8.6f  \n',x1);  %to print x_i for each step

       if f(x1)== 0.0    % solved the equation exactly
          disp('x: is the exact solution')
          break          % jumps out of the for loop
       end
end
disp('-----------------------------------------------------')
fprintf(' The solution is : %8.6f \n',x1);
fprintf(' Error= %8.6f  \n',Er);
fprintf(' Number of iterations %d  \n',k);
disp('-----------------------------------------------------')
```

2.5 Références

- Amireche, M. *TP1 - Méthodes Numériques*. Université de Larbi Ben M'hidi, 2019-2020.
- Amireche, M. *TP2 - Méthodes Numériques*. Université Badji Mokhtar Annaba, 2018-2019.
- Fortin, André. *Analyse Numérique pour Ingénieurs*. L'École Polytechnique de Montréal.
- Volkov, Alexey. *Numerical Analysis ME 349, Engineering Analysis*.

2nd year PE **University of Biskra**

PW numerical methods **2024/2025**

3 PW 02- Interpolation

Newton interpolation is a method used to approximate a function or a set of data points by constructing an interpolating polynomial. This polynomial passes through the given data points exactly. The Newton interpolation method is particularly useful when the data points are unevenly spaced.

3.1 Objectives:

1. Utilize Newton interpolation with divided differences to approximate a smooth curve passing through a given set of data points (x_n, y_n).
2. Develop MATLAB code to compute the coefficients of the interpolating polynomial and generate the interpolated curve for visualization and analysis.

3.2 Newton's Polynomial Interpolation

The basic idea behind Newton interpolation is to construct a polynomial of degree n that passes through $n+1$ given data points. This polynomial is expressed in terms of divided differences, which are a way of organizing and calculating the coefficients of the polynomial.

Here's a simplified outline of the Newton interpolation process:

1. Given a set of $n+1$ data points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, where the x_i are distinct.
2. Define the divided differences recursively. The zeroth divided difference is the function value at the given points:

$$f[x_i] = y_i.$$

The first divided differences are defined as:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i},$$

and in general, the n^{th} divided differences are:

$$f[x_i, x_{i+1}, \dots, x_n] = \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0}.$$

3. Construct the interpolating polynomial using these divided differences. The Newton interpolation polynomial is given by:

$$P_n(x) = \boldsymbol{a_0} + \boldsymbol{a_1}(x - x_0) + \boldsymbol{a_2}(x - x_0)(x - x_1) + \cdots + \boldsymbol{a_n}(x - x_0)(x - x_1) \dots (x - x_{\mathrm{n}-2})(x - x_{\mathrm{n}-1})$$

Where:

$$\boldsymbol{a_0} = f[x_0], \boldsymbol{a_1} = f[x_0, x_1], \boldsymbol{a_2} = f[x_0, x_1, x_2], \dots., \boldsymbol{a_n} = f[x_0, x_1, x_2, \dots, x_{\mathrm{n}}]$$

3.2.1 Algorithm: computing the divided differences

1) Given the points $(x_i, y_i), i = 0, \dots, n$.
2) Initialize $f_{i,0} = y_i, i = 0, \dots, n$
3) Calculate $f_{i,\mathrm{j}}, i = 1, \dots, n$

for $i = 1{:}\,n$
  for $j = 1{:}\,i$
    $f_{i,j} = \dfrac{f_{i,j-1} - f_{i-1,j-1}}{x_i - x_{i-j}}$
  end
end

4) Result: The diagonal, $f_{i,i}$ now contains $f[x_0, \dots, x_i]$.

.

3.2.2 Exercise 01:

1. Given the following table of data points (x_i, y_i) , plot the points on a graph using MATLAB:

x_i	0	1	2	3
y_i	1	2	9	28

2. Write a MATLAB script to calculate the divided differences for the given data points and determine the interpolating polynomial of order 3 using Newton's divided differences method.
3. Plot the interpolating polynomial along with the original data points on the same graph.
4. What conclusion can be drawn from observing the plotted graph regarding the relationship between the interpolated polynomial and the given data points

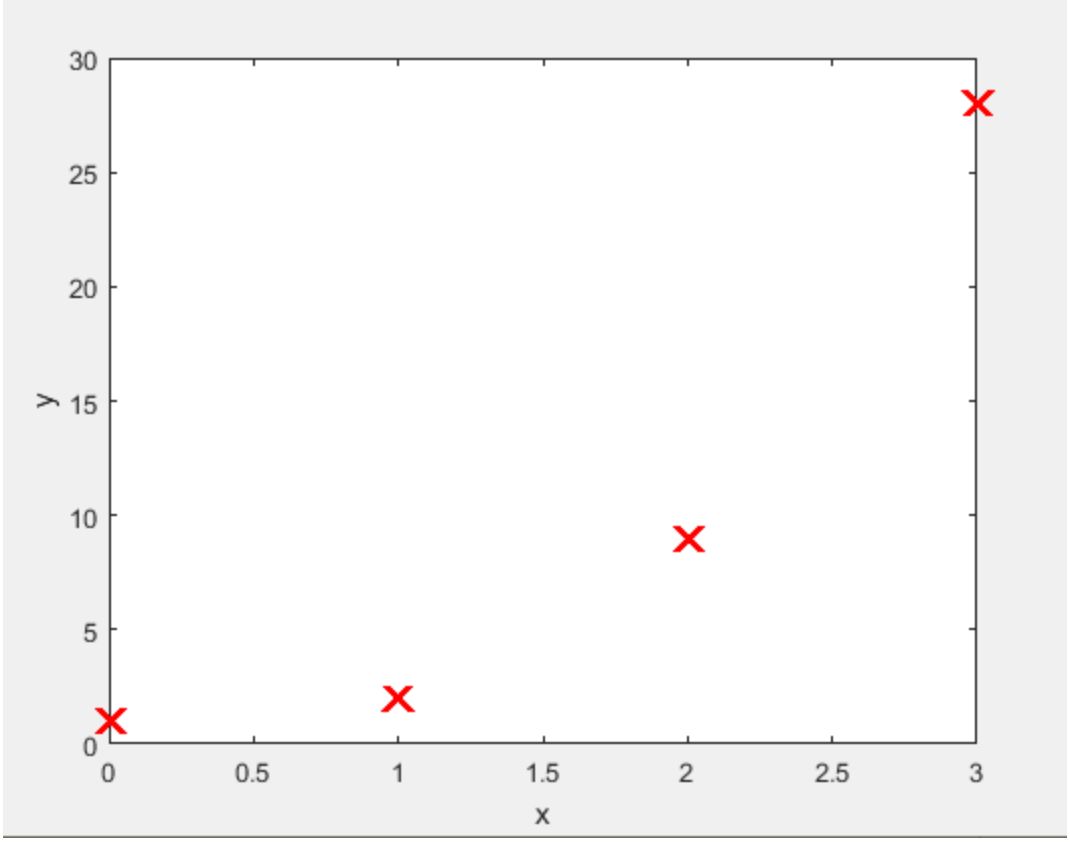
3.2.3 Solution

1. Plot the points on a graph using MATLAB:

Create a new script and write the following:

```
clear; clc
x=[0 1 2 3];
y=[1 2 9 28];
plot(x,y,'rx','MarkerSize',12, 'LineWidth',3 )
xlabel('x')
ylabel('y')
```

After running the code, you will obtain this graph:



2. Write a MATLAB script to calculate the divided differences:

```
f=y;
a(1)=f(1);
for j=1 :3
   for k=1:(4-j)
      f(k)=(f(k+1)-f(k))./( x(k+j)- x(k));
      a(j+1)=f(1) ;
   end
end
fprintf('The divided difference is: %2.8f  \n',a);
```

After running the code, you will obtain this result:

The divided difference is: 1.00000000

The divided difference is: 1.00000000

The divided difference is: 3.00000000

The divided difference is: 1.00000000

Determine the interpolating polynomial of order 3:

$$P_3(x) = \boldsymbol{a_0} + \boldsymbol{a_1}(x - x_0) + \boldsymbol{a_2}(x - x_0)(x - x_1) + \boldsymbol{a_3}(x - x_0)(x - x_1)(x - x_2)$$

$$P_3(x) = \mathbf{1} + (x - 0) + \mathbf{3}(x - 0)(x - 1) + (x - 0)(x - 1)(x - 2)$$

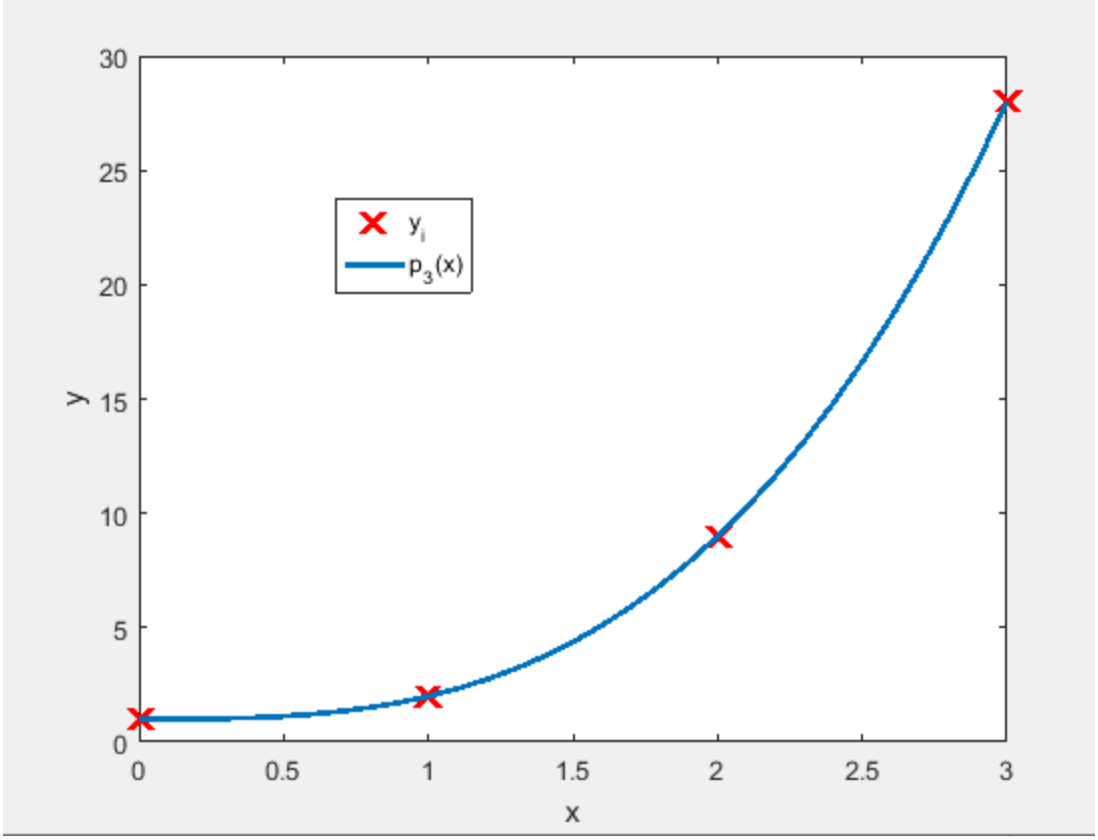
$$P_3(x) = 1 + x^3$$

3. Plot the interpolating polynomial along with the original data points on the same graph.

Write this part:

```
z=x(1):0.1 : x(end);
px=1 + z.^3;
hold on
plot(z, px, 'LineWidth',2)
legend('y_i','p_3(x)')
```

After running the code, you will obtain this graph:


30
25
20
15
10
5
0
y
0
0.5
1
1.5
2
2.5
3
x
y_i
p_3(x)

3.2.4 Exercise 02:

1. Given the following table of data points (x_i, y_i) , plot the points on a graph using MATLAB:

x_i	0	1	2	3	5
y_i	1	2	9	28	54

2. Write a MATLAB script to calculate the divided differences for the given data points and determine the interpolating polynomial of order 4 using Newton's divided differences method.
3. Plot the interpolating polynomial along with the original data points on the same graph.
4. What conclusion can be drawn from observing the plotted graph regarding the relationship between the interpolated polynomial and the given data points.

3.3 Références

Fortin, André. Analyse Numérique pour Ingénieurs. L'École Polytechnique de Montréal.

4 PW 03- Numerical integration methods

The objective of this practical work is the numerical calculation of a definite integral using the midpoint method (rectangles), trapezoids and Simpson's methods.

4.1 Rectangle method

The rectangle method consists of:

- Dividing the interval $[a, b]$ into n equal segments. Thus, we obtain $(n + 1)$ equidistant points: Let's define:

$$x_j = a + jh, \qquad j = 0,1, \dots, n \qquad with \;\; h = \frac{b-a}{n}$$

- Approximate the area of each "slice" by a rectangle (Figure 01).
- Considering the midpoint of each subinterval to increase precision, the expression of the integral becomes in this case:

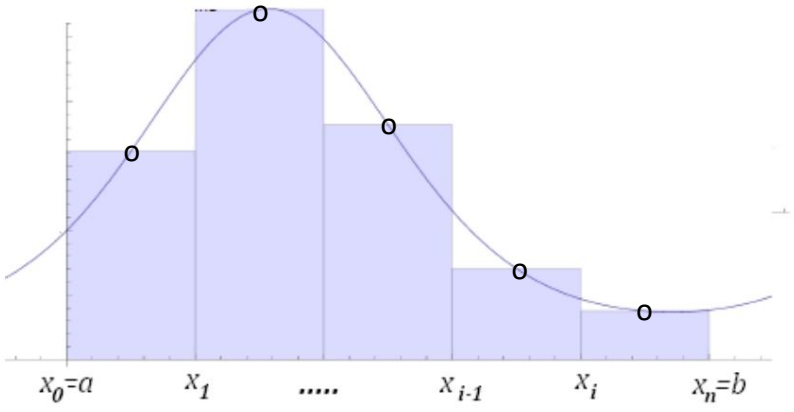


Figure 01: Rectangle method

$$I(f) = \int_a^b f(x)dx \approx h \sum_{j=0}^{n-1} f\left(x_j + \frac{h}{2}\right)$$

4.2 Trapezoidal Method

The trapezoidal method consists of:

- Divide the interval $[a, b]$ into n equal segments. This results in $(n + 1)$ equidistant points. Let: $x_j = a + jh, j = 0,1, \dots, n$ with $h = \frac{b-a}{n}$.
- Approximate the area of each "slice" by a trapezoid constructed from the function values at the boundaries of each subinterval (Figure 02).
- The general formula for the integral using the trapezoidal method is written as

$$I(f) = \int_a^b f(x)dx \approx \frac{h}{2} \sum_{i=0}^{n-1} \left(f(x_j) + f(x_{j+1})\right)$$

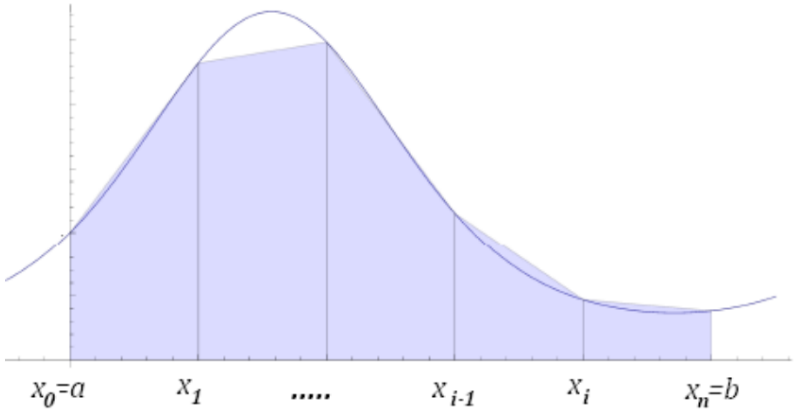


Figure 02: Trapezoidal Method

4.3 Simpson's method:

The Simpson's method consists of

- Divide the interval $[a, b]$ into n equal segments, where n is an even number $(n = 2m)$. This results in $(2m + 1)$ equidistant points: $x_j = a + jh, j = 0,1, \dots, n$ with $h = \frac{b-a}{n}$.
- Approximate the function on each "slice" by a parabola constructed from three consecutive points (Figure 03).

- The general formula for the integral using Simpson's method is written as

$$I(f) = \int_a^b f(x)dx \approx \frac{h}{3}\left(f(a) + f(b) + 4\sum_{j=1}^{m} f(x_{2j-1}) + 2\sum_{j=1}^{m-1} f(x_{2j})\right)$$

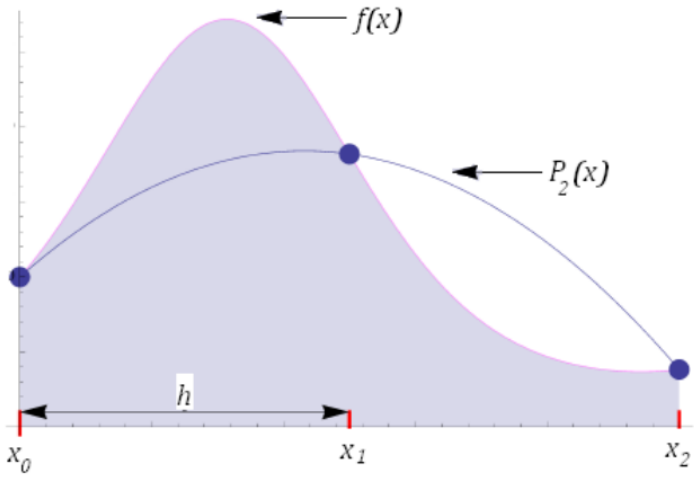


Figure 03: Simpson's method

4.4 Application:

We propose to calculate the definite integral

$$I(f) = \int_0^2 ln(2x+1) + 2\, e^{-2x} dx$$

01) Write a program that calculates this integral using the midpoint, trapezoidal, and Simpson's methods with $n = 10$.

02) Calculate the absolute error of each method and draw conclusions based on the comparison of the errors.

03) Repeat the execution with $n = 50$, then draw conclusions.

Given: The exact value of the integral $I(f)$ is 3.00527914219611.

04) Apply the same steps for the integral: $I(f) = \int_0^2 2\, e^{-2x} dx$.

4.5 Solution

```
%============================================================
%============================================================
% University of Biskra Algeria, 2 year chemical process
% Created by Dr Khaled Athmani     02/04/2023
% A code for solving an intgral numerically using:
% 01) midpoint        02) trapezoidal    03) Simpson
% The integral is: int_0^2 f(x) dx, f(x)=log(2*x+1)+2*exp(-2*x)
%============================================================
%============================================================

clear
clc
% local variables and functions
I_ana=3.00527914219611; % The exaction solution
f=@(x) log(2*x+1)+2*exp(-2*x); % the function f(x)
a=0; % is the lower bond
b=2; % is the upper bond
n=10;% number of sections
h=(b-a)/n;


%********************** Midpoint method***********************
I_num=0;
  for j=0:n-1
    x=a+j*h; % x_j
    I_num=I_num+h*f(x+h/2);% s=h*sum f(x_j+h/2)
  end
 disp('=============  Midpoint method ==============')
 error=abs(I_num-I_ana);
 fprintf('Numerical solution: %2.12f \n',I_num)
 fprintf('Abselute error: %2.12e \n',error)
 disp('=============================================')


 % ******************Trapezoidal method ********************
 I_num=0;
  for j=0:n-1
    x=a+j*h; %  x_j
    I_num=I_num+ h/2*(f(x)+f(x+h)); % s=h*sum f(x_j+h/2)
  end
```

```
 disp('============= Trapezoidal method ==============')
 error=abs(I_num-I_ana);
 fprintf('Numerical solution: %2.12f \n',I_num)
 fprintf('Abselute error: %2.12e \n',error)
 disp('================================================')


 % ********************** Simpson method **********************
 I_num=0;

for j=1:n-1
   x=a+j*h;

   if mod(j,2)==0
       I_num=I_num+2*f(x);
   else
       I_num=I_num+4*f(x);
   end

end

I_num=h/3*(f(a)+f(b)+I_num);

 disp('============= Simpson method ==================')
 error=abs(I_num-I_ana);
 fprintf('Numerical solution: %2.12f \n',I_num)
 fprintf('Abselute error: %2.12e \n',error)
 disp('================================================')

 %====================}}}}}}} END {{{{{{{====================
```

4.6 References

- Meddahi, Youssuof. *Travaux Pratiques Méthodes Numériques*. Université Hassiba Benbouali de Chlef, 2018-2019.
- Université de Tlemcen. *Chapitre 4 - Méthodes Numériques*. Available at: https://elearn.univ-tlemcen.dz/pluginfile.php/126336/mod_resource/content/1/Chapitre%204.pdf (Accessed: 17 March 2025).

2nd year PE **University of Biskra**

PW numerical methods **2024/2025**

5 PW4- Resolution of differential equations

This practical work involves solving ordinary differential equations using algorithms of basic methods: Euler's method and the fourth-order Runge-Kutta method. We will compare these two methods with the exact solution in this practical work.

5.1 Review on numerical solution methods

We consider a first-order differential equation in resolved form:

$$y' = \frac{dy}{dx} = f(x, y)$$

With the initial condition:

$$y(x_0) = y_0$$

5.2 Euler's Method

The principle is to approximate the solution y over $[a, b]$ by a piecewise linear function. By discretizing the parameter, we define: $x_i = a + ih$ where $h = (b - a)/n$ is the step size. The piecewise linear function will connect the points with coordinates (x_i, y_i). The objective is to propose an algorithm to construct the y_i from y_0 :

$$\begin{cases} y_0 = y(x_0) \\ y_{i+1} = y_i + hf(x_i, y_i) \end{cases}$$

5.3 Runge-Kutta methods

The Runge-Kutta methods are numerical analysis methods for approximating solutions to differential equations. They were named in honor of the mathematicians Carl Runge and Martin Wilhelm Kutta, who developed the method in 1901.

For the fourth-order Runge-Kutta method, the following iteration is used:

$$y_0 = y(x_0)$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = h * f(x_i, y_i)$$
$$k_2 = h * f\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right)$$
$$k_3 = h * f\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right)$$
$$k_4 = h * f(x_i + h, y_i + k_3)$$

5.4 Applications:

Consider the differential equation:

$$\begin{cases} \frac{dy}{dx} = y + \exp(-x), \quad x \in [0,1] \\ y(0) = 5 \end{cases}$$

01) Write a Matlab program to calculate 10 iterations of Euler's method, then of the 4th-order Range Kutta method, each time comparing the approximate solutions with the exact solution:

$$y(x) = \frac{1}{2}e^{-x}(-1 + 11\, e^{2x})$$

02) Complete the previous program to plot and compare the curves: exact solution, approximate solution using Euler's method, and approximate solution using fourth-order Runge-Kutta method. What is your conclusion.

03) Taking $n = 50$ and redoing questions 1 and 2.

5.4.1 Euler's method Matlab code:

```
%=======================================================
%=======================================================
% University of Biskra Algeria,  2 year chemical process
% Created by Dr Khaled Athmani     02/04/2023
% A code for solving an ODE  numerically using:
% 01) Euler Method   02) Runge-Kutta method
% ODE: dy/dtx= f(y,x), y + exp(-x)
%=======================================================
%=======================================================

clear all
clc
% local variables and functions
f=@(x,y) y + exp(-x); % dy/dx= f(y,x)
n=10; % number of iterations
a=0; % a   x=[a, b]
b=1; % b
h=(b-a)/n;
%x=a:h:b; % vector x =[x0, x1, x2, ..... xn]

% Euler method ------------------------------------------------
----------
y(1)=5; % initial condition
yana(1)=5;
E(1)=0;
x(1)=a;
for j=1:n
    x(j+1)=a+j*h;
    y(j+1)=y(j)+ h*f(x(j),y(j)) ; % y(j+1)
    yana(j+1)=1/2 .*exp(-x(j+1)).* (-1 + 11.* exp(2.*x(j+1))); %
Exact solution
    E(j+1)=abs(yana(j+1)-y(j+1)); % Absolute error
end


%Plot the results with the exact solution----------------------
---------
```

```
%plot(x,y,'r*',x, yana  ,'k-','MarkerSize',9,'LineWidth',1.2)
%legend('Euler','Exact solution')
%xlabel('x')
%ylabel('y(x)')



% Rung Kutta 04 method----------------------------------------
----------
yrk(1)=5; % initial condition
for j=1:n
   k1=h*f(x(j),yrk(j)); % k1

   k2=h*f(x(j)+h/2,yrk(j)+k1/2);  % k2

   k3=h*f(x(j)+h/2,yrk(j)+k2/2); % k3

   k4=h*f(x(j)+h,yrk(j)+k3); % k4

   yrk(j+1)=yrk(j)+1/6*(k1 +2*k2 +2*k3 +k4); % y(j+1)
end


%  Plot the results with the exact solution--------------------
-----------
%
plot(x,y,'r*',x,yrk,'b*',x, yana  ,'k-
','MarkerSize',9,'LineWidth',1.2)
legend('Euler','Rung Kutta 4','Exact soltion')
xlabel('x')
ylabel('y(x)')

%**********************  {{{{ END }}}}**********************
```

5.5 References

- Meddahi, Youssuof. Travaux Pratiques Méthodes Numériques. Université Hassiba Benbouali de Chlef, 2018-2019.

2nd year PE **University of Biskra**

PW numerical methods **2024/2025**

6 PW5- Numerical solution of linear equation systems

In this practical work, we'll focus on solving systems of linear algebraic equations using the Jacobi method, with the aid of a MATLAB program. Linear algebra plays a crucial role across various disciplines, making efficient solution methods essential. Through this practical work, we aim to provide both theoretical insights into the Jacobi method and hands-on experience in implementing it computationally using MATLAB.

6.1 The Jacobi Method

The Jacobi method is an iterative technique used to solve a system of linear equations. It's based on splitting the coefficient matrix of the system into a diagonal component and the remaining off-diagonal components. Then, it iteratively updates the solution vector until convergence is achieved.

The Jacobi method sometimes converges even if these conditions are not satisfied:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

For each $k \geq 1$, generate the components $x_i^{(k)}$ of $\boldsymbol{x}^{(\boldsymbol{k})}$ from $\boldsymbol{x}^{(\boldsymbol{k-1})}$ by

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\sum_{j=1, j \neq i}^{n} \left(-a_{ij} x_j^{(k-1)} \right) + b_i \right], \quad \text{for } i = 1,2, \dots n$$

In the Jacobi method, convergence is typically measured by the difference between successive approximations of the solution vector. This difference is indeed given by the Euclidean norm of $\boldsymbol{x}^{(\boldsymbol{k+1})} - \boldsymbol{x}^{(\boldsymbol{k})}$:

$$E = \left\| \boldsymbol{x}^{(\boldsymbol{k-1})} - \boldsymbol{x}^{(\boldsymbol{k})} \right\|$$

6.1.1 Algorithm

input: initial guess $\boldsymbol{x}^{(\mathbf{0})}$ to the solution, (diagonal dominant) matrix $\boldsymbol{A}$, right-hand side vector $\boldsymbol{b}$, convergence criterion

Output: solution when convergence is reached

$k = 0$

while convergence not reached do

for $i := 1$ step until n do

$S = 0$

for $j := 1$ step until n do

if $j \neq i$ then

$S = S + a_{ij} x_j^{(k)}$

end

end

$x_i^{(k+1)} = \frac{1}{a_{ii}} (b_i - S)$

end

$k = k + 1$

end

6.1.2 Applications:

Consider the following system of linear equations:

$$\begin{cases} 5\,x_1 - 2x_2 + 3\,x_3 = -1 \\ -3\,x_1 + 9\,x_2 + x_3 = 2 \\ 2\,x_1 - x_2 - 7x_3 = 3 \end{cases}$$

04) Write a MATLAB script to solve this system of linear equations using the Jacobi method. Use an initial guess $x^0 = [0\ 0\ 0]$ and iterate until the solution reaches a specified accuracy level, 10^{-5}:

05) Print the solutions at each iteration and determine the total number of iterations required.

For the first 3 iterations you must find:

---

at the iteration 1

-0.2000000

0.2222222

-0.4285714

---

at the iteration 2

0.1460317

0.2031746

-0.5174603

---

at the iteration 3

0.1917460

0.3283951

-0.4158730

At the final iteration you must find:

0.1861192

0.3312319

-0.4227112

6.1.3 Solution

```
%================================================================
% A code to solve a linear equation system (Ax=b) by using
Jacobi method
%
% Created by Khaled Athmani   on 01/04/2024
%================================================================

clear
clc
A=[5 -2 3; % is the matrix A
   -3 9 1;
   2 -1 -7];
b=[-1 2 3]'; % is the column vector b
x=[0 0 0]'; % is the column vector x (unknown)
xf=[0 0 0]';
E=1.e-5;
%n=10; % is the number of iterations

%x=A^(-1)*b

%for k=1: n
En=10*E;
k=0;
while(En > E)
  for i=1:length(x)
    s=0;
    for j=1:length(x)
        if i ~=j
          s=s-A(i,j).*x(j);
        end

    end
    xf(i)=1./A(i,i).*(s+b(i));

    %fprintf('%2.5f \n',xf(i))
  end
  disp('-----------------------------------')
    fprintf('at the iteration %d \n',k+1)
    fprintf('%2.7f \n',xf)
  En=norm(x-xf);
```

```
  x=xf;
  k=k+1;
end
%fprintf('at the iteration %d \n',k)
```

6.2 References

- Meddahi, Youssuof. *Travaux Pratiques Méthodes Numériques*. Université Hassiba Benbouali de Chlef, 2018-2019.
- Wikipedia. *Jacobi Method*. Available at: https://en.wikipedia.org/wiki/Jacobi_method (Accessed: 17 March 2025).

7 Exams

Duration: 45 min

Name: ……

Course: Numerical methods

Group: ……

Test A (6 points)

Exercise 01 (4 pts):

Write a MATLAB script for solving the nonlinear equation $f(x) = 0$ using the bisection method on the interval [-2, 0] with an accuracy of $E = 2\ 10^{-3}$.

$$f(x) = -6x - x\text{^}2 - 5$$

Display the estimated roots and the number of iterations used to achieve the solution.

Plot the function $f(x)$ for $x \in [-2, 2]$

Solution:

……

……

……

……

……

……

……

……

……

……

……

……
……
……
……
……
……
……

Exercise 02 (2 pts):

We propose to calculate the definite integral

$$I(f) = \int_0^2 2\, x\, e^{-2\,x} dx$$

05) Write a MATLAB program that calculates this integral using the midpoint with $n = 12$.

Solution:

……
……
……
……
……
……
……
……
……
……
……

……
……
……
……
……
……
……
……
.

Duration: 45 min Name: ……..……

Course: Numerical methods Group: ……

Test B (6 points)

Exercise 01 (4pts):

Consider the differential equation:

$$\begin{cases} \dfrac{dy}{dt} = 2\,y + \exp(-2t)\,, \quad t \in [0,1] \\ y(0) = 5 \end{cases}$$

06) Write a MATLAB program to solve this problem using Euler's method, where $t_{j+1} - t_j = 0.1$each time comparing the approximate solutions with the exact solution:

$$y(t) = \frac{1}{4}e^{-2t}(-1 + 5e^{4t})$$

07) Complete the previous program to plot and compare the curves: exact solution, approximate solution.

Solution:

……
……

Exercise 02 (2pts):

We propose to calculate the definite integral

$$I(f) = \int_0^2 2\, x\, e^{-2\,x} dx$$

06) Write a program that calculates this integral using the trapezoidal with $n = 15$.

……

……

……

……

……

……

……

……

……

……

……

……..