ENUME Project: assignment B #12

Samuele Ferraro

Warsaw University of Technology, Faculty of Electronics and Information Technology

11 May 2023

I declare that this piece of work, which is the basis for recognition of achieving learning out-comes in the Numerical Methods course, was completed on my own.

Contents

1 Mathematical symbols used

- x = general purpose independent variable.
- y = general purpose dependent variable.
- $f(\cdot)$ = general purpose function as $y = f(x)$
- Δ = Absolute error.
- $||\cdot||$ = Euclidian norm operator.
- $\dot{x}$ = exact value.
- $f'(\cdot)$ = first derivative of function f.
- $f''(\cdot)$ = second derivative of function f,
- $|\cdot|$ = absolute value operator.
- $min(\cdot)$ = minimum value operator.
- 1,E-a = different notation for 10^{-b}
- $\hat{x}$ = extimated value of x.

2 Introduction: Nonlinear algebraic equations

2.1 Definition

A nonlinear algebraic equation is mathematical equation that involves functions where the output does not changes proportionally with the input.

$$f(x) = 0 \tag{1}$$

Where $f(x)$ is a nonlinear function

Examples of these functions are:

- Polynomial equation with degree greater than 1
- trigonometric function as sin, cos, etc.
- exponential and logarithmic functions

In other words, the purpose is finding the roots (or zeros) of $f(x)$.

2.2 Solving nonlinear algebraic equations: iterative methods

An iterative algorithm used to get data d to result w has the form of:

$$w_{i+1} = \phi_i(w_i, w_{i-1}, ..., d) \tag{2}$$

where

- $\phi_i(\cdot)$ is the operator that defines the iterative algorithm
- w_i is the ith approximation of the solution of the equation.
- i = number of iterations

In order to be sure to find a solution, some conditions have to be respected:

- The solution must exist in a certain range where the function is continuous
- The calculation process must lead to a convergent solution

The convergence of an iterative algorithm can be:

- Global if there exist a non-empty set $\mathbb{W}_0$ of starting points w_0 where:

$$\|\Delta w_i\| = \|w_i - \dot{w}\| \text{ for } i = 0, 1, 2, 3, .. \tag{3}$$

- Local if it is convergent for a any $\mathbb{W}_0$ of the point $\dot{w}$. It is analysed by the error function:

$$\Delta w_{i+1} = \Phi(\Delta w_i; d) \text{ for } i = 0, 1, 2, 3, ... \tag{4}$$

  Usually it is evaluated using two parameter: ρ and C. An iterative algorithm is locally convergent if $C < 1$ for $\rho = 1$ and $C < \infty$ for $\rho > 1$.

  The higher the value is ρ, the faster is the convergence of the algorithm.

2.2.1 Bisection method

Considering $f(x)$ continuous in $[a_0, b_0]$ and $f(a_0) \cdot f(b_0) < 0$, the bisection method is defined by

$$x_{i+1} = \frac{1}{2}(a_{i+1} + b_{i+1}) \text{ with } \left\{ \begin{matrix} a_{i+1} = a_i; b_{i+1} = x_i \text{ if } f(a_i) \cdot f(x_i) < 0 \\ a_{i+1} = x_i; b_{i+1} = b_i \text{ if } f(a_i) \cdot f(x_i) > 0 \end{matrix} \right\} \text{ for } i = 0, 1, 2, 3... \quad (5)$$

A graphical representation of the bisection method is presented in Figure 1:

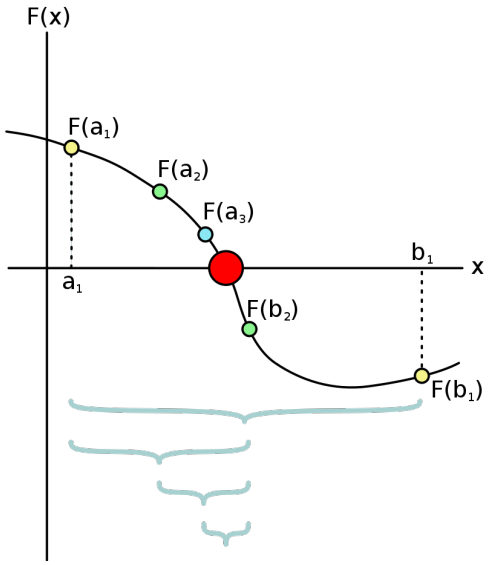

Figure 1: Bisection method graphical representation

It has $\rho = 1$ and C= $\frac{1}{2}$.

2.2.2 Regula-falsi method

Considering $f(x)$ a function, the regula-falsi method is defined by the formula

$$x_{i+1} = x_i - \frac{x_i - x_0}{f(x_i) - f(x_0)} f(x_i) \text{ for } i = 0, 1, 2, 3... \quad (6)$$

Where x_0 is the starting point.

A graphical representation of the regula-falsi method is shown in Figure 2:

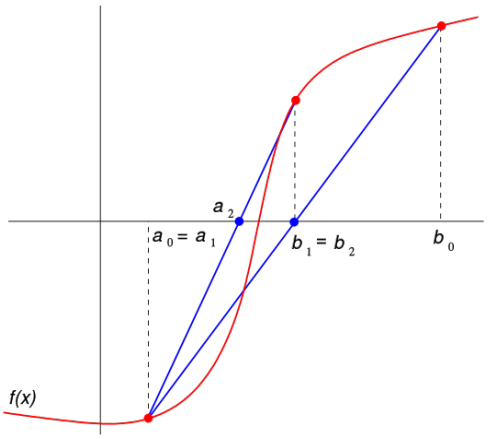

Figure 2: Regula-falsi method graphical representation

It has $\rho = 1$ and C= $\frac{f'(\dot{x})}{f(x_0)}(\dot{x} - x_0) + 1$.

2.2.3 Newton's method

Considering $f(x)$, Newton's tangent method is define by the formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \text{ for } i = 0, 1, 2, 3... \tag{7}$$

where x_i for $i = 0$ is x_0, the starting point. This method can not be used if $f'(x) = 0$.

A graphical example is presented in Figure 3:

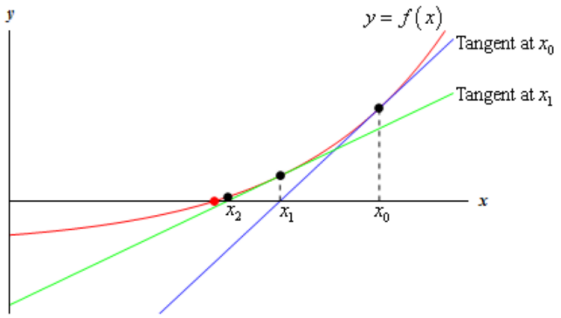

Figure 3: Newton's method graphical representation

It has $\rho = 2$ and C= $-\frac{1}{2}\frac{f''(\dot{x}}{f'(\dot{x})}$.

2.2.4 Muller's method: version II

The Muller's method is based on the approximation of $f(x)$ around x_i using quadratic function.

$$\hat{f}(x - x_i) = a_i(x - x_i)^2 + b_i(x - x_i) + c_i \tag{8}$$

The 2nd version of the method uses the interpolation of $f'(x_{i-2})$, $f'(x_{i-1})$, $f'(x_i)$:

$$\begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{bmatrix} -(x_i - x_{i-1})^2 & (x_i - x_{i-1}) \\ -(x_i - x_{i-2})^2 & (x_i - x_{i-2}) \end{bmatrix}^{-1} \cdot \begin{bmatrix} f(x_i) - f(x_{i-1}) \\ f(x_i) - f(x_{i_2}) \end{bmatrix} \text{ and } c = f(x_i) \tag{9}$$

The iterative formula is

$$x_i = x_i - \frac{2c_i}{b_i + sgn(b_i)\sqrt{{b_i}^2 - 4a_ic_i}} \tag{10}$$

It has $\rho = 1.84$.

3 Assignment B

1. Use MATLAB's function fzero to solve the equation

   $$f(x) = 0 \tag{11}$$

   with

   $$f(x) = 2[exp(-(\frac{x}{8} - 1)^6)]^{12} + 0.001x^3 - 2.5 \text{ for } x \in [1, 10] \tag{12}$$

   Plot the function $f(x)$ for $x \in [1, 10]$. Mark the horizontal line corresponding to y = 0 and the point corresponding to the solution of $f(x) = 0$.

2. . Write a MATLAB function for solving nonlinear algebraic equations using the bisection method. The function should have the following input arguments:

   - a *function_handle*, representing the function $f(x)$ defining the equation to be solved

- a two-element vector $[x_{min}, x_{max}]$, representing the boundaries of the interval of in which the solution is to be sought;
- a scalar, representing the maximum acceptable magnitude Δ of the absolute error of the solution.

The function should return a vector containing the approximations of the solution obtained in consecutive iterations. Test the function by using it for solving $f(x) = 0$ with $\Delta = 10^{-12}$.

3. Write three more MATLAB functions for solving nonlinear algebraic equations using:
   - the regula-falsi method,
   - the Newton's method,
   - version II of Muller's method.

   The syntax of these functions should be analogous to that of the function from Task #2. In the case of regula-falsi method, the values $x_0 = x_{min}$ and $x_1 = x_{max}$ should be the starting points. In the case of the Newton's method, the value $x_0 = x_{max}$ should be the starting point. In the case of the Muller's method, the values $x_0 = x_{min}$, $x_1 = \frac{x_{min}+x_{max}}{2}$ and $x_2 = x_{max}$ should be the starting points. Test the functions by using them for solving $f(x) = 0$ with $\Delta = 10^{-12}$.
4. For all four methods implemented in Tasks #2 and #3, plot the absolute errors of the approximations of the solution, obtained in consecutive iterations for $\Delta = 10^{-12}$. Determine those errors by taking the solution obtained using *fzero* as reference.
5. For all four methods implemented in Tasks #2 and #3, plot the number of iterations performed before reaching the acceptable accuracy of the solution for $\Delta \in [10^{-15}, 10^{-1}]$.
6. Determine the number of iterations necessary to reach the acceptable accuracy of the solution using the bisection method with $\Delta \in [10^{-15}, 10^{-1}]$ according to the formula presented on the lecture slide #4-13. Compare the results with those obtained in Task #5.
7. Plot and compare the number of iterations necessary to reach the acceptable accuracy of the solution using the Newton's method with $\Delta \in [10^{-15}, 10^{-1}]$ for two different starting points: $x_0 = x_{min}$ and $x_0 = x_{max}$

3.1 Task 1: use of *fzero* function

Using the MATLAB's function *fzero* the obtained result is $x = 7.9370$.

The plot of Eq.(12) is obtained by the script Task1.m, that is reported in the appendix (§4.1) , and is presented in Figure 4:

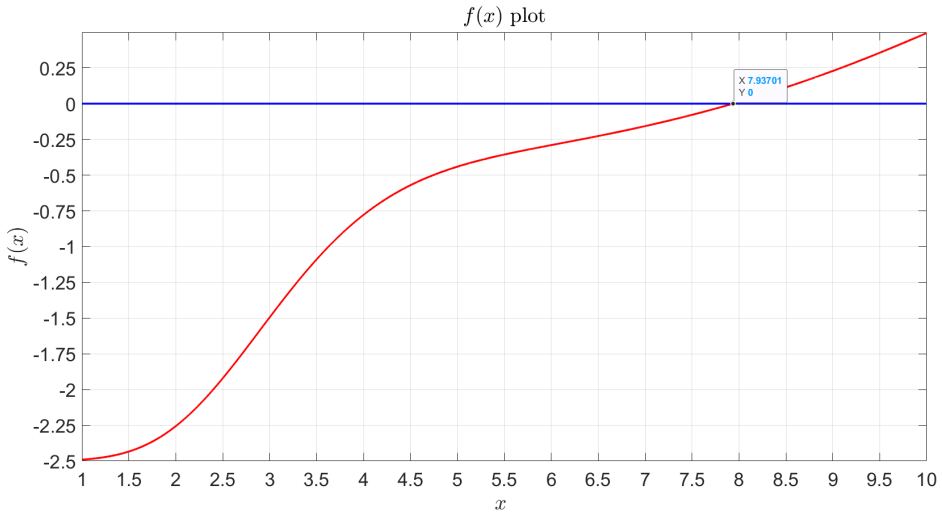

Figure 4: $f(x)$ plot

It shows the solution of Eq.(11).

3.2 Task 2: bisection method

The MATLAB function is reported in the appendix (§4.2) and it applies the Eq.(5). Applying it to the function Eq.(12), the wanted result is obtained after 44 iterations, as shown in Figure 5:

Number of iterations	Result approximations
1	5,500000
2	7,750000
3	8,875000
4	8,312500
5	8,031250
6	7,890625
7	7,960938
8	7,925781
9	7,943359
10	7,934570
11	7,938965
12	7,936768
13	7,937866
14	7,937317
15	7,937042
16	7,936905
17	7,936974
...	...
44	7,937005

Figure 5: Representation of vector of approximated roots by bisection method

The obtained result is comparable with the *fzero* one: $|x_{fzero} - x_{bisect}| = 4.9560 \cdot 10^{-13} < \Delta = 10^{-12}$.

3.3 Task 3: Other methods

3.3.1 Regula-falsi method

The regula-falsi method is implemented using the formula Eq.(6) in the MATLAB script reported in the appendix (§4.3). Applying it to the function Eq.(12), the wanted result is obtained after 39 iterations, as shown in Figure 6:

Number of iterations	Result approximations
1	10,000000
2	8,509596
3	8,174736
4	8,043788
5	7,986397
6	7,960140
7	7,947904
8	7,942153
9	7,939440
10	7,938157
11	0,937551
12	7,937263
13	7,937127
14	7,937063
15	7,937033
...	...
39	7,937005

Figure 6: Representation of vector of approximated roots by regula-falsi method

Also here the result is comparable with the *fzero* one: $|x_{fzero} - x_{Regula}| = 1.9806 \cdot 10^{-13} < \Delta = 10^{-12}$

3.3.2 Newton's method

The method is implemented using the formula Eq.(7) in the MATLAB script in §4.4. The wanted result for Eq.(12) is obtained after 6 iterations, as shown in Figure 7:

Number of iterations	Result approximations
1	10,000000
2	8,244913
3	7,948217
4	7,937014
5	7,937005
6	7,937005

Figure 7: Representation of vector of approximated roots by Newton's method

In this case the final result is comparable with the *fzero* one too:
$|x_{fzero} - x_{Newton}| = 4.3965 \cdot 10^{-13} < \Delta = 10^{-12}$

3.3.3 Muller's method: version II

Muller's method is implemented solving the system in Eq.(9) and using Eq.(10) using the script §4.5. In its application for Eq.(12) the expected result is reached after 6 iteration, as represented in Figure 8:

Number of iterations	Result approximations
1	6,714275
2	7,861275
3	7,936318
4	7,937006
5	7,937005
6	7,937005

Figure 8: Representation of vector of approximated roots by Muller's method

The differences between the final results and the *fzero* one is:
$|x_{fzero} - x_{Muller}| = 1.7764 \cdot 10^{-15} < \Delta = 10^{-12}$.

3.4 Task 4: Absolute error of the solution

In order to confront the progression of the absolute error of all the previous methods (§3.1, §3.2), the absolute error functions about Eq.(12) take $fzero$ result as reference and they are plotted in Figure 9, using the MATLAB script in appendix §4.6:

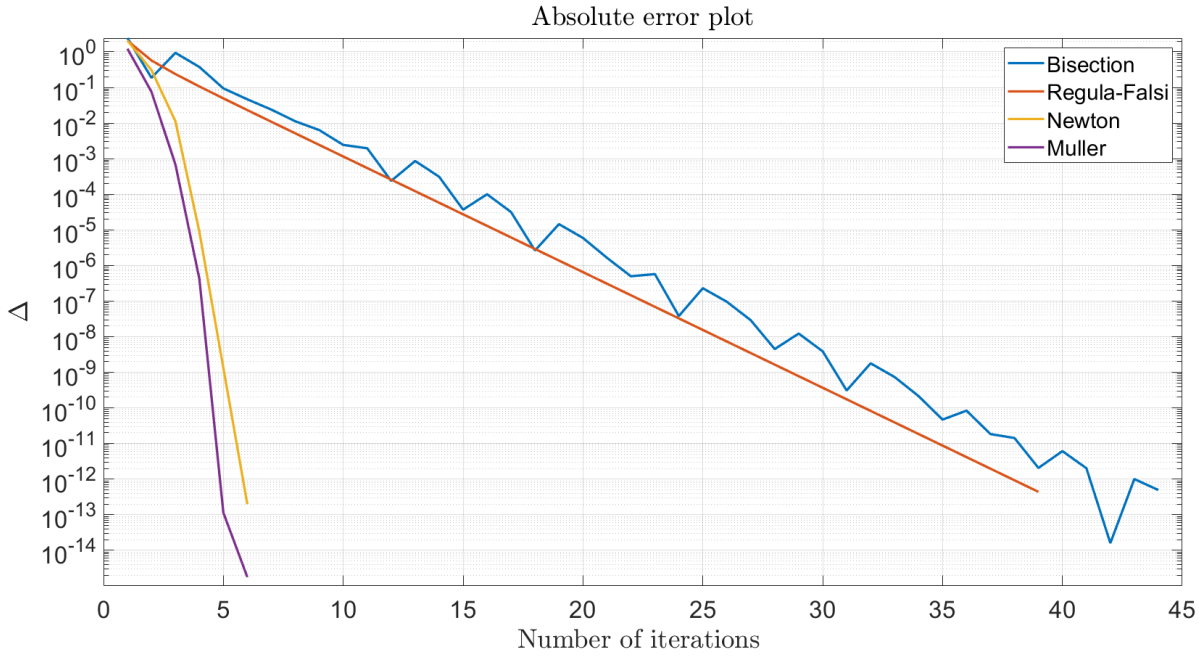

Figure 9: Absolute error progression by iterations

Most of the lines are interrupted because the iterations of the algorithms are blocked when the error is below 10^{-12}.

As expected, the Muller and Newton methods are the fastest in reducing the absolute error at each iteration, while the regula-falsi and bisection ones are slower. The reason behind this behaviour is explained in §2, where it is said that the coefficient ρ determines the speed of the convergence, the higher it is, the faster is the convergence. Indeed $\rho_{Muller} = 1.84$ and $\rho_{Newton} = 2$ that are higher than $\rho_{Regula} = \rho_{Bisection} = 1$.

3.5 Task 5: Number of iterations analysis

For the purpose of calculating the number of iterations with different absolute error requirements, the script "Task5.m" in the appendix (§4.7) is used. In Figure 10 there is the representation of them considering all the methods (§3.1, §3.2) applied to the function Eq.(12):

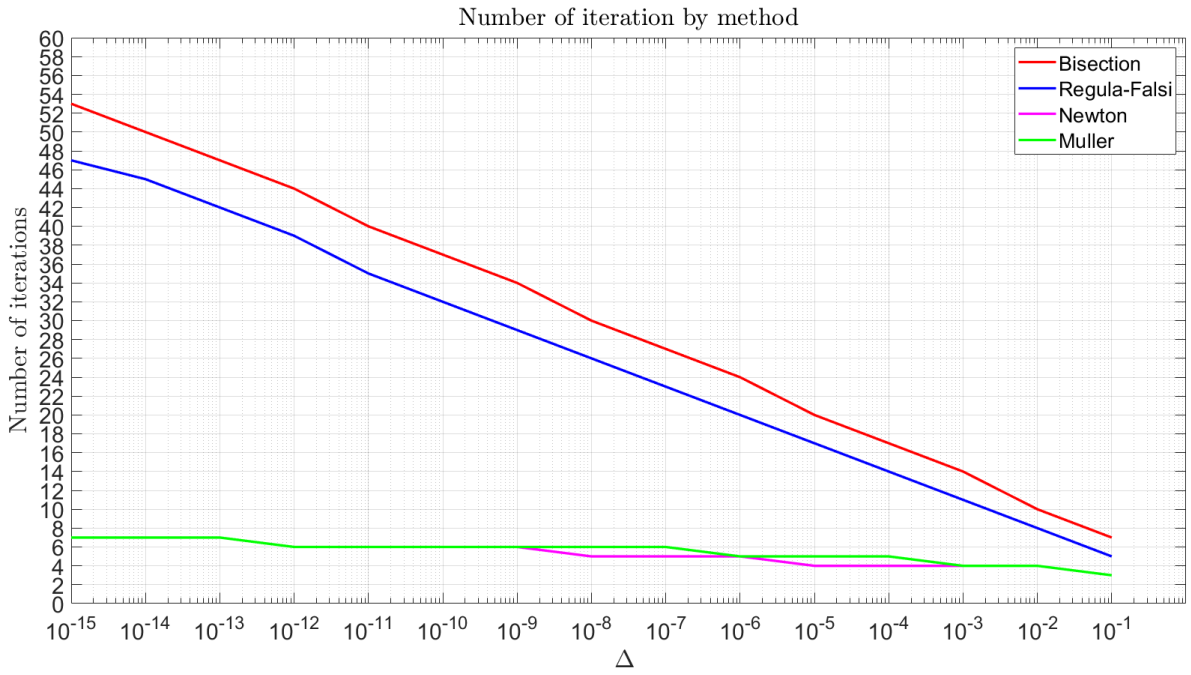

Figure 10: Number of iterations by absolute error requirements

One can observe that the number of iterations required by regula-falsi and bisection methods for a high precision result is almost logarithmically linear, while for Muller's and Newton's methods the iterations needed are way more fewer and their graph is quasi-constant.

For instance, if we consider the highest precision ($\Delta = 10^{-15}$) on Figure 10, the iterations required for Muller's and Newton's method are only 7, in confront of the 47 iterations required by regula-falsi one and the 53 required by the bisection one.

As said in §3.4, the reason behind this behaviour is the speed of convergence for each algorithm.

3.6 Task 6: Iterations and precision of the bisection method

The formula used to calculate the number of iterations needed to reduce the absolute error below Δ is:

$$I = min\left\{ i > \log_2 \frac{|b-a|}{\Delta} \right\} \tag{13}$$

Where:

- i is the iteration
- a and b are the bounds of the interval $[a, b]$ in which the root is calculated

Using this formula for Eq.(12) in the script §4.8 and comparing its plot with the one obtained in Task 5 (§3.6), the obtained plot is in Figure 11:

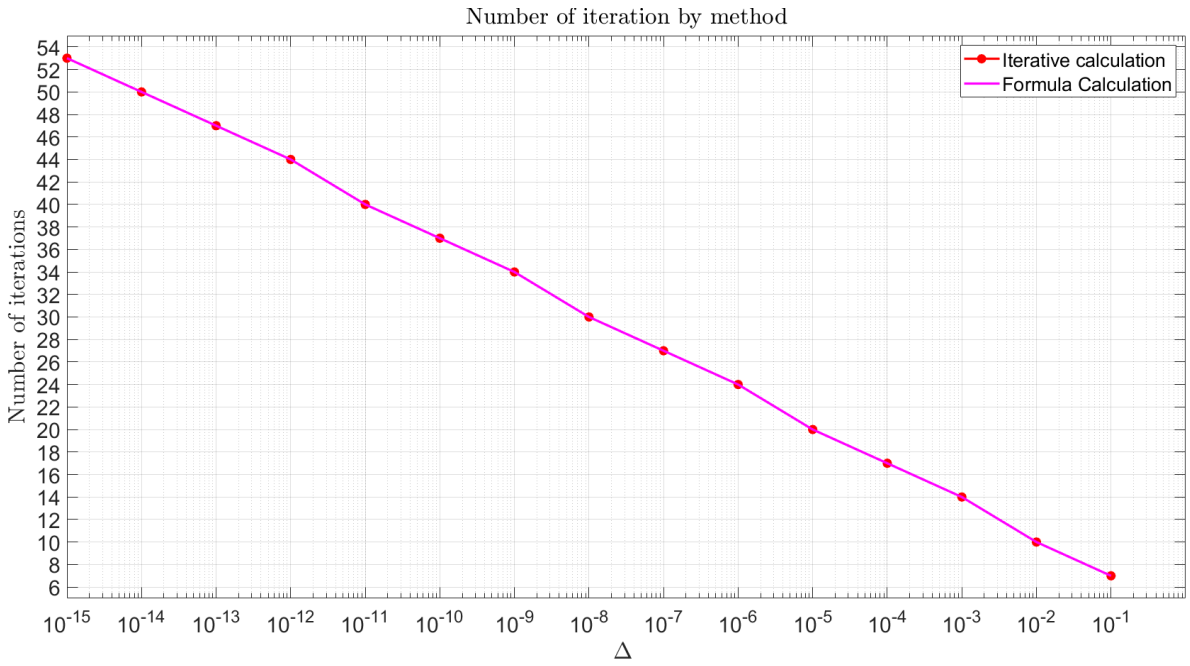

Figure 11: Number of iterations needed for each magnitude of absolute error

One can observe that the plots are identical. In order to check it let's see the output values from script §4.8 in Figure 12:

	Iterative calculation	Formula calculation
1,E-15	53	53
1,E-14	50	50
1,E-13	47	47
1,E-12	44	44
1,E-11	40	40
1,E-10	37	37
1,E-09	34	34
1,E-08	30	30
1,E-07	27	27
1,E-06	24	24
1,E-05	20	20
1,E-04	17	17
1,E-03	14	14
1,E-02	10	10
1,E-01	7	7

Figure 12: Outputs from the calculation by script §4.8

As expected, the output datas are the same, so iterative algorithm executed by the script in §4.2 respects the theoretical formula presented in Eq.(13).

3.7 Task 7: Newton's method iterations by changing starting point

With the intention of confront the number of iterations buy considering different starting points ($x = 1$ and $x = 10$), the same Newton's function script (§4.4) have been used in §4.9 on Eq.(12), which plot is represented in Figure 13:

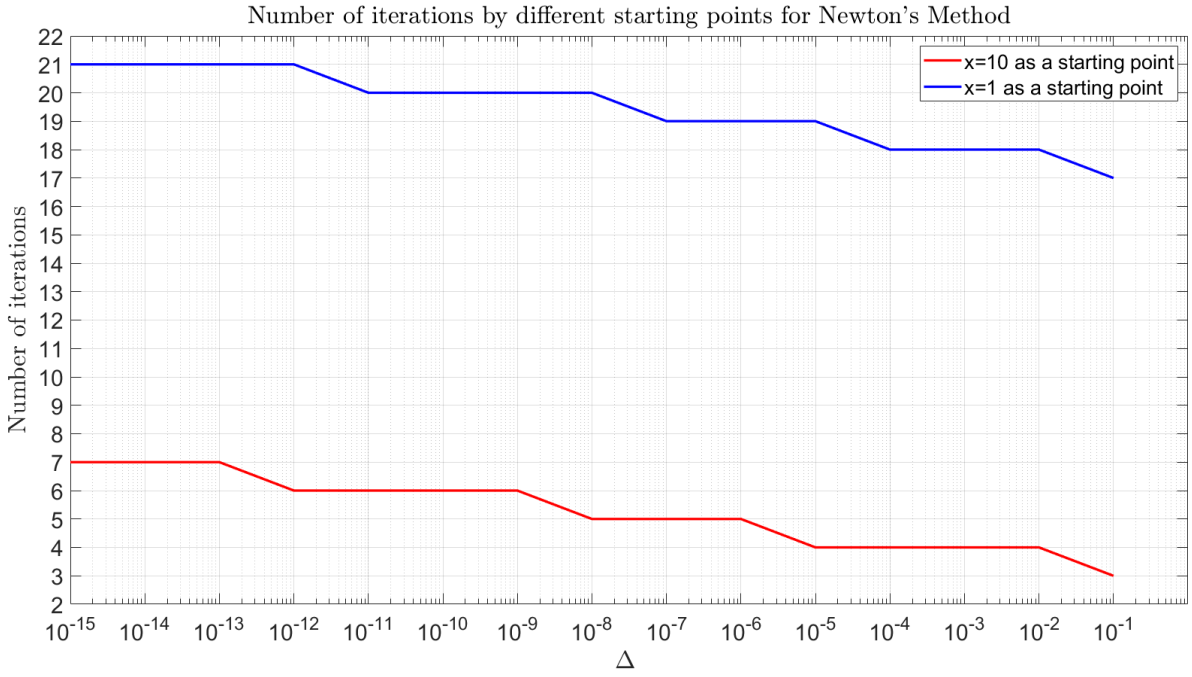

Figure 13: Number of iterations need for each magnitude of absolute error

It can be noticed that the number of iterations changes significantly depending on the starting point. As shown in §2.2.3, Newton's method uses the derivative in function's points, in other words, the slope of the function's tangent in each point: so the closer is the initial guess, the fewer number of iterations are needed. As we known from Task 1 (§3.1), the root is located in $x = 7.9370$ which is closer to $x = 10$ than $x = 1$. 21 iterations are the maximum iteration that can be done in the interval $[1, 10]$, because $x = 1$ is the furthest point from 7.9370.

One can also observe that the behaviour of the plot is the same: indeed, considering $x = 1$ the maximum number of iteration is 21 and the minimum one is 17, while for $x = 10$ the maximum one is 7 and the minimum one is 3. So the difference between the maximum and the minimum is 4 for both the cases, which means that the convergence speed from $\Delta = 10^{-1}$ to $\Delta = 10^{-15}$ is the same.

In conclusion, Newton's iterative algorithm have the same convergence regardless of the starting point, as expected from the parameters ρ and C from §2.2.3, which do not depend on the starting point.

4 Appendix: MatLab codes used

4.1 Task1.m

```
f=@(x) 2*[exp(-(x/8 - 1).^6)].^12 +0.001*x.^3 - 2.5;

%defining limits
x_0=[1 , 10];

%calculating the zero
z=fzero(f, x_0);
disp(z);

x = linspace(x_0(1) , x_0(2), 10000);
y=f(x);

lvl_zero= zeros(1, length(x));

%plotting
plot (x ,y ,'r', 'LineWidth', 2);
hold on;
plot(x, lvl_zero, 'b', 'Linewidth', 2, 'LineStyle','-');
grid on;

title ("$f(x)$ plot" , 'FontSize', 25, 'Interpreter', 'latex') ;
yticks(-2.5 :0.25 :max(y));
xticks(x_0(1) : 0.5 : x_0(2));
xlabel (" $x$ ", 'FontSize', 25,  'Interpreter', 'latex') ;
ylabel ("$f(x)$" , 'FontSize', 25, 'Interpreter', 'latex') ;
set ( gca ,'FontSize' ,20) ;
```

4.2 bisectionMethod.m

```
function  zero = bisectionMethod(f, interval, Delta)

    %setting initial values
    x_min=interval(1);
    x_max=interval(2);

    %inizializing the output
    zero = [];

    while abs(x_max - x_min) > Delta
    %calculating the midpoint
    x_curr= (x_max+x_min)/2;

        if f(x_min) * f(x_curr) < 0
            x_max=x_curr;
        else
            x_min = x_curr;
        end

    %appenending the new approximation

    zero = [zero, x_curr];
    end
end
```

4.3 Regula_FalsiMethod.m

```
function zero = Regula_FalsiMethod(f, interval, Delta)

    %setting initial values
    x_0=interval(1);
    x_curr=interval(2);

    %initializing the previous approximation with x_0: it is done only
       for
    %the first iteration.
    x_prev=x_0;

    %initializing output vector
    zero=x_curr;

    while abs(x_curr- x_prev) > Delta
        x_next = x_curr - (f(x_curr)) *  (x_curr-x_0)/(f(x_curr)-f(x_0))
           ;

        %saving the previous approximation in order to calculate the
           error
        x_prev=x_curr;

        %updating the approximation
        x_curr = x_next;

        %appending the approximation
        zero=[zero, x_curr];

    end

end
```

4.4 newtonMethod.m

```
function zero = newtonMethod(f, interval, Delta)
    %h is used for the derivative calculation
    h=10^(-12);

    %initializing the initial value
    x_next= interval(2);
    x_curr=interval(1); %note that this is an arbitrary value in order
       to start the loop

    %initializing the output
    zero=[];

    while  abs(x_curr- x_next) > Delta
        %updating the value
        x_curr = x_next;

        %calculation of the derivative
        df_curr = (f(x_curr+h) - f(x_curr-h)) / (2*h);

        %calculation of the value
        x_next= x_curr - (f(x_curr))/df_curr;
```

```
        %appending the new approximation
        zero = [zero , x_curr];
    end
end
```

4.5 mullerMethod.m

```
function zero = mullerMethod(f, interval, Delta)

    %Initializing the values
    x_curr=interval(2);
    x_old=(interval(1)+interval(2))/2;
    x_oldold=interval(1);

    %I will use this cofficients:
    %Coeff= [a_curr, b_curr]' ;
    %c_curr;

    %defining output
    zero=[];

    while abs(x_curr - x_old ) > Delta


        %defining the matrices
        A = [ -(x_curr - x_old)^2 , (x_curr  - x_old) ;  -(x_curr -
            x_oldold)^2 , (x_curr-x_oldold)];
        b = [f(x_curr) - f(x_old) ; f(x_curr) - f(x_oldold)];

        %calculation of the coefficients
        Coeff = A\b;
        c_curr=f(x_curr);

        %iteration of the formula

        x_next = x_curr - (2*c_curr)/ (Coeff(2) + sign(Coeff(2)) * sqrt
            ((Coeff(2))^2 - 4*Coeff(1)* c_curr));

        %updating the values
        x_oldold=x_old;
        x_old=x_curr;
        x_curr=x_next;

        %appending the new approximation
        zero = [zero , x_curr];
    end
end
```

4.6 Task4.m

```
f=@(x) 2*[exp(-(x/8 - 1).^6)].^12 +0.001*x.^3 - 2.5;

x_0=[1 , 10];
Delta= 10^(-12);

z0 = fzero(f,x_0);
```

```
%calculating the absolute error for Bisection
z1=bisectionMethod(f,x_0,Delta);
AbError1= abs(z1 - z0);

%Absolute error for Regula-falsi method
z2=Regula_FalsiMethod(f,x_0,Delta);
AbError2= abs(z2-z0);

%Absolute error for Newton's method
z3=newtonMethod(f,x_0,Delta);
AbError3=abs(z3-z0);

%Absolute error for Muller's method
z4=mullerMethod(f,x_0,Delta);
AbError4=abs(z4-z0);

%In order to plot everything, I need to put every array at the same
    length;
len=max([numel(AbError4), numel(AbError3), numel(AbError2), numel(
    AbError1)]);

%building the array full of zeros to append for each function
App1=zeros(1,len-length(AbError1));
App2=zeros(1,len-length(AbError2));
App3=zeros(1,len-length(AbError3));
App4=zeros(1,len-length(AbError4));

%appending
AbError1 = [AbError1, App1];
AbError2 = [AbError2, App2];
AbError3 = [AbError3, App3];
AbError4 = [AbError4, App4];


%plotting the absolute error

semilogy(AbError1, 'LineWidth', 2);
hold on;
semilogy(AbError2, 'LineWidth', 2);
hold on;
semilogy(AbError3, 'LineWidth', 2);
hold on;
semilogy(AbError4, 'LineWidth', 2);
hold off;

grid on;

title ("Absolute error plot" , 'FontSize', 25, 'Interpreter', 'latex') ;
yticks(10.^(-14:1:2));
legend('Bisection', 'Regula-Falsi', 'Newton', 'Muller');
xlabel (" Number of iterations ", 'FontSize', 25,  'Interpreter', 'latex
    ') ;
ylabel ("$\Delta$" , 'FontSize', 25, 'Interpreter', 'latex') ;
set ( gca ,'FontSize' ,20) ;
```

4.7 Task5.m

```
Delta=[10.^(-15:1:-1)];

f=@(x) 2*[exp(-(x/8 - 1).^6)].^12 +0.001*x.^3 - 2.5;
x_0=[1 , 10];

%Number of iteration of a bisection method

%initializing the array where I want to put the result of each iteration
    of the for loop
It1=zeros(size(Delta));
for i = 1:numel(Delta)
    z= bisectionMethod(f,x_0, Delta(i));
    It1(i) = numel(z);
end

%Number of iteration for Regula-falsi method
It2=zeros(size(Delta));
for i = 1:numel(Delta)
    z= Regula_FalsiMethod(f,x_0, Delta(i));
    It2(i) = numel(z);
end

%Number of iteration for Newton's Method
It3=zeros(size(Delta));
for i = 1:numel(Delta)
    z= newtonMethod(f,x_0, Delta(i));
    It3(i) = numel(z);
end

%Number of iteration for Muller's method
It4=zeros(size(Delta));
for i = 1:numel(Delta)
    z = mullerMethod(f,x_0, Delta(i));
    It4(i) = numel(z);
end

%plotting the results

semilogx( Delta, It1,'r', 'LineWidth', 2);
hold on;
semilogx( Delta, It2, 'b', 'LineWidth', 2);
hold on;
semilogx( Delta, It3,'m', 'LineWidth', 2);
hold on;
semilogx( Delta, It4,'g',  'LineWidth', 2);
hold on;
grid on;

title ("Number of iteration by method" , 'FontSize', 25, 'Interpreter',
   'latex') ;
legend('Bisection', 'Regula-Falsi', 'Newton', 'Muller');
xlabel ("  $\Delta$ ", 'FontSize', 25,  'Interpreter', 'latex') ;
ylabel ("Number of iterations" , 'FontSize', 25, 'Interpreter', 'latex')
    ;

set ( gca ,'YTick', 0:2:60, 'XTick', 10.^(-15:1:-1), 'FontSize' ,18) ;
```

4.8 Task6.m

```
Delta=[10.^(-15:1:-1)];
f=@(x) 2*[exp(-(x/8 - 1).^6)].^12 +0.001*x.^3 - 2.5;
x_0=[1 , 10];

%Initializing I
I=zeros(size(Delta));

%Calculation according to the formula
I = ceil(log2((abs(x_0(2)-x_0(1)))./Delta));

%Calculation of the iterations by Task 5;
It1=zeros(size(Delta));
for i = 1:numel(Delta)
    z= bisectionMethod(f,x_0, Delta(i));
    It1(i) = numel(z);
end


semilogx(Delta, It1, 'ro-', 'LineWidth', 2, 'MarkerFaceColor', 'r', '
   MarkerSize', 6);

hold on;
semilogx(Delta,I,'m', 'LineWidth', 2);
hold off;
grid on;

title ("Number of iteration by method" , 'FontSize', 25, 'Interpreter',
   'latex') ;
legend('Iterative calculation', 'Formula Calculation' );
xlabel ("  $\Delta$ ", 'FontSize', 25,  'Interpreter', 'latex') ;
ylabel ("Number of iterations" , 'FontSize', 25, 'Interpreter', 'latex')
    ;

set ( gca ,'YTick', 0:2:60, 'XTick', 10.^(-15:1:-1), 'FontSize' ,18) ;
```

4.9 Task7.m

```
f=@(x) 2*[exp(-(x/8 - 1).^6)].^12 +0.001*x.^3 - 2.5;

x_0=[1 , 10];
Delta=[10.^(-15:1:-1)];

%in order to change the starting point of the Newton's algorithm, I
   create
% a x_1 vector which cointains the boundaries of the interval in
   reversed
%order.

x_1=[10,1];

%calculating the number of iterations with x=10 as a starting point
It1=zeros(size(Delta));
for i = 1:numel(Delta)
    z= newtonMethod(f,x_0, Delta(i));
    It1(i) = numel(z);
end
```

```
%calculating the number of iterations with x=1 as a starting point
It2=zeros(size(Delta));
for i = 1:numel(Delta)
    z= newtonMethod(f,x_1, Delta(i));
    It2(i) = numel(z);
end

semilogx( Delta, It1,'r', 'LineWidth', 2);
hold on;
semilogx( Delta, It2, 'b', 'LineWidth', 2);
hold off;
grid on;

title ("Number of iteration by different starting points for Newton's
   Method" , 'FontSize', 25, 'Interpreter', 'latex') ;
legend('x=10 as a starting point', 'x=1 as a starting point');
xlabel ("  $\Delta$ ", 'FontSize', 25,  'Interpreter', 'latex') ;
ylabel ("Number of iterations" , 'FontSize', 25, 'Interpreter', 'latex')
    ;

set ( gca ,'YTick', 0:1:22,  'XTick', 10.^(-15:1:-1), 'FontSize' ,18) ;
```

5 References

Wikipedia: Bisection method (https://en.wikipedia.org/wiki/Bisection_method)

Wikipedia: Regula falsi method (https://en.wikipedia.org/wiki/Regula_falsi)

Paul's online note: Newton's method (https://tutorial.math.lamar.edu/classes/calci/newtonsmethod.aspx)

Roman Morawski: Numerical Methods, Lecture Slides.