[ENUME] Report of a project no.2

Jan Świerczyński

293660

Task no.1

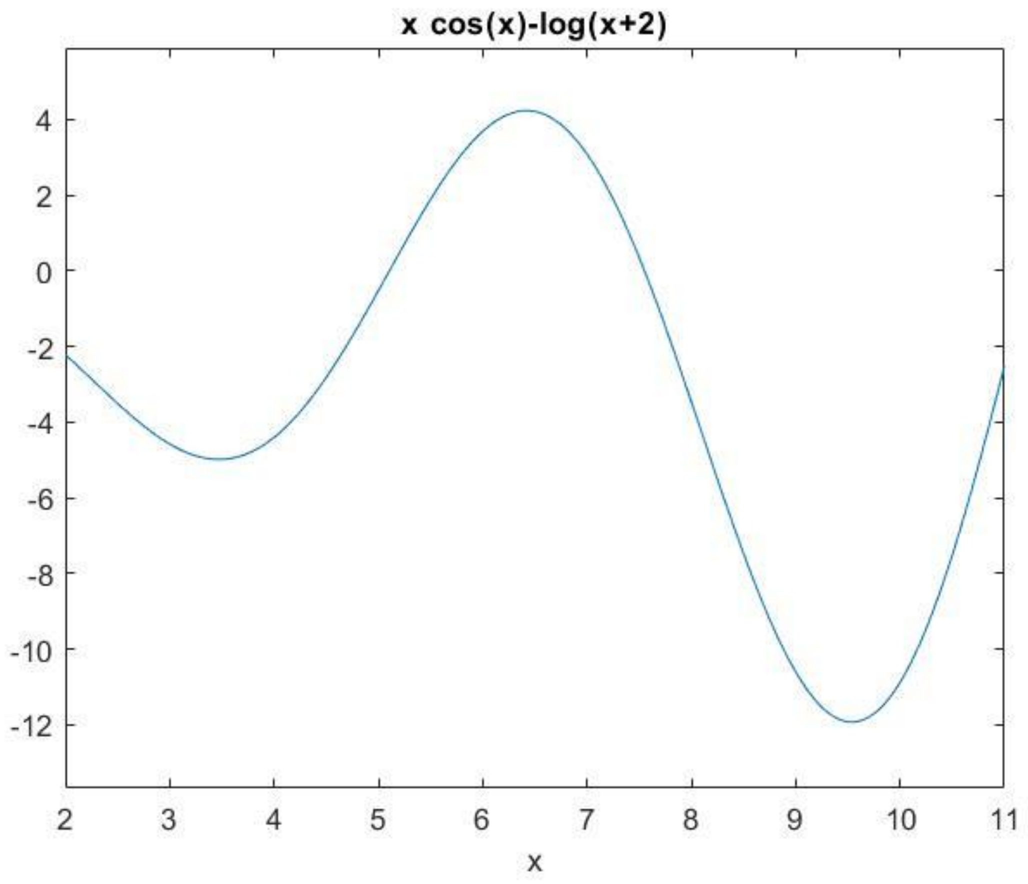
In task number one, we were assigned to write a program, which will be designed to find all zeros of a given function, using two methods/algorithms.

In my case it was:

- The bisection method,
- The Newton's method,

I had to find zeros on the interval x€[2, 11] for the function f(x) = x*cos(x) – ln(x+2).

Which looked like this (on a mentioned interval):

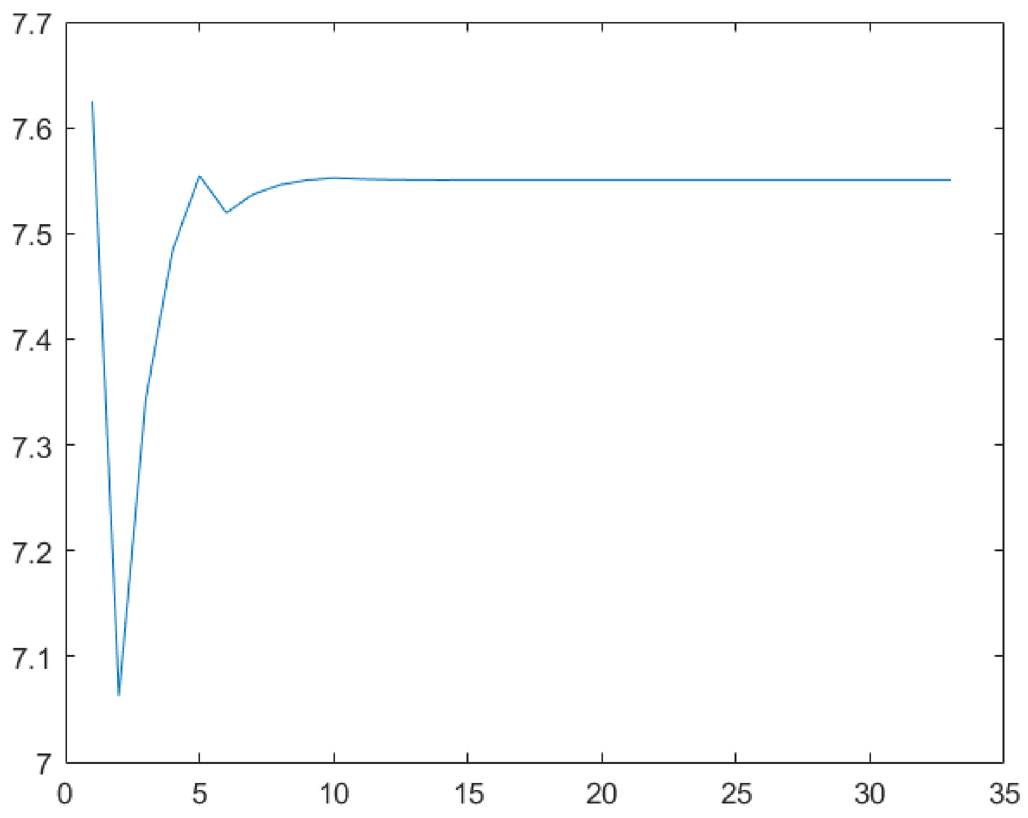


We can observe, that this function has got two roots, on this plot. One of them is on the subinterval x_1€[2, 6.5], second one is on another subinterval x_2€[6.5, 11].

Bisection method: In bisection method, we are starting from selecting a valid subinterval [a, b], which will isolate our root. As I mentioned before, those selected subintervals are [2, 6.5] and [6.5, 11], for x_1, x_2 respectively. Depending on which root we want to calculate, our subinterval isolating selected root, is divided into two equal parts. The point ‘cutting’ those two parts we call middle point, “c”. Then we calculate the value of the function in point “c” and we obtain our first approximation of a zero point of the function. Our next step in the algorithm is to calculate the product of f(a)*f(c) and f(b)*f(c). Our new subinterval for next iteration is chosen from the negative product of one of those equations. (If f(b)*f(c) is smaller than 0, then we have to assign our obtained c value as our new a, etc). Iterations are repeated until absolute value of f(c) is bigger than 10^{-10}. (the error that I had chosen for a stop test for all the methods).

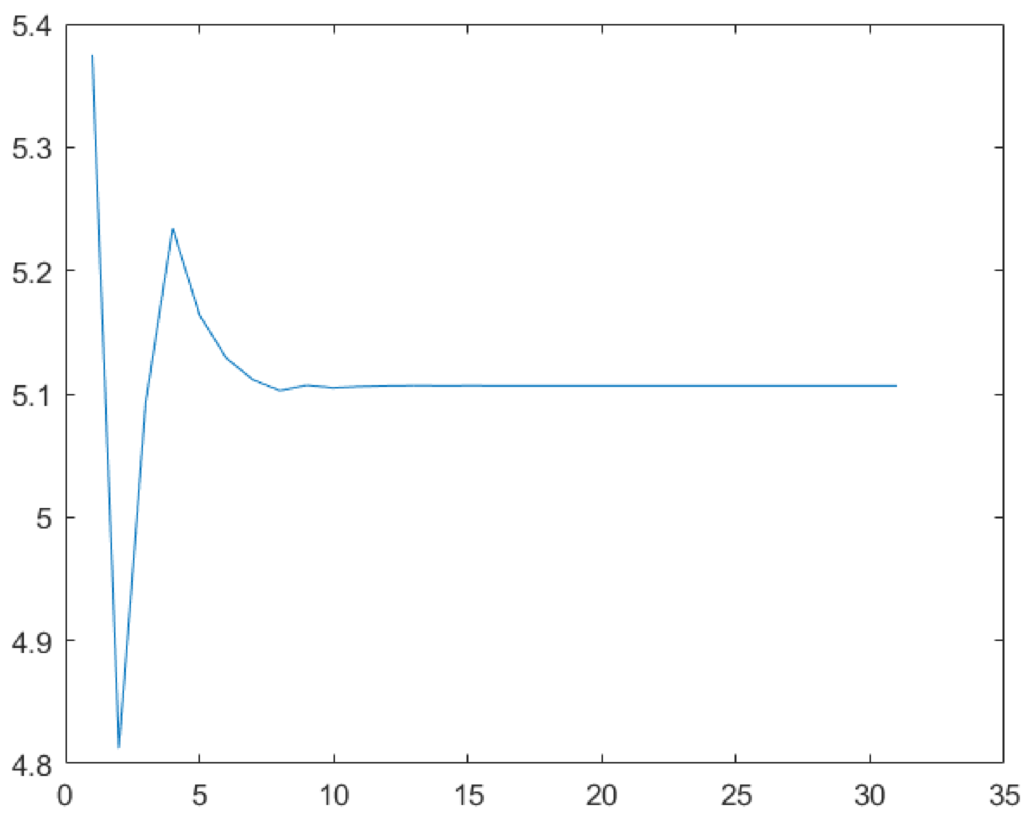
Results:

```
Root between 2 and 6.5 is:
    5.1065
Number of iterations:
    32
Error:
   7.2472e-10
```



Plot showing how the approximation of a root was changing with every iteration.

```
Root between 6.5 and 11 is:
    7.5505
Number of iterations:
    34
Error:
   7.6411e-10
```



Plot showing how the approximation of a root was changing with every iteration.

Fortunately given function was not a “flat” function (given f-ction has got sinusoidal form), in case of, as I called it, “flat” function (derivative of the function is really small in a close intervals to the root) the bisection method could simply fail.

Newton's method: is a recursive algorithm for approximating the root of a differentiable function, also called tangent method. It is approximating the root of the function f(x) operating on the first order of its expansion into Taylor series at a given starting point.

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n).$$

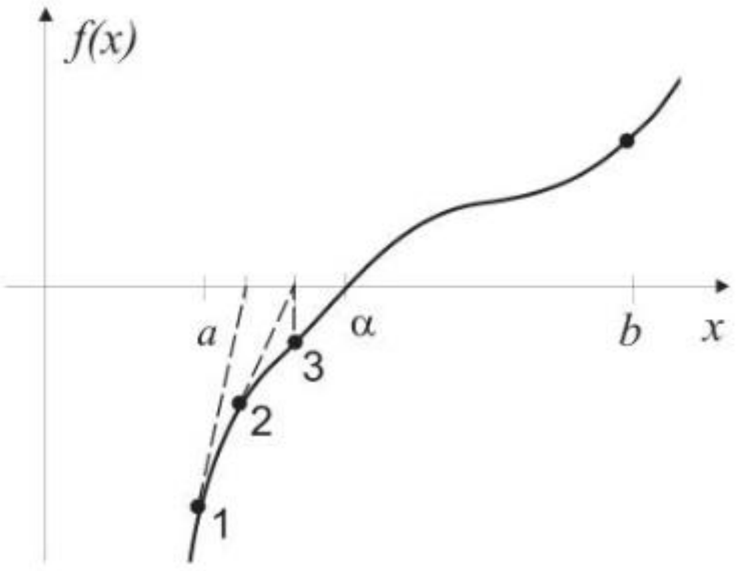
Next point x_{n+1} results as a root of the obtained linear function:

$$f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0,$$

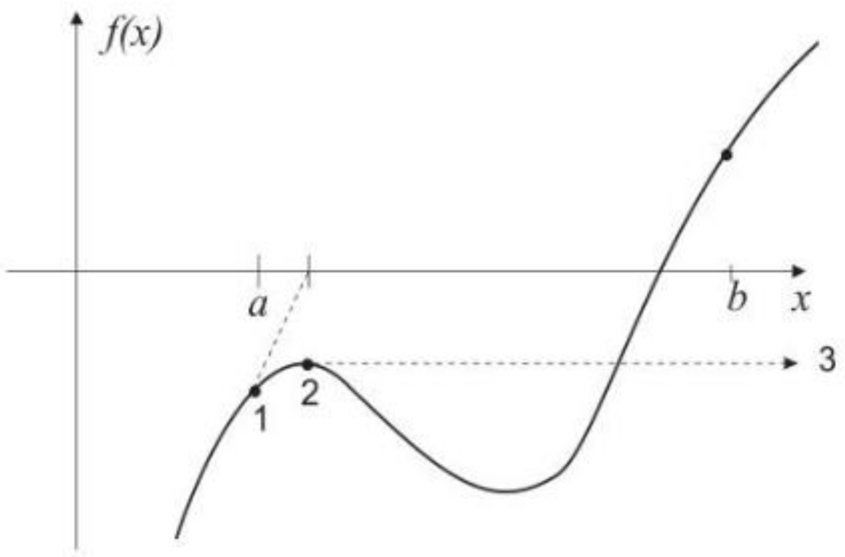
Thus, we are given a formula:

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

Illustration of a Newton's method:



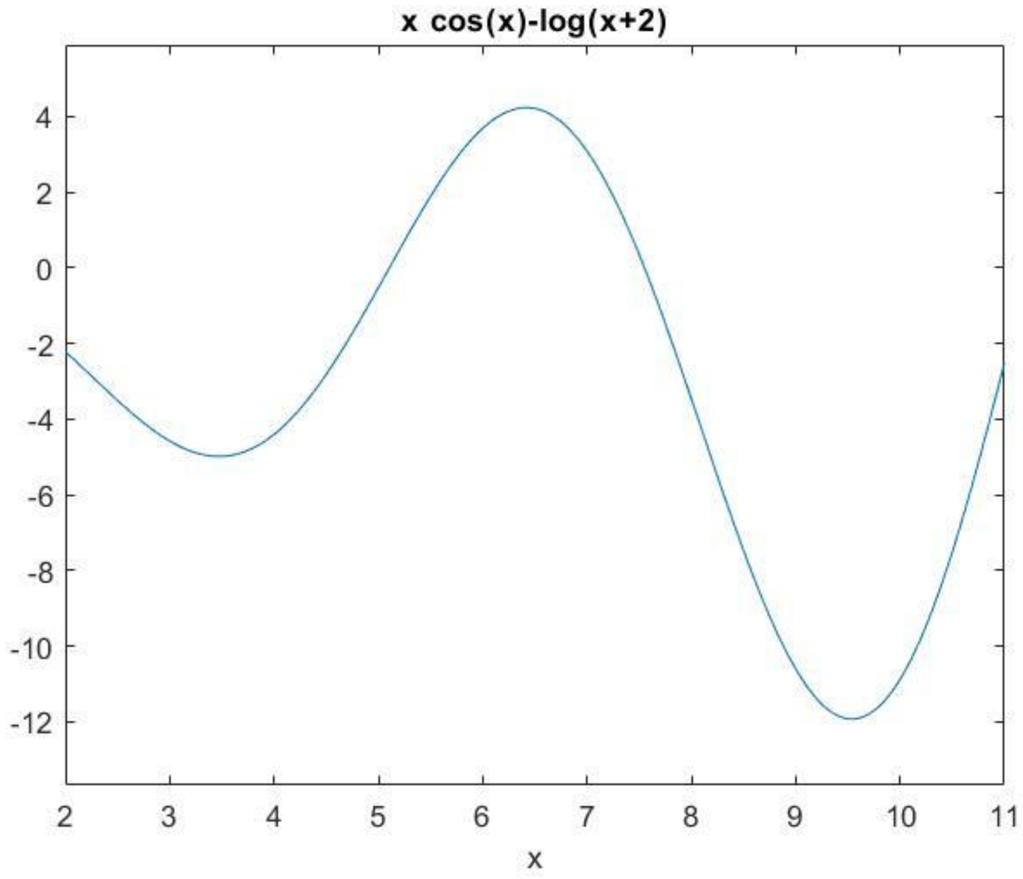
What we can say about this method? It is locally convergent, if an initial point is too far from the root than the divergence may occur. Example:



However, Newton's method is very fast, it's convergence is quadratic.

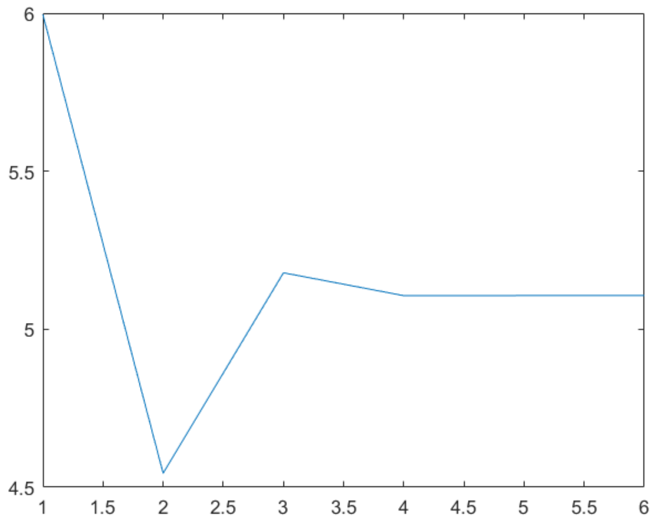
Again method is not recommended to use for a flat functions, as we can see on a pictures above the slope of the function has to be steep.

Graph of the function is the same as for the bisection method.



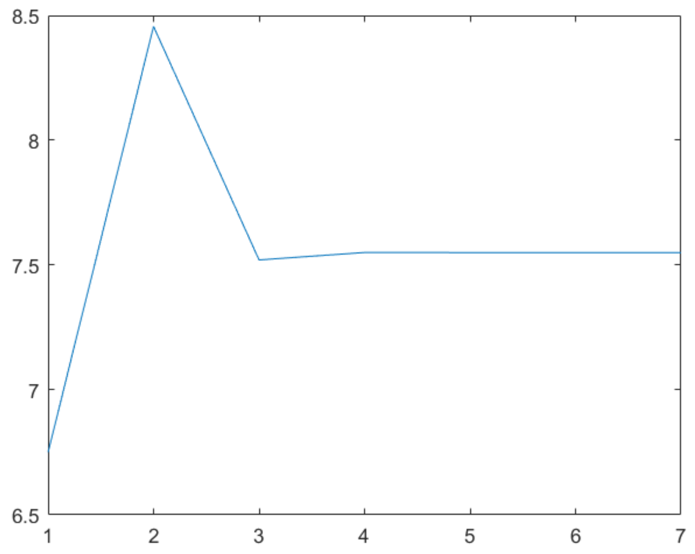
Obtained results for the first root:

```
**NEWTON's METHOD**
X:
    5.1065

Error:
   3.7536e-10

Iterations:
     6
```



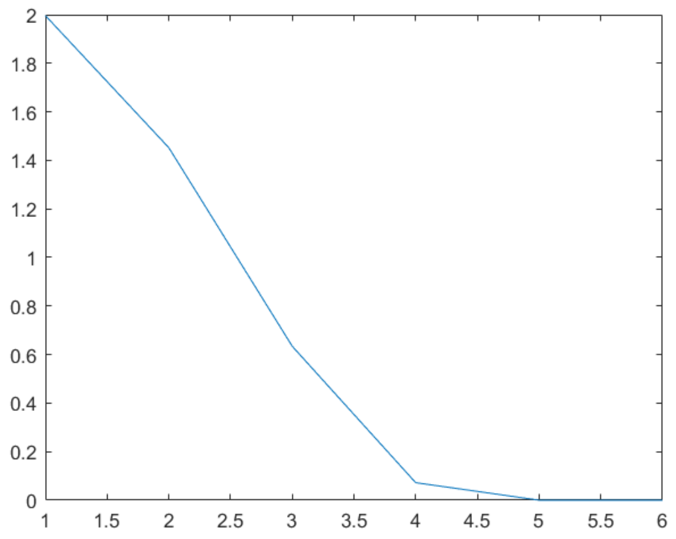
Plot showing how the approximation of a root was changing with every iteration.

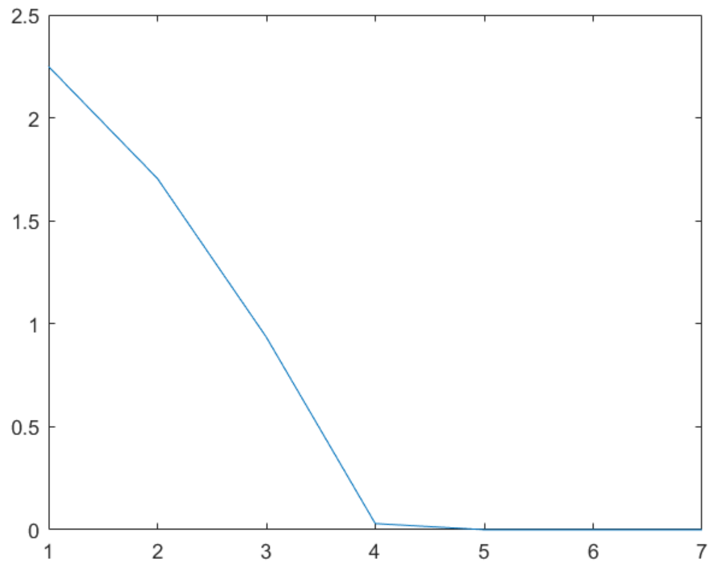
```
**NEWTON's METHOD**
X:
    7.5505

Error:
     0

Iterations:
     7
```



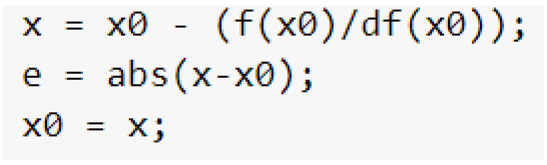
Plot showing how the approximation of a root was changing with every iteration.

We can see, that the error of a given solution is slightly bigger compared to the bisection method, but the number of iterations is five times smaller. Stop condition for both methods was the same (when the error reaches 10^{-10}).





Plot showing how the error was changing with every iteration.
The error for the Newton's method was calculating from absolute value of x – x0 after each iteration :



```
x = x0 - (f(x0)/df(x0));
e = abs(x-x0);
x0 = x;
```

where "e" stands for an error.

Task no.2

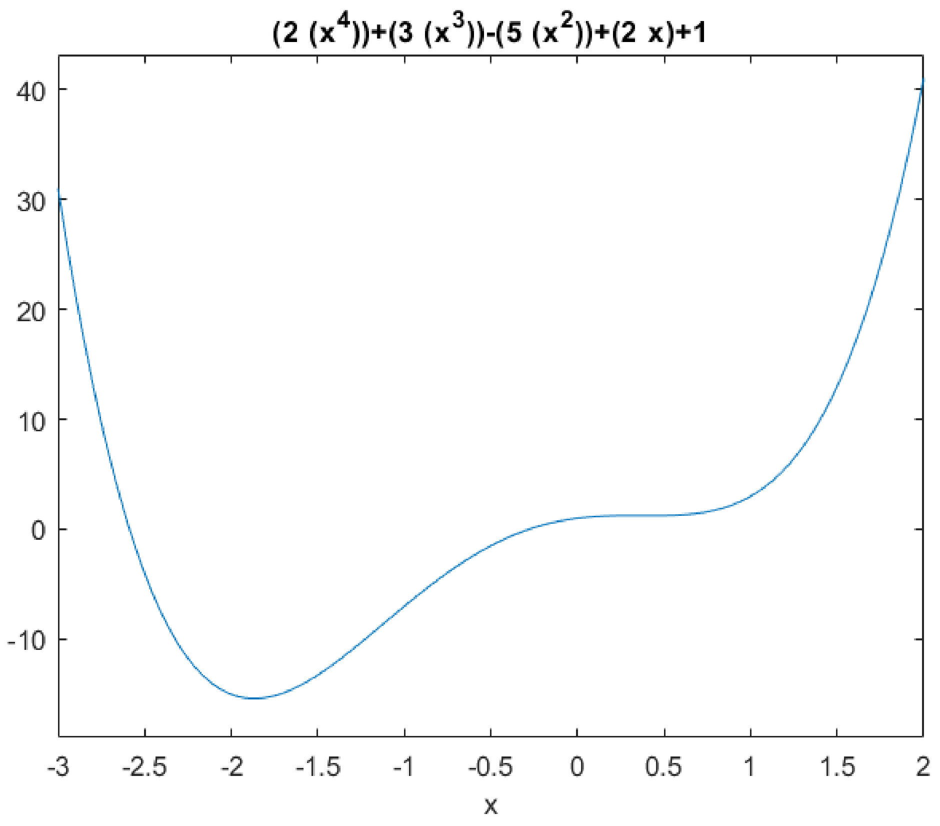
In task number two, we were asked to find all (real and complex) roots of the polynomial. Mine polynomial looked like this:

$f(x) = a_4 * x^4 + a_3 * x^3 + a_2 * x^2 + a_1 * x + a_0;$

Where:

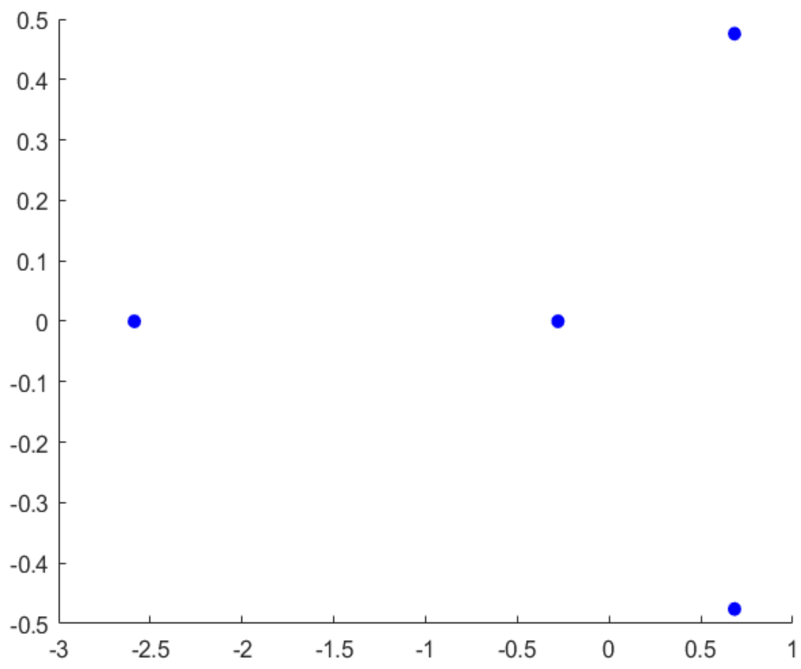
$[a_4\ a_3\ a_2\ a_1\ a_0]$ = [2 3 -5 2 1]

Plot of the function:



```
r = 4×1 complex
   -2.5869 + 0.0000i
    0.6829 + 0.4759i
    0.6829 - 0.4759i
   -0.2790 + 0.0000i
```

Roots obtained from the matlab function.



We had to use Mullers method's (MM1 and MM2) and compare their results.

Muller's Method: the idea of the Muller's method is to approximate the polynomial locally in a neighbourhood of a root by a quadratic function. Basically it is a generalized secant method, where linear interpolation based on two points is applied. There is also a possibility of realization the algorithm using only one point (MM2).

MM1: We are taking under consideration three points x_0, x_1, x_2, together with the polynomial values in those points (f(x_0), f(x_1), f(x_2)). We are creating a quadratic function, which is going through each of those points. The next step is to find a roots of our new function, and one of them becomes the approximation of the polynomial. According to the book I took x_2 as my approximation of a solution and used presented algorithm.

$$z_0 = x_0 - x_2,$$
$$z_1 = x_1 - x_2.$$

Quadratic equation:

$$y(z) = az^2 + bz + c.$$

Calculation of a, b and c of the parabola:

$$\begin{aligned} az_0^2 + bz_0 + c = y(z_0) &= f(x_0), \\ az_1^2 + bz_1 + c = y(z_1) &= f(x_1), \\ c = y(0) &= f(x_2). \end{aligned}$$

=

$$\begin{aligned} az_0^2 + bz_0 &= f(x_0) - f(x_2), \\ az_1^2 + bz_1 &= f(x_1) - f(x_2). \end{aligned}$$

And finally formulas for the roots of the parabola:

$$z_+ = \frac{-2c}{b + \sqrt{b^2 - 4ac}},$$
$$z_- = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.$$

From those roots we are choosing the one with smaller absolute value, and we are adding chosen one to the x_2 to get our approximation of x (repeating after each iteration).

$$x_3 = x_2 + z_{\min},$$

where

$$\begin{aligned} z_{min} &= z_+, \quad \text{if } |b+\sqrt{b^2-4ac}| \geq |b-\sqrt{b^2-4ac}|, \\ z_{min} &= z_-, \quad \text{in the opposite case.} \end{aligned}$$

For next iteration we are selecting only points that are closer to the x_3.

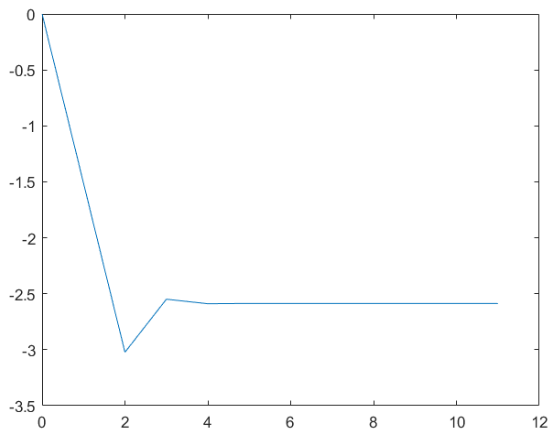
Note: MM1 is working as good for real and complex roots.

Implementation of the function MM1:

```
MM1(-2.9, -3.1, -40, f);% showing real root
MM1(0, -1, 100, f);  % showing real root
MM1(0.7, 0.8, 10+10i, f); % showing complex root
MM1(0.7, 0.8, 10-10i, f); % showing complex root
```

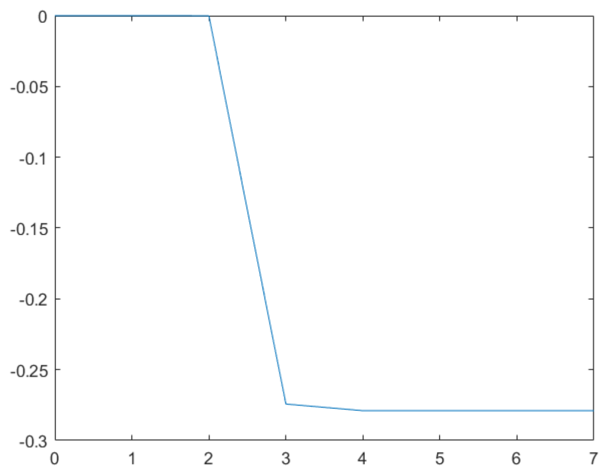
Outputs:

```
** MM1 **
X:
   -2.5869

Error:
   2.1095e-10

Iterations:
    11
```



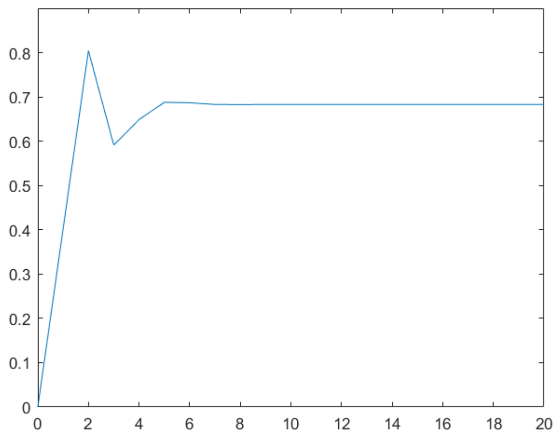
Plot showing how the approximation of a root was changing with every iteration.

```
X:
   -0.2790

Error:
   3.6716e-10

Iterations:
     7
```



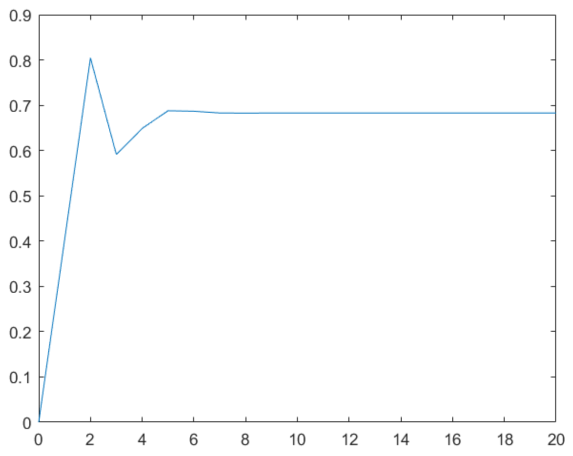
Plot showing how the approximation of a root was changing with every iteration.

```
X:
   0.6829 + 0.4759i

Error:
   7.4718e-10

Iterations:
    20
```



Plot showing how the approximation of a root was changing with every iteration.

```
X:
   0.6829 - 0.4759i

Error:
   7.4718e-10

Iterations:
    20
```



Plot showing how the approximation of a root was changing with every iteration.

MM2: other version of Muller's method, using values of a polynomial and of its first and second order derivatives at one point only is often recommended. It is slightly more effective, due to difference in the calculations. We are not calculating values of the polynomial in three different points anymore, now we are calculating values of a polynomial and its first and second derivative in one point.

MM2 formulas:

$$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}.$$

Where:

$$\begin{aligned} y(0) &= c = f(x_k), \\ y'(0) &= b = f'(x_k), \\ y''(0) &= 2a = f''(x_k), \end{aligned}$$

And x is assigned after each iteration with a new approximation:

$$x_{k+1} = x_k + z_{min},$$

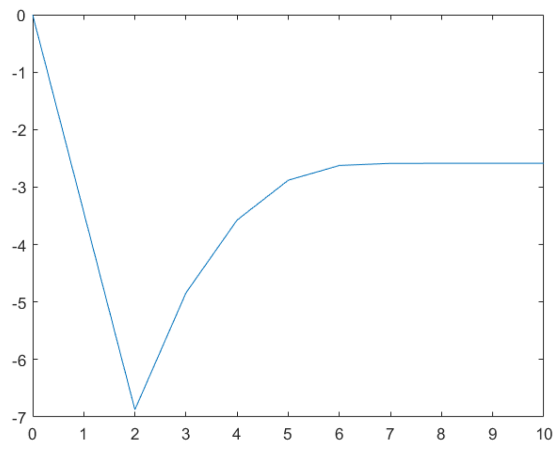
```
MM2(x1, f, df, ddf); % showing real root
MM2(x2, f, df, ddf); % showing real root
MM2(x3, f, df, ddf); % showing complex root
MM2_(x3, f, df, ddf); % showing complex root
```

Where:

```
x1 = -10.0;
x2 = -1.5;
x3 = 9.0;
```

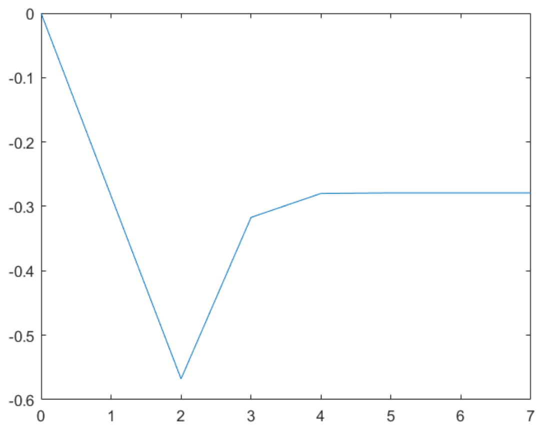
Outputs:

```
** MM2 **
X:
   -2.5869

Error:
   3.0198e-14

Iterations:
    10
```



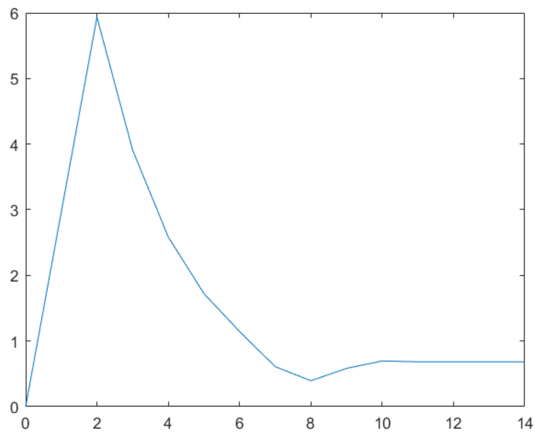
Plot showing how the approximation of a root was changing with every iteration.

```
** MM2 **
X:
   -0.2790

Error:
   1.3090e-13

Iterations:
     7
```



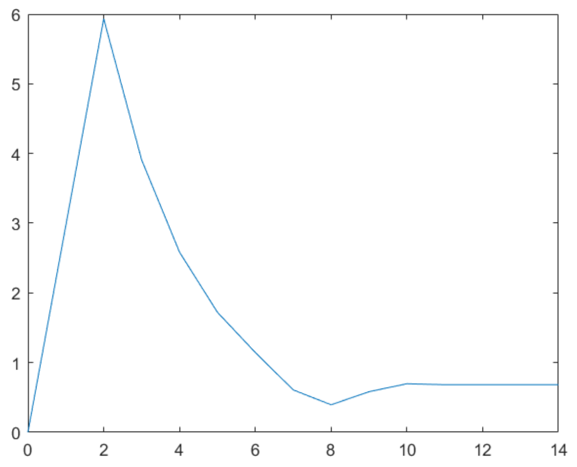
Plot showing how the approximation of a root was changing with every iteration.

```
** MM2 **
X:
   0.6829 + 0.4759i

Error:
   8.8818e-16

Iterations:
    14
```



Plot showing how the approximation of a root was changing with every iteration.

```
** MM2 **
X:
   0.6829 - 0.4759i

Error:
   8.8818e-16

Iterations:
    14
```



Plot showing how the approximation of a root was changing with every iteration.

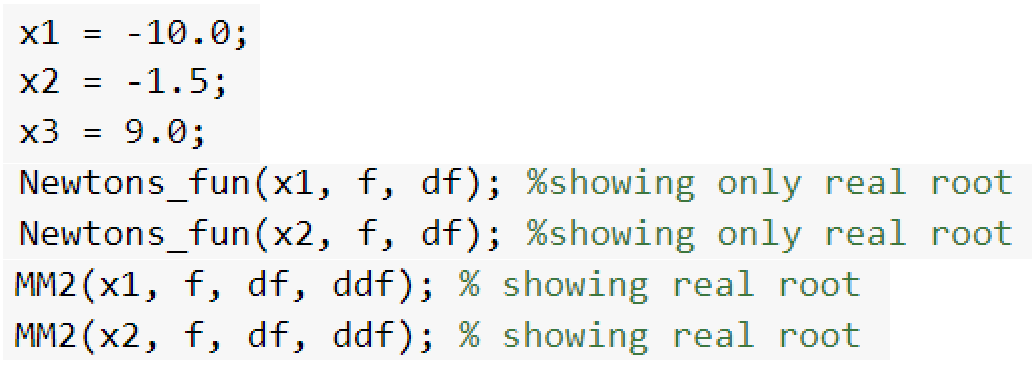
	MM 1				MM 2				
	Real 1	Real2	Complex1	Complex 2	Real 1	Real2	Complex 1	Complex 2	
1.	0	0	0+0i	0+0i	0	0	0+0i	0+0i	
2.	-3.0225	-0.0000	0.8043+0.0344i	0.8043 - 0.0344i	-6.8728	-0.5677	5.9332	5.9332	
3.	-2.5488	-0.2743	0.5914 + 0.3918i	0.5914 - 0.3918i	-4.8390	-0.3171	3.9101	3.9101	
4.	-2.5894	-0.2790	0.6489 + 0.4978i	0.6489 - 0.4978i	-3.5725	-0.2798	2.5853	2.5853	
5.	-2.5868	-0.2790	0.6878 + 0.4875i	0.6878 - 0.4875i	-2.8806	-0.2790	1.7225	1.7225	
6.	-2.5869	-0.2790	0.6867 + 0.4751i	0.6867 - 0.4751i	-2.6238	-0.2790	1.1451	1.1451	
7.	-2.5869	-0.2790	0.6829 + 0.4747i	0.6829 - 0.4747i	-2.5876	-0.2790	0.6087	0.6087	
8.	-2.5869		0.6826 + 0.4758i	0.6826 - 0.4758i	-2.5869		0.3951 + 0.6992i	0.3952 - 0.6992i	

9.	-2.5869		0.6829 + 0.4760i	0.6829 - 0.4760i	-2.5869		0.5832 + 0.4205i	0.5832 - 0.4205i	
10.	-2.5869		0.6830 + 0.4759i	0.6830 - 0.4759i	-2.5869		0.6968 + 0.4761i	0.6829 - 0.4761i	
11.	-2.5869		0.6830 + 0.4759i	0.6830 - 0.4759i			0.6831 + 0.4757i	0.6829 - 0.4757i	
12.			0.6829 + 0.4759i	0.6829 - 0.4759i			0.6829 + 0.4759i	0.6829 - 0.4759i	
13.			0.6829 + 0.4759i	0.6829 - 0.4759i			0.6829 + 0.4759i	0.6829 - 0.4759i	
14.			0.6829 + 0.4759i	0.6829 - 0.4759i			0.6829 + 0.4759i	0.6829 - 0.4759i	
15.			0.6829 + 0.4759i	0.6829 - 0.4759i					
16.			0.6829 + 0.4759i	0.6829 - 0.4759i					
17.			0.6829 + 0.4759i	0.6829 - 0.4759i					
18.			0.6829 + 0.4759i	0.6829 - 0.4759i					
19.			0.6829 + 0.4759i	0.6829 - 0.4759i					
20			0.6829 + 0.4759i	0.6829 - 0.4759i					

We can see that MM2 is indeed faster than MM1.
Error for MM1
Real 1 = 2.1095e-10
Real 2 = 3.6716e-10
Complex's = 7.4718e-10

Error for MM2
Real 1 = 3.0198e-14
Real 2 = 1.3090e-13
Complex's = 8.8818e-16

And also more accurate.



```
x1 = -10.0;
x2 = -1.5;
x3 = 9.0;
Newtons_fun(x1, f, df); %showing only real root
Newtons_fun(x2, f, df); %showing only real root
MM2(x1, f, df, ddf); % showing real root
MM2(x2, f, df, ddf); % showing real root
```

	Newton			MM2					
	X1	X2		X1	X2				
1	-7.6449	-0.2073		0	0				
2	-5.8992	-0.2865		-3.0225	-4.9749e-05				
3	-4.6206	-0.2790		-2.5488	-0.2743				
4	-3.7083	-0.2790		-2.5894	-0.2790				
5	-3.0962	-0.2790		-2.5868	-0.2790				
6	-2.7440	-0.2790		-2.5869	-0.2790				
7	-2.6079			-2.5869	-0.2790				

8	-2.5874			-2.5869					
9	-2.5869			-2.5869					
10	-2.5869			-2.5869					
11	-2.5869								

Surprisingly for given initial x1, MM2 was faster than Newton's method, however we can see that for x2 MM2 is slightly slower, as it should be.

Error for Newton
Real 1 = 4.0856e-14
Real 2 = 5.5511e-17

Error for MM2
Real 1 = 3.0198e-14
Real 2 = 1.3090e-13

Also, for given stop test for x1 MM2 is more accurate, however for x2 Newton's method is.

Task no.3

In task number 3 we had to find all roots (real and complex) of a polynomial given in task number 2. However we had to use Laguerre's method, then compare it to the MM2 results.

Laguerre's method: this method is defined with a formula:

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)(f'(x_k))^2 - nf(x_k)f''(x_k)]}},$$

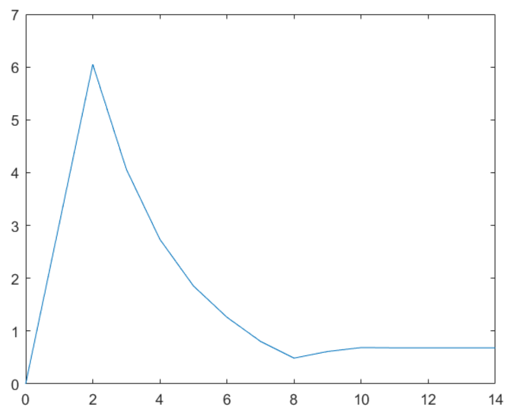
Where "n" denotes the order of the polynomial and the sign in the denominator is chosen in a way assuring a larger absolute value of the denominator, as in MM2. The formula for Laguerre's method is similar to one from MM2, however there are some little differences, like in Laguerre's we are taking the order of the polynomial under consideration. Theoretically Laguerre's method should be better, if not the best out of all presented in our project. Laguerre's method is globally convergent.

```
x1 = -10.0;
x2 = -1.5;
x3 = 9.0;
```

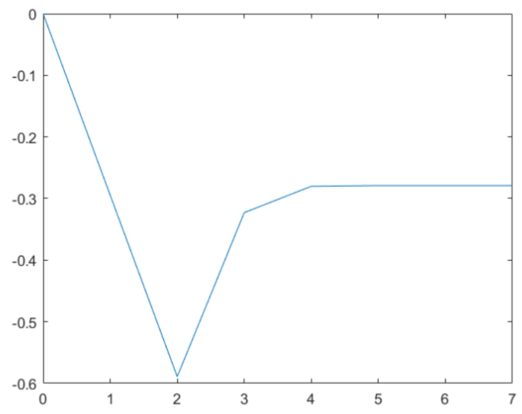
```
Laguerre(x1, f, df, ddf); % showing real root
Laguerre(x2, f, df, ddf); % showing real root
Laguerre(x3, f, df, ddf); % showing complex root
Laguerre_(x3, f, df, ddf); % showing complex root
```

Output:

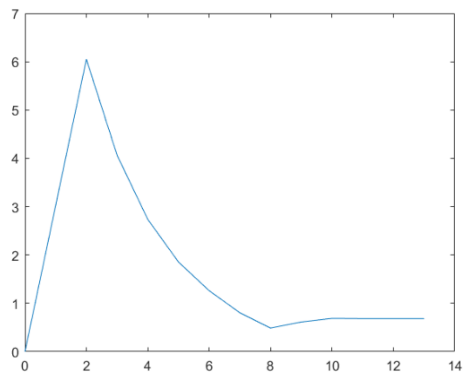
```
** Laguerre **
X:
   0.6829 - 0.4759i

Error:
   1.2413e-16

Iterations:
    14

Plot Iterations/Solution
```



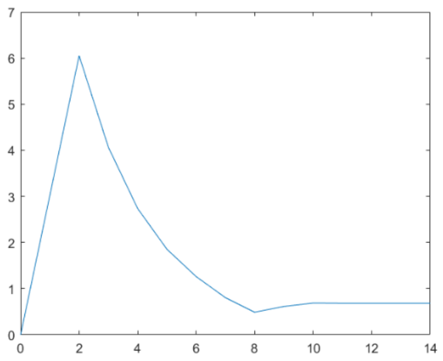
```
** Laguerre **
X:
   -0.2790

Error:
   4.0506e-13

Iterations:
     7

Plot Iterations/Solution
```



```
** Laguerre **
X:
   0.6829 + 0.4759i

Error:
   6.5189e-10

Iterations:
    13

Plot Iterations/Solution
```



```
** Laguerre **
X:
   0.6829 - 0.4759i

Error:
   1.2413e-16

Iterations:
    14

Plot Iterations/Solution
```



	MM2					Laguerre			
	X1	X2	X3	X3		X1	X2	X3	X3
1	0	0	0	0		0	0	0	0
2	- 6.8728	- 0.5677	5.9332	5.9332		- 6.9904	- 0.5890	6.0502	6.0502
3	- 4.8390	- 0.3171	3.9101	3.9101		- 4.9887	- 0.3229	4.0645	4.0645
4	- 3.5725	- 0.2798	2.5853	2.5853		- 3.7034	- 0.2801	2.7367	2.7367
5	- 2.8806	- 0.2790	1.7225	1.7225		- 2.9595	- 0.2790	1.8544	1.8544
6	- 2.6238	- 0.2790	1.1451	1.1451		- 2.6457	- 0.2790	1.2610	1.2610
7	- 2.5876	- 0.2790	0.6087	0.6087		- 2.5887	- 0.2790	0.8046	0.8046

8	- 2.5869		0.3951 + 0.6992i	0.3951 - 0.6992i		- 2.5869		0.4877 + 0.6848i	0.4877 - 0.6848i
9	- 2.5869		0.5832 + 0.4205i	0.5832 - 0.4205i		- 2.5869		0.6137 + 0.4527i	0.6137 - 0.4527i
10	- 2.5869		0.6968 + 0.4761i	0.6968 - 0.4761i		- 2.5869		0.6880 + 0.4742i	0.6880 - 0.4742i
11			0.6831 + 0.4757i	0.6831 - 0.4757i				0.6829 + 0.4758i	0.6829 - 0.4758i
12			0.6829 + 0.4759i	0.6829 - 0.4759i				0.6829 + 0.4759i	0.6829 - 0.4759i
13			0.6829 + 0.4759i	0.6829 - 0.4759i				0.6829 + 0.4759i	0.6829 - 0.4759i
14			0.6829 + 0.4759i	0.6829 - 0.4759i					0.6829 - 0.4759i

Error for Laguerre
Real 1 = 1.2279e-12
Real 2 = 4.0506e-13
Complex 1 = 6.5189e-10
Complex 2 = 1.2313e-16

Error for MM2
Real 1 = 3.0198e-14
Real 2 = 1.3090e-13
Complex's = 8.8818e-16

All things concerned it is hard to say, that Laguerre method is better then MM2, for given initial points. Laguerre's iterated only for one initial point faster then MM2 and, what is a surprise for me, Laguerre's has got bigger error. For both cases the error value was calculated in an exact same way:

```
e = abs(x - x0);
```
for every iteration.

Bibliography:

"Numerical Methods", Piotr Tatjewski;

CODES:

```
% Task 1a) bisection method
% Given interval is [2,11]
% and given f-ction is : f(x) = x*cos(x)-log(x+2);
clear;


y = @(x) x*cos(x)-log(x+2);
fplot(@(x) x*cos(x)-log(x+2), [2 11])
a = 6.5;
b = 11;
i = 1;
c = (a + b)/2;
i_vec = zeros(i);
x_vec = zeros(i);
%eps


while abs(y(c)) > 10e-10
```

```
    if (y(c) * y(b)) < 0
        a = c;
    elseif (y(c) * y(a)) < 0
        b = c;

    end

    c = (a+b)/2;
    i_vec(i) = i;
    x_vec(i) = c;
    i = i+1;
end


disp("Root between 6.5 and 11 is: ");
disp(c);
disp("Number of iterations: ");
disp(i);
disp("Error: ");
disp(abs(y(c)));
plot(i_vec, x_vec)
x_vec


y2 = @(x) x*cos(x)-log(x+2);


a = 2;
b = 6.5;
c = (a + b)/2;
i = 1;
i_vec = zeros(i);
x_vec = zeros(i);


%eps

while abs(y2(c)) > 10e-10

    if (y2(c) * y2(b)) < 0
        a = c;
    else
        b = c;
    end

    c = (a+b)/2;
    i_vec(i) = i;
    x_vec(i) = c;
    i = i+1;
end


disp("Root between 2 and 6.5 is: ");
disp(c);
disp("Number of iterations: ");
disp(i)
disp("Error: ");
disp(abs(y(c)));
x_vec
plot(i_vec, x_vec)





% Task 1b) Newton's method
% Given interval is [2,11]
% and given f-ction is : f(x) = x*cos(x)-log(x+2);
clear
syms x;


f = @(x) x*cos(x)-log(x+2);
int = [2 6.5];
int2 = [6.5 11];
fzero(f,int)
fzero(f,int2)
ezplot('x*cos(x)-log(x+2)',[2 11])


df = @(x) cos(x) - (1/(x + 2)) - (x*sin(x));


x = 4;


Newtons_fun(x, f, df);




x2 = 9;


Newtons_fun(x2, f, df);
function [x_vec n_vec] = Newtons_fun (x0, f, df)




n = 0;
nmax = 30;
eps;
e = 1;
x_vec = zeros(n);
n_vec = zeros(n);
e_vec = zeros(n);
```

```
while e >= 10e-10 && n <= nmax

      x = x0 - (f(x0)/df(x0));
      e = abs(x-x0);
      x0 = x;
      n=n+1;
      x_vec(n) = x;
      n_vec(n) = n;
      e_vec(n) = e;
end
disp("**NEWTON's METHOD**");
disp("X: ");
disp(x);
disp("Error: ");
disp(e);
disp("Iterations: ");
disp(n);
disp("Plot Iterations/Solution");


plot(n_vec, x_vec)
x_vec


%disp("Plot Iterations/Error");
%plot(n_vec, e_vec);


end

% Find all real and complex roots of the polynomial
% f(x) = 2x^4 + 3x^3 - 5x^2 + 2x + 1
% [a4 a3 a2 a1 a0] = [2 3 -5 2 1]
% MM1 and MM2 / Newton's method and compare to MM2
%                  (the same initial points)


clear
syms x;
p = [2 3 -5 2 1];
r = roots(p)
scatter(real(r),imag(r),'filled','blue')
ezplot('(2*(x^4))+(3*(x^3))-(5*(x^2))+(2*x)+1',[-3 2])
f = @(x) 2*x^4 + 3*x^3 - 5*x^2 + 2*x + 1;
df = @(x) 8*x^3 + 9*x^2 - 10*x + 2;
%g = diff(df(x))
ddf = @(x) 24*x^2 + 18*x - 10;


x1 = -10.0;
x2 = -1.5;
x3 = 9.0;
disp("*****************MM1******************");
MM1(-2.9, -3.1, -40, f);% showing real root
MM1(0, -1, 100, f);  % showing real root
```

```
MM1(0.7, 0.8, 10+10i, f); % showing complex root
MM1(0.7, 0.8, 10-10i, f); % showing complex root
disp("***************MM2*****************");
MM2(x1, f, df, ddf); % showing real root
MM2(x2, f, df, ddf); % showing real root
MM2(x3, f, df, ddf); % showing complex root
MM2_(x3, f, df, ddf); % showing complex root
disp("***************NEW*****************");
Newtons_fun(x1, f, df); %showing only real root
Newtons_fun(x2, f, df); %showing only real root
disp("Newton for x3 value / prove that ");
 disp("it cannot calc complex root: ");
Newtons_fun(x3, f, df); %Trying to get complex
disp("***************LAG*****************");
% Task 3
Laguerre(x1, f, df, ddf); % showing real root
Laguerre(x2, f, df, ddf); % showing real root
Laguerre(x3, f, df, ddf); % showing complex root
Laguerre_(x3, f, df, ddf); % showing complex root
disp("**************************************");


function [x n] = MM1(x0, x1, x2, f)


    e = 1;
    n = 1;
    nmax = 100;
    x_vec = zeros(n);
    n_vec = zeros(n);
    e_vec = zeros(n);
    while e >= 10e-10 && n <= nmax

        z0 = x0 - x2;
        z1 = x1 - x2;
        c = f(x2);
        %a*z0 + ((f(x1) - c)/z1) - a*z1 == (f(x0) - c)/z0;
        a = (((f(x0) - c)/z0) - ((f(x1) - c)/z1))/(z0 - z1);
        b = ((f(x1) - c)/z1) - a*z1;

        sqr_plus = b + sqrt((b^2) - (4*a*c));
        sqr_min = b - sqrt((b^2) - (4*a*c));
        z = (-1*2*c)/(sqr_plus);
        z_ = (-1*2*c)/(sqr_min);


        if abs(sqr_plus) >= abs(sqr_min)

            z_min = z;
        else

            z_min = z_;
        end
        %disp("uhjgjhgjhjh");
        x = x2 + z_min;
        e = abs(x - x2);
        n = n+1;
        x2 = x;
        x_vec(n) = x;
        n_vec(n) = n;
        e_vec(n) = e;

    end


    %disp("** MM1 **");
    disp("X: ");
    disp(x2);
    disp("Error: ");
    disp(e);
    disp("Iterations: ");
    disp(n);
%disp("Plot Iterations/Solution");
x_vec
plot(n_vec, x_vec)


%disp("Plot Iterations/Error");
%plot(n_vec, e_vec);
end

function [x n] = MM2(x0, f, df, ddf)


    e = 1;
    n = 1;
    %eps;
    nmax = 20;
    x_vec = zeros(n);
    n_vec = zeros(n);
    e_vec = zeros(n);
    while e >= 10e-10 && n <= nmax
    % for i = 1:10
         a = ddf(x0)/2;
         b = df(x0);
         c = f(x0);

        denom = b+sqrt((b^2)-2*a*c);
        denom_ = b-sqrt((b^2)-2*a*c);
        z = (-1*2*c)/denom;
        z_ = (-1*2*c)/denom_;
        abs(z);
        abs(z_);

    if (abs(z) > abs(z_))

        x = x0 + z_;
        e = abs(x - x0);
        x0 = x;
    elseif (abs(z_) > abs(z))
```

```
        x = x0 + z;
        e = abs(x - x0);
        x0 = x;
    else

        x = x0 + z;
        e = abs(x - x0);
        x0 = x;

    end
    % end

         n = n+1;
         x_vec(n) = x;
         n_vec(n) = n;
         e_vec(n) = e;
    end

        disp("** MM2 **");
        disp("X: ");
        disp(x);
        disp("Error: ");
        disp(e);
        disp("Iterations: ");
        disp(n);
%disp("Plot Iterations/Solution");
x_vec
plot(n_vec, x_vec)
%disp("Plot Iterations/Error");
%plot(n_vec, e_vec)


end
function [x n] = MM2_(x0, f, df, ddf)


    e = 1;
    n = 1;
    %eps;
    nmax = 20;
    x_vec = zeros(n);
    n_vec = zeros(n);
    e_vec = zeros(n);
    while e >= 10e-10 && n <= nmax

         a = ddf(x0)/2;
         b = df(x0);
         c = f(x0);

        denom = b+sqrt((b^2)-2*a*c);
        denom_ = b-sqrt((b^2)-2*a*c);
        z = (-1*2*c)/denom;
        z_ = (-1*2*c)/denom_;
        abs(z);
        abs(z_);

    if (abs(z) > abs(z_))

        x = x0 + z_;
        e = abs(x - x0);
        x0 = x;
    elseif (abs(z_) > abs(z))

        x = x0 + z;
        e = abs(x - x0);
        x0 = x;
    else

        x = x0 + z_;
        e = abs(x - x0);
        x0 = x;

    end


        n = n+1;
        x_vec(n) = x;
        n_vec(n) = n;
        e_vec(n) = e;
    end

    disp("** MM2 **");
    disp("X: ");
    disp(x);
    disp("Error: ");
    disp(e);
    disp("Iterations: ");
    disp(n);
%disp("Plot Iterations/Solution");
x_vec
plot(n_vec, x_vec)
%disp("Plot Iterations/Error");
%plot(n_vec, e_vec)


end

function [x n] = Laguerre (x0, f, df, ddf)




    e = 1;
    n = 1;
    %eps;
    nmax = 20;
    x_vec = zeros(n);
    n_vec = zeros(n);
    e_vec = zeros(n);
    while e >= 10e-10 && n <= nmax

         a = ddf(x0)/2;
         b = df(x0);
         c = f(x0);

        denom = b+sqrt(3*(3*(b^2)-4*a*c));
        denom_ = b-sqrt(3*(3*(b^2)-4*a*c));
        z = (4*c)/denom;
        z_ = (4*c)/denom_;
        abs(z);
        abs(z_);

    if (abs(z) > abs(z_))

        x = x0 - z_;
        e = abs(x - x0);
        x0 = x;
    elseif (abs(z_) > abs(z))

        x = x0 - z;
        e = abs(x - x0);
        x0 = x;
    else

        x = x0 - z;
        e = abs(x - x0);
        x0 = x;

    end


        n = n+1;
        x_vec(n) = x;
        n_vec(n) = n;
        e_vec(n) = e;
    end

    disp("** Laguerre **");
    disp("X: ");
    disp(x);
    disp("Error: ");
    disp(e);
    disp("Iterations: ");
    disp(n);
disp("Plot Iterations/Solution");
x_vec
plot(n_vec, x_vec)
%disp("Plot Iterations/Error");
%plot(n_vec, e_vec)
end

function [x] = Laguerre_ (x0, f, df, ddf)
```

```
e = 1;
n = 1;
%eps;
nmax = 20;
x_vec = zeros(n);
n_vec = zeros(n);
e_vec = zeros(n);
while e >= 1e-10 && n <= nmax

     a = ddf(x0)/2;
     b = df(x0);
     c = f(x0);

    denom = b+sqrt(3*(3*(b^2)-4*a*c));
    denom_ = b-sqrt(3*(3*(b^2)-4*a*c));
    z = (4*c)/denom;
    z_ = (4*c)/denom_;
    abs(z);
    abs(z_);

if (abs(z) > abs(z_))

    x = x0 - z_;
    e = abs(x - x0);
    x0 = x;
elseif (abs(z_) > abs(z))

    x = x0 - z;
    e = abs(x - x0);
    x0 = x;
else

    x = x0 - z_;
    e = abs(x - x0);
    x0 = x;

end


    n = n+1;
    x_vec(n) = x;
    n_vec(n) = n;
    e_vec(n) = e;
end

    disp("** Laguerre **");
    disp("X: ");
    disp(x);
    disp("Error: ");
    disp(e);
    disp("Iterations: ");
    disp(n);
disp("Plot Iterations/Solution");
x_vec
plot(n_vec, x_vec)
%disp("Plot Iterations/Error");
%plot(n_vec, e_vec)
end
```