Chmielewska Urszula

300167

Numerical Methods

Project B No. #14

Advisor: Dr Adam Krzemienowski

TASK 1.

The program which finds all zeros of the function in a given interval [2, 11]

$$f(x) = 0.5x * \cos(x) - \ln(x)$$

using:

a) the false position method
b) the Newton's method

Theoretical background

The false position method

It is mainly called the Regula Falsi method. Let's choose an initial interval $[a_0, b_0]$ containing a root of the function. In every iterations there are two operations:

1) Division of the current interval $[a_n, b_n]$ into two subintervals, with the point of intersection c_n:

$$c_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)} = \frac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$$

2) Then the next interval is chosen as the subinterval corresponding to the product with the negative value.

Convergence

The false position method is always convergent, since it always chooses and shortens the interval containing the root. The convergence is linear (p = 1). This provides that the function is continuous and differentiable. However there is one weak point of this method. The convergence may be very slow, due to the fact that a situation, when one point from the interval does not change in any iteration, may occur. Basically it means that iterations do not lead to shortening the intervals to zero.

To avoid the slow convergence, there is a modified standard formula (presented above) which improves the convergence:

$$c_n = \frac{a_n \frac{f(b_n)}{2} - b_n f(a_n)}{\frac{f(b_n)}{2} - f(a_n)}$$

The remedy is to take a smaller value of the function at the "frozen" end of the subsequent intervals. The above formula is for the case with frozen interval limit, for at least two iterations, from the right end. For the frozen left end, the following formula should be used:

$$c_n = \frac{a_n f(b_n) - b_n f\frac{(a_n)}{2}}{f(b_n) - \frac{f(a_n)}{2}}$$

The presented false position method with the modifications is called the modified Regula Falsi algorithm. It is super linearly convergent and the length of the subsequent intervals converges to zero.

Newton's method

The Newton's method uses approximations of the function f(x) by the first order part of its expansion into a Tylor series at a current point x_n (where x_n denoted a current approximation of the root)

$$f(x) \cong f(x_n) + f'(x_n)(x - x_n)$$

The next approximated point x_{n+1} is obtained as a result of:

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

The above equations leads us to the formula:

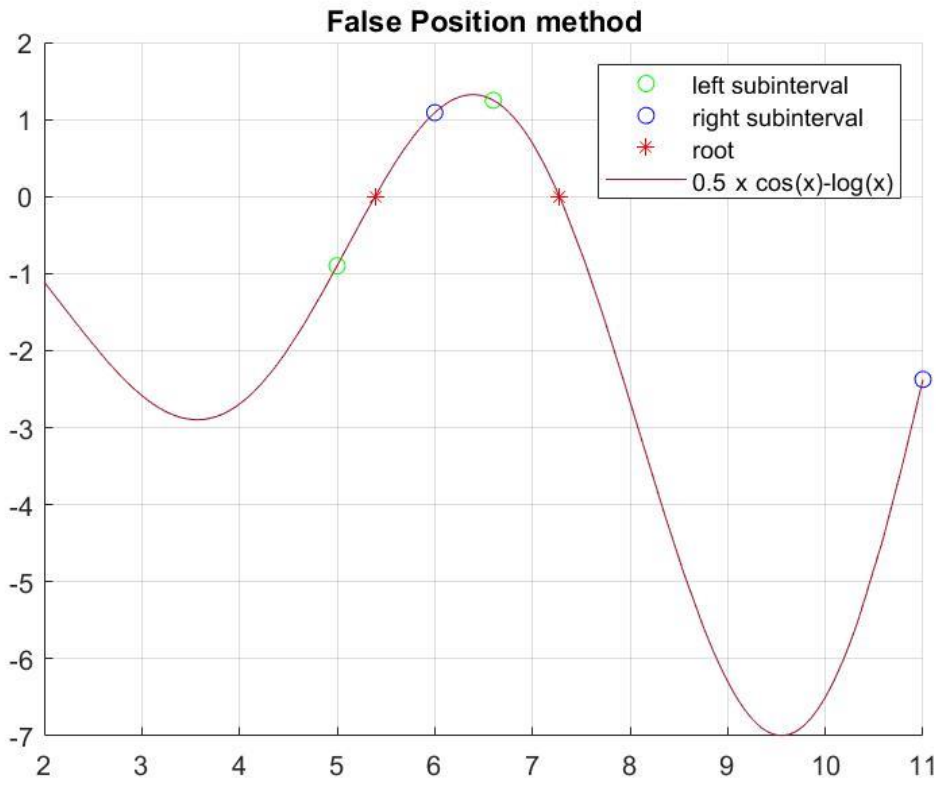
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

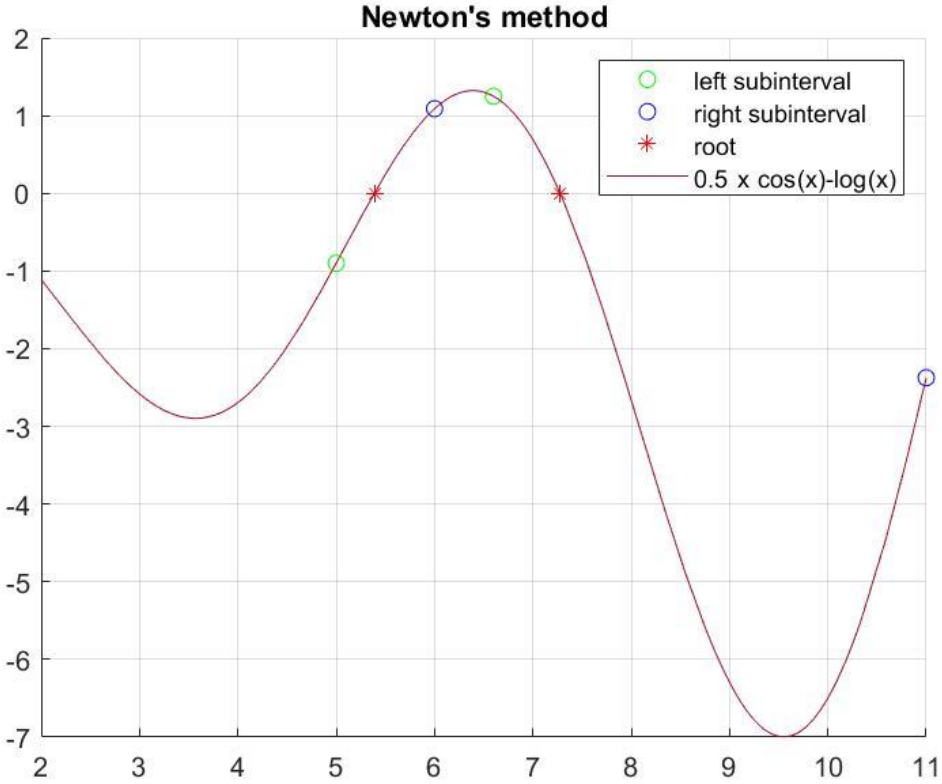
Convergence

Newton's method is locally convergent. If this method is convergent, then it is usually very fast, since the convergence is quadratic (p = 2). It may happen that the method fails because of the divergence. It happens when an initial point is outside the root's set of attraction.

The best effectiveness of the Newton's method is obtained when the function derivative at the root is sufficiently far from zero. Then the slope of the function is steep. However it may diverge if the derivative is close to zero, because it is sensitive to numerical errors when it is close to the root.

GRAPHICAL RESULTS





RESULTS IN NUMBERS

False position method

	Iteration	Argument value	Function value
First root	1	5.4526	0.1427
	2	5.3907	0.0066
	3	5.3879	2.6426e-04
	4	5.3877	1.0516e-05
	5	5.3877	4.1836e-07
	6	5.3877	1.6643e-08
	7	5.3877	6.6211e-10
Second root	1	8.1168	-3.1484
	2	7.0308	0.6277
	3	7.4661	-0.5986
	4	7.1855	0.2547
	36	7.2770	7.6309e-09

Newton's method

	Iteration	Argument value	Function value
First root	1	5.3845	-0.0073
	2	5.3877	-4.6102e-06
	3	5.3877	-1.8592e-12
Second root	1	6.9429	0.8052
	2	7.3720	-0.2891
	3	7.2808	-0.0110
	4	7.2770	-1.9899e-05
	5	7.2770	-6.5362e-11

COMPARISON

Both methods managed to find the same roots values for the taken precision 10^{-9}

Iterations needed for:	False position	Newton's
First root	7	3
Second root	36	5

Conclusions

By the first look at the obtained results we can conclude that the Newton's method is much faster than the False Position method. The second root we obtained in only 5 steps by Newton's method, while False Position took us 36 iterations. Finally both methods obtained the same results and wanted precision, however it Newton was more efficient.

To choose initial values for the intervals, firstly it was necessary to look at the plot of the graph. There was a problem with Newton's method with choosing a proper initial points. Method failed with finding root, when the slope of the function in a given interval, was too steep. From this we can conclude that the proper choosing of the initial points is very important.

To improve the False Position method to be faster, there are two options. I obtained a significant improvement if I took an initial point with the value closer to the root. For example 36 iterations were needed for the initial point 6.6 and if we change the initial point to 7, then we need 11 iterations. Another improvement was possible by modifying formulas for c_n calculations on the "frozen" subintervals as it was presented in the theoretical introduction. However the task was to find roots using the false position method and with modifications it is called modified regula falsi algorithm. That is the reason that this possible improvement was described only in the theoretical part.

Finally, the Newton's method is fast however it can easily diverge when the subinterval of the function is too steep. False position method is very slow, however it is always convergent because it always chooses and shortens the interval containing the root.

TASK 2.

Find all (real and complex) roots of the polynomial

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0,$$

for given a's

$$[a_4\ a_3\ a_2\ a_1\ a_0\ ] = [2\ 5 - 2\ 3\ 7]$$

Using Muller's method implementing both the MM1 and MM2 versions. Compare the results. Find also real roots using the Newton's method and compare the results with the MM2 version of the Müller's method (using the same initial points).

Theoretical background

Muller's method

Muller's method was developed specially for finding roots of a quadratic equations. Real and complex. It is developed using a quadratic interpolation based on three different points.

MM1

We consider points x0, x1 and x2 with its corresponding polynomial values. For the mentioned points, we need to construct a quadratic function, that passes through these point. Then we find roots and select one of them as a next approximation of the solution.

Let us introduce a new variable:

$$z = x - x_2$$

$$z_0 = x_0 - x_2$$

$$z_1 = x_1 - x_2$$

Then the interpolating parabola is defined as a quadratic equation using the variable z and from the three given points we formulate the following system of equations:

$$az_0^2 + bz_0 + c = y(z_0) = f(x_0)$$

$$az_1^2 + bz_1 + c = y(z_1) = f(x_1)$$

$$c = y(0) = f(x_2)$$

By solving the above equations we obtain the roots:

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

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

For the next iteration, root which has a smaller absolute value, is chosen:

$$x_3 = x_2 - z_{min}$$

MM2

This method is slightly more effective numerically, since the method uses the values of a polynomial and its first and second derivatives at one point, and it is numerically less expensive.

From the quadratic parabola equation definition, we have that $z = x - x_k$

And for the point z = 0, we obtain:

$$y(0) = c = f(x_k)$$

$$y'(0) = b = f'(x_k)$$

$$y''(0) = 2a = f''(x_k)$$

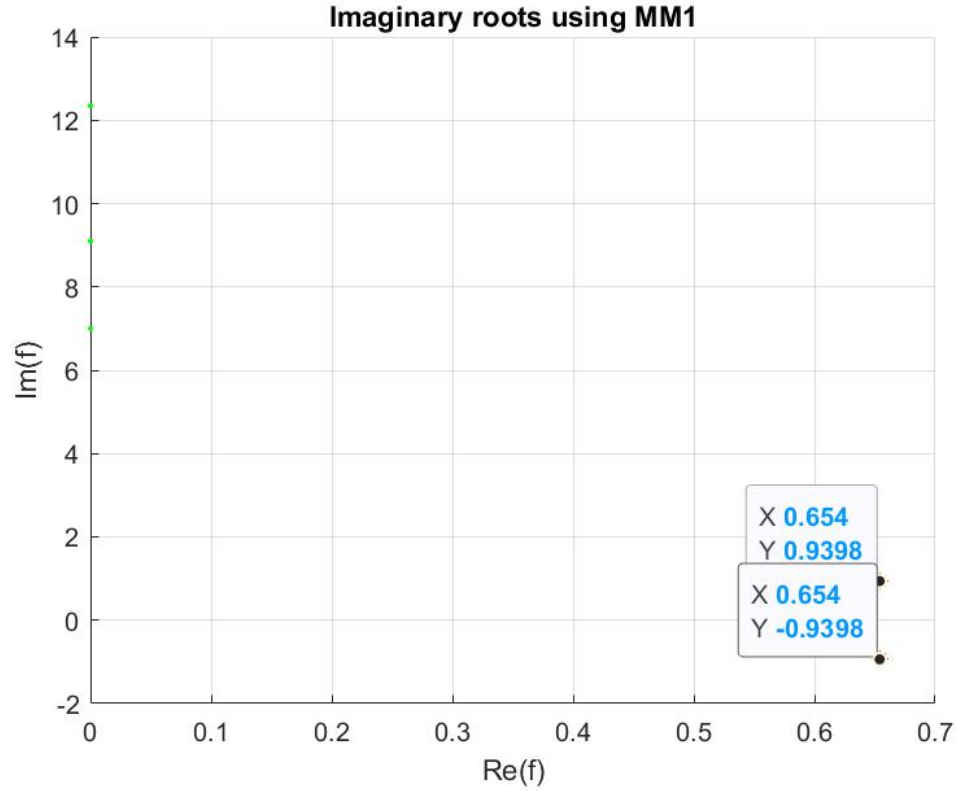
What leads us to the formula for finding the roots:

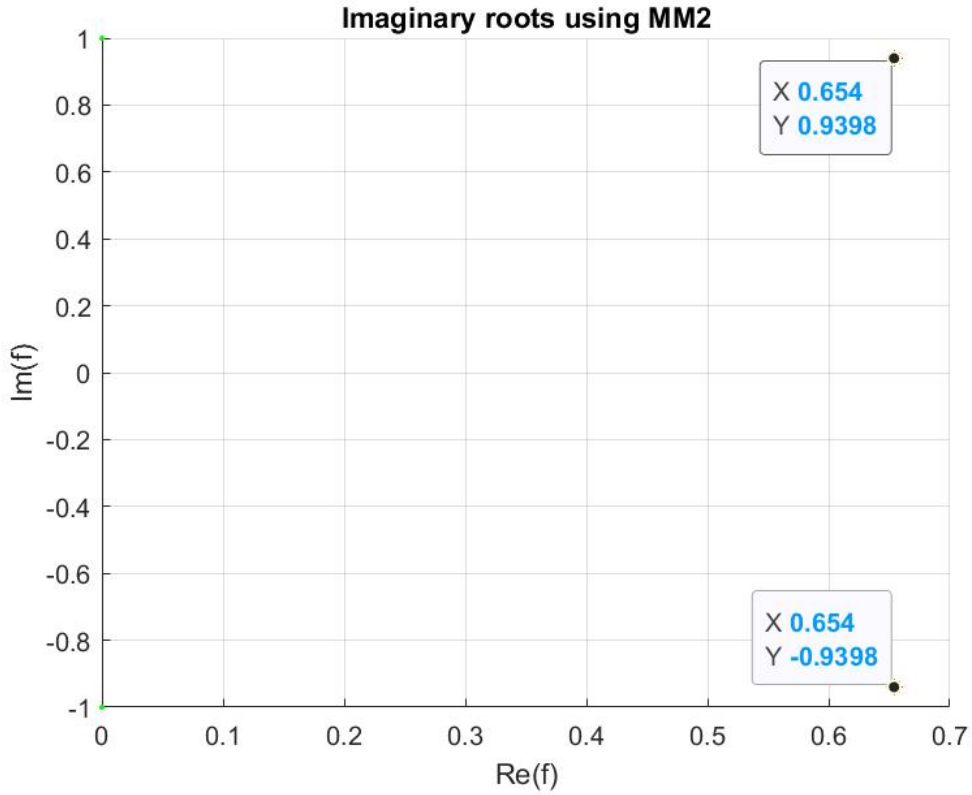
$$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) +/- \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}$$

For the approximate solution we choose a root with a smaller absolute value:

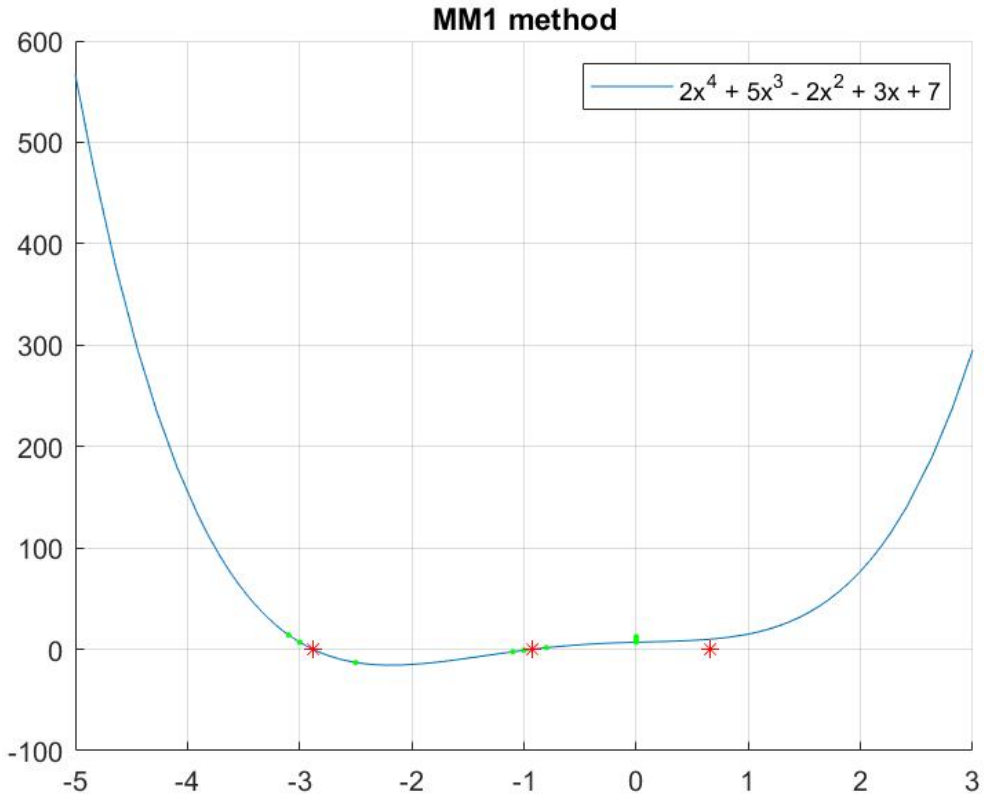
$$x_{k+1} = x_k + z_{min}$$

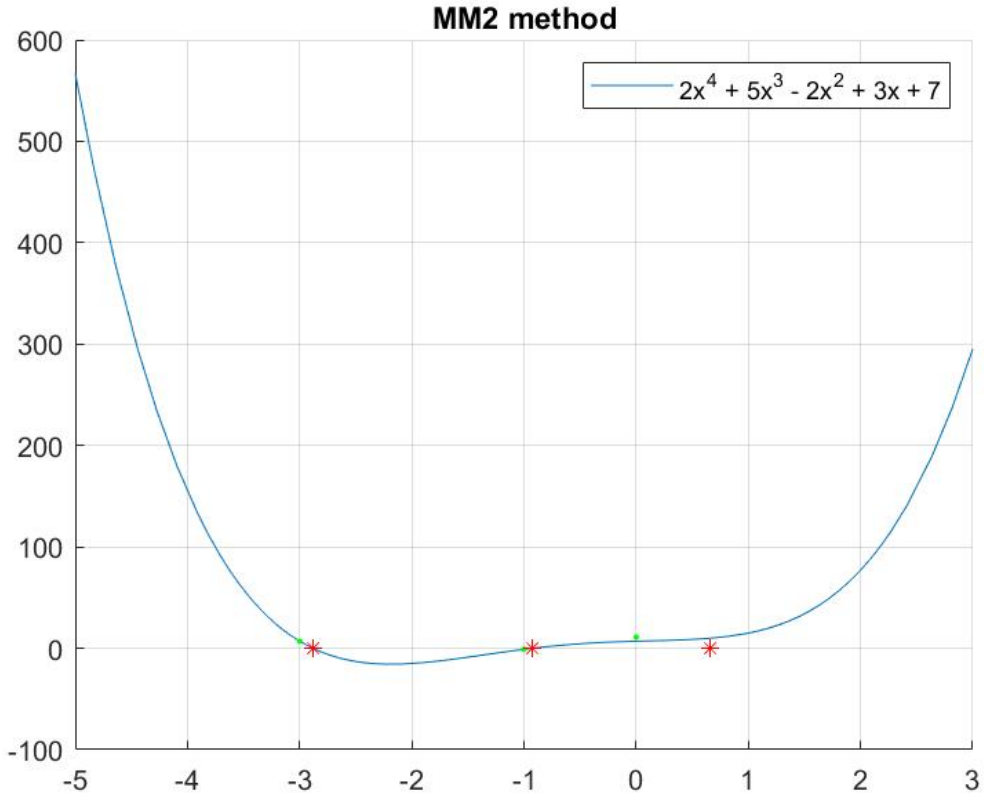
GRAPHICAL RESULTS OF IMAGINARY ROOTS





GRAPHICAL RESULTS OF ROOTS





RESULTS IN NUMBERS

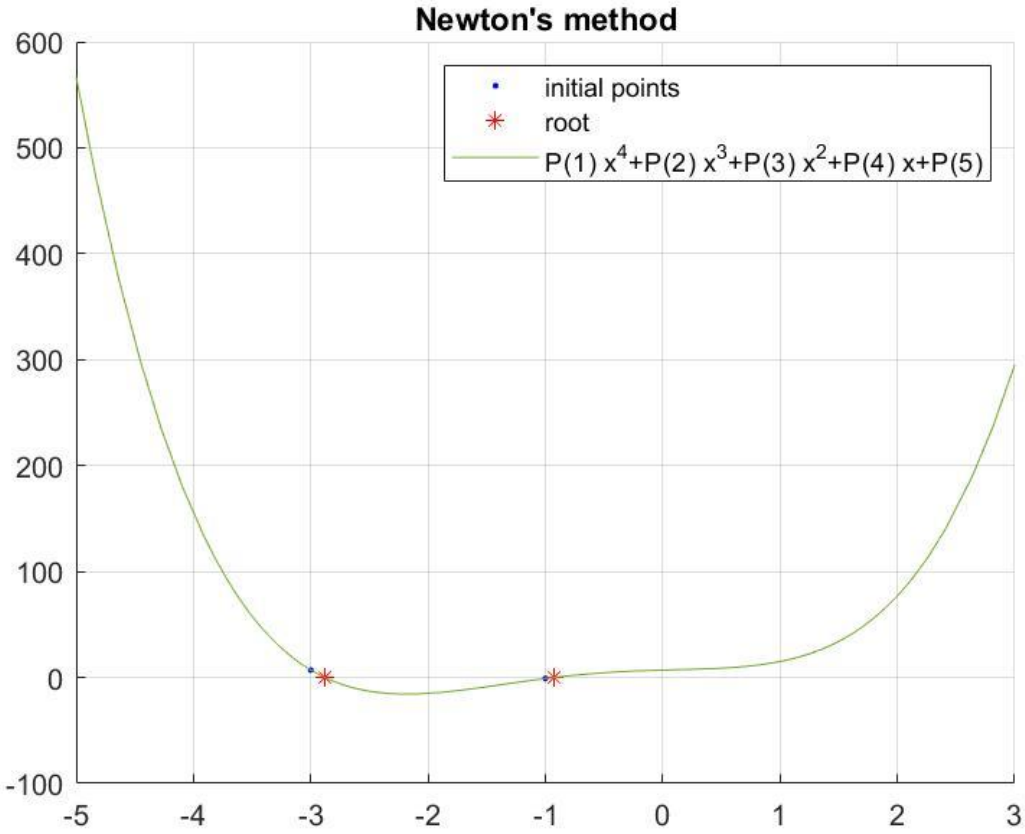
MM1

	Iteration	Argument value	Function value
First imaginary root	1	0.7236 + 0.7214i	3.2300 + 3.8786i
	2	0.7000 + 0.9428i	-0.8881 + 0.8221i
	7	0.6540 + 0.9398i	-1.7764e-15 - 1.7764e-15i
Second imaginary root	1	0.7236 - 0.7214i	3.2300 - 3.8786i
	2	0.7000 - 0.9428i	-0.8881 - 0.8221i
	7	0.6540 - 0.9398i	-1.7764e-15 + 1.7764e-15i
First real root	1	-2.8847	0.1716
	2	-2.8814	-0.0016
	5	-2.8814	-1.5987e-14
Second real root	1	-0.9262	0.0043
	2	-0.9266	-6.9686e-06
	4	-0.9266	3.2157e-11

MM2

	Iteration	Argument value	Function value
First imaginary root	1	0.5965 + 0.8560i	2.2466 + 0.4175i
	2	0.6542 + 0.9403i	-0.0120 - 0.0049i
	4	0.6540 + 0.9398i	2.6645e-15 + 4.4409e-16i
Second imaginary root	1	0.5965 - 0.8560i	-0.0120 + 0.0049i
	2	0.6542 - 0.9403i	7.1753 + 0.2343i
	4	0.6540 - 0.9398i	2.6645e-15 - 4.4409e-16i
First real root	1	-0.9266	-0.0011
	2	-0.9266	-1.4913e-12
	3	-0.9266	-8.8818e-16
Second real root	1	-2.8808	-0.0318
	2	-2.8814	4.0477e-09
	3	-2.8814	1.7764e-15

GRAPHICAL RESULTS OF REAL ROOTS - NEWTON'S METHOD



RESULTS IN NUMBERS

	Iterations	Argument value	Function value
First real root	1	-0.9286	-0.0266
	2	-0.9266	-2.2458e-05
	3	-0.9266	-1.6149e-11
Second real root	1	-2.8939	0.6638
	2	-2.8816	0.0084
	4	-2.8814	2.1316e-14

COMPARISON

MM1 WITH MM2

Using all of the methods the obtained results in numbers were the same. However the number of iterations needed was different. Used precision was also the same and equal to 10^{-9}.

Iterations needed for:	MM1	MM2
First real root	5	3
Second real root	4	3
First imaginary root	7	4
Second Imaginary root	7	4

MM2 WITH NEWTON'S METHOD

Both methods managed to obtain the same results with the same used precision 10^{-9}.

Iterations needed for:	Newton's method	MM2
First real root	3	3
Second real root	4	3

Conclusions

Both Muller's methods managed to find all real and complex roots. There is a difference in the number of needed iterations in MM1 and MM2 methods, however it is not as significant as it was in the first task between Newton's and False position methods. While comparing the number of iterations, MM2 method is faster than MM1. The tolerance precision value was taken the same for both methods, and MM2 method converged more precisely and quicker.

Modified version of Newton's method, with only one initial point, also managed to find root of a given polynomial. However this method works only in the case of finding the real roots. Newton's method for the same initial points as taken for the MM2, calculated roots in almost the same number of iterations. That was the reason of the proper setting the initial points for which the slope of the function was proper for the sensitivity of the Newton's method.

Finally, we can conclude that MM1 and MM2 methods are mor complex than the Newton's method, since by their usage, we are able to find complex roots. However the Newton's method has a higher progress per iteration because of quadratic order of the convergence and it is very fast. On the other hand, the MM2 method converges faster than the MM1 method and also it needs only one initial point which is easier to be set.

TASK 3.

Find all (real and complex) roots of the polynomial f(x) from the Task 2. using the Laguerre's method. Compare the results with the MM2 version of the Müller's method (using the same initial points).

Theoretical background

Laguerre's method

The Laguerre's method is regarded as one of the best methods for polynomial root finding. This method is defined by the formula:

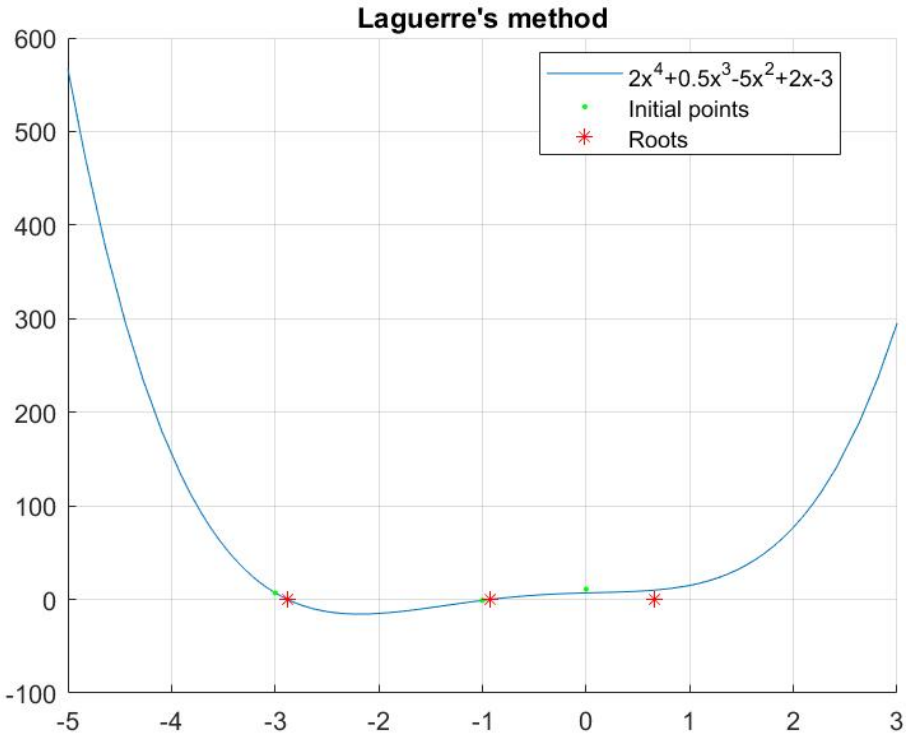
$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)\left(f'(x_k)\right)^2 - nf(x_k)f''(x_k)]}}$$

In this notation n denoted the order of the polynomial and the sign in the denominator is chosen in a way to obtain a larger absolute value of the denominator. Technically, Laguerre's method looks similar to the MM2 method with some improvements. We make a hypothesis that this method will be the best out of all the method's analyzed in this project.

Convergence

Laguerre's method is globally convergent when starting from any real initial point. Divergence may occur while finding complex roots, however it is very rare.

GRAPHICAL RESULTS OF ROOTS



ROOTS RESULTS IN NUMBERS

	Iterations	Argument value	Function value
First real root	1	-0.9267	-0.0014
	2	-0.9266	-3.5021e-12
Second real root	1	-2.8814	0.0011
	2	-2.8814	1.7764e-15
First imaginary root	1	0.6394 + 0.9379i	0.2960 - 0.2181i
	2	0.6540 + 0.9398i	2.2511e-06 + 4.8721e-06i
	3	0.6540 + 0.9398i	2.6645e-15 + 4.4409e-16i
Second imaginary root	1	0.6394 - 0.9379i	0.2960 + 0.2181i
	2	0.6540 - 0.9398i	2.2511e-06 - 4.8721e-06i
	3	0.6540 - 0.9398i	2.6645e-15 - 4.4409e-16i

COMPARISON

LAGUERRE'S METHOD WITH MM2

Iterations needed for:	Laguerre's	MM2
First real root	2	3
Second real root	2	3
First imaginary root	3	4
Second imaginary root	3	4

Conclusions

As expected in the theoretical introduction, results proved that the Laguerre's method is the fastest method and also allows us to find complex roots.

Laguerre's and MM2 methods gave us the same roots results. In the Laguerre's method the proper results were obtain in the second iteration for all of the cases, however in the case of finding complex roots, the method needed third iteration to obtain the chosen precision (10^{-9}) of the function value.

Finally, the main advantage of the Laguerre's method is that it is globally convergent and, out of the all presented in the project methods, it gave the fastest results. However the usage of this function was easier after the familiarization with the previous methods.

APPENDIX

```
%TASK1

function [root, iterations] = newton(f, x0, x1, tolerance, imax)

hold on
plot(x0, f(x0), 'go')
plot(x1, f(x1), 'bo')
grid on

root = (x0 + x1)/2;
iterations = 0;

%derivative of f(x)
syms x
dfdx = eval(['@(x)' char(diff(f(x)))]);

for i = 1:imax
    root = root - (f(root)/dfdx(root))
    iterations = iterations + 1
    f(root)

    if abs(f(root)) <= tolerance
        break;
    end
end

plot(root, f(root), 'r*')
legend('left subinterval', 'right subinterval', 'root')
hold off

end

function runNewton
f = @(x)0.5*x*cos(x) - log(x);

imax = 20;
[root2, it2] = newton(f, 5, 6, 10e-9, imax);
[root3, it3] = newton(f, 6.6, 11, 10e-9, imax);

hold on
fplot(f, [2, 11])
title("Newton's method")
hold off

end

function [root, iterations] = falsePosition(f, a, b, tolerance, imax)

hold on
plot(a, f(a), 'go')
plot(b, f(b), 'bo')
grid on

c = 0;
iterations = 0;

for i = 1:imax

    c = (a*f(b)-b*f(a))/(f(b)-f(a))
```

```
    iterations = iterations + 1
    f(c)

    if c > 0
       b = c;
    else
       a = c;
    end

    if abs(f(c)) <= tolerance
        break;
    end

end
root = c;

plot(root, f(root), 'r*')
legend('left subinterval', 'right subinterval', 'root')
hold off

end

function runFalsePosition
f = @(x)0.5.*x.*cos(x) - log(x);

imax = 100;
[root2, it2] = falsePosition(f, 5, 6, 10e-9, imax);
[root3, it3] = falsePosition(f, 7, 8, 10e-9, imax);

hold on
fplot(f, [2,11])
title("False Position method")
hold off
end

%TASK2

function [root, iterations] = MM1(f, x0, x1, x2, tolerance, imax)

iterations = 0;
c = 1;

while abs(c) > tolerance && x0 ~= x1 && x1 ~= x2 && iterations < imax

    z0 = x0 - x2;
    z1 = x1 - x2;

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

    delta = b*b - 4*a*c;

    if abs(b + sqrt(delta)) >= abs(b - sqrt(delta))
        root = x2 - (2*c)/(b + sqrt(b*b - 4*a*c));
    else
        root = x2 - (2*c)/(b - sqrt(b*b - 4*a*c));
    end

    new0 = abs(root - x0);
```

```
    new1 = abs(root - x1);
    new2 = abs(root - x2);

    if new0 >= new1 && new0 >= new2
        x0 = x2;
        x2 = root;
    elseif new1 >= new0 && new1 >= new2
        x1 = x2;
        x2 = root;
    elseif new2 >= new0 && new2 >= new1
        x2 = root;
    end
    iterations = iterations + 1
    root
    f(root)
end
end

function runMM1
P = [2 5 -2 3 7];
f = @(x)P(1)*x^4 + P(2)*x^3 + P(3)*x^2 + P(4)*x + P(5);

imax = 10;

hold on;
fplot(f)
grid on
xlim([-5, 3])
ylim([-100, 600])

x0 = 0;
x1 = 1.1i;
x2 = 0.8i;
[root1, it1] = MM1(f, x0, x1, x2, 10e-9, imax);
plot(x0, f(x0), 'g.')
plot(root1, f(root1), 'r*')
plot(x1, f(x1), 'g.')
plot(x2, f(x2), 'g.')


x0 = 0;
x1 = -1.1i;
x2 = -0.8i;
[root2, it2] = MM1(f, x0, x1, x2, 10e-9, imax);
plot(x0, f(x0), 'g.')
plot(x1, f(x1), 'g.')
plot(x2, f(x2), 'g.')
plot(root2, f(root2), 'r*')

x0 = -3.1;
x1 = -3;
x2 = -2.5;
[root3, it3] = MM1(f, x0, x1, x2, 10e-9, imax);
plot(x0, f(x0), 'g.')
plot(x1, f(x1), 'g.')
plot(x2, f(x2), 'g.')
plot(root3, f(root3), 'r*')

x0 = -1.1;
x1 = -1;
x2 = -0.8;
```

```
[root4, it4] = MM1(f, x0, x1, x2, 10e-9, imax);
plot(x0, f(x0), 'g.')
plot(x1, f(x1), 'g.')
plot(x2, f(x2), 'g.')
plot(root4, f(root4), 'r*')

title("MM1 method")
legend('2x^4 + 5x^3 - 2x^2 + 3x + 7');
hold off;

figure(2)
hold on
plot(real(root1), imag(root1), 'r*')
plot(0, f(0), 'g.')
plot(1.1i, f(1.1i), 'g.')
plot(0.8i, f(0.8i), 'g.')

plot(real(root2), imag(root2), 'r*')
plot(0, f(0), 'g.')
plot(-1.1i, f(-1.1i), 'g.')
plot(-0.8i, f(-0.8i), 'g.')
xlabel('Re(f)');
ylabel('Im(f)');
title('Imaginary roots using MM1')
grid on;
hold off

end

function [xk, iterations] = MM2(P, xk, tolerance, imax)

iterations = 0;
dP = polyder(P);
ddP = polyder(dP);

c = 1;

while abs(c) > tolerance && iterations < imax

    a = polyval(ddP, xk)/2;
    b = polyval(dP, xk);
    c = polyval(P, xk);

    delta = b*b - 4*a*c;
    root1 = -2*c/(b+sqrt(delta));
    root2 = -2*c/(b-sqrt(delta));

    if abs(root1) <= abs(root2)
        zmin = root1;
    else
        zmin = root2;
    end

    xk = zmin +xk;
    iterations = iterations +1
    xk
    polyval(P, xk)


end
end
```

```
function runMM2
P = [2 5 -2 3 7];
f = @(x)P(1)*x^4 + P(2)*x^3 + P(3)*x^2 + P(4)*x + P(5);

imax = 10;

hold on
fplot(f)
grid on
xlim([-5, 3])
ylim([-100, 600])

xk = -3;
[root1, it1] = MM2(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root1, f(root1), 'r*')

xk = -1;
[root2, it2] = MM2(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root2, f(root2), 'r*')

xk = -1i;
[root3, it3] = MM2(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root3, f(root3), 'r*')

xk = 1i;
[root4, it4] = MM2(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root4, f(root4), 'r*')

title("MM2 method")
legend('2x^4 + 5x^3 - 2x^2 + 3x + 7');
hold off;

figure(2)
hold on
plot(real(root3), imag(root3), 'r*')
plot(-1i, 'g.')
plot(real(root4), imag(root4), 'r*')
plot(1i, 'g.')
xlabel('Re(f)');
ylabel('Im(f)');
title('Imaginary roots using MM2')
grid on;
hold off

end

function [root, iterations] = modifiedNewton(f, x, tolerance, imax)

hold on
plot(x, f(x), 'b.')
grid on

root = x;
iterations = 0;

syms x
dfdx = eval(['@(x)' char(diff(f(x)))]);
```

```
for i = 1:imax
    root = root - (f(root)/dfdx(root))
    iterations = iterations + 1
    f(root)

    if abs(f(root)) <= tolerance
        break;
    end
end

plot(root, f(root), 'r*')
legend('initial points', 'root')
hold off

end

function runModifiedNewton
P = [2 5 -2 3 7];
f = @(x)P(1)*x^4 + P(2)*x^3 + P(3)*x^2 + P(4)*x + P(5);

imax = 50;

[root1, it1] = modifiedNewton(f, -1, 10e-9, imax);
[root2, it2] = modifiedNewton(f, -3, 10e-9, imax);

hold on
fplot(f, [-5, 3])
title("Newton's method")
grid on
hold off

end


%TASK3

function [root, iterations] = laguerre(P, xk, tolerance, imax)

iterations = 0;

dP = polyder(P);
ddP = polyder(dP);
n = length(P) - 1;

while abs(polyval(P, xk)) > tolerance && iterations < imax

    eDen = sqrt((n-1) * ((n-1) * (polyval(dP, xk)^2) - n*polyval(P,
xk)*polyval(ddP, xk)));

    if abs(polyval(dP, xk) + eDen) > abs(polyval(dP, xk) - eDen)
        den = polyval(dP, xk) + eDen;
    else
        den = polyval(dP, xk) - eDen;
    end

    % formula for Laguerre's method
    xk = xk - (n * polyval(P, xk) / den)
    iterations = iterations + 1
    polyval(P, xk)
end
```

```
root = xk;
end

function runLaguerre
P = [2 5 -2 3 7];
f = @(x)P(1)*x^4 + P(2)*x^3 + P(3)*x^2 + P(4)*x + P(5);

imax = 10;

hold on
fplot(f)
grid on
xlim([-5, 3])
ylim([-100, 600])

xk = -3;
[root1, it1] = laguerre(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root1, f(root1), 'r*')

xk = -1;
[root2, it2] = laguerre(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root2, f(root2), 'r*')

xk = -1i;
[root3, it3] = laguerre(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root3, f(root3), 'r*')

xk = 1i;
[root4, it4] = laguerre(P, xk, 10e-9, imax);
plot(xk, f(xk), 'g.')
plot(root4, f(root4), 'r*')

title("Laguerre's method")
legend('2x^4+0.5x^3-5x^2+2x-3', 'Initial points', 'Roots')
hold off

end
```