Numerical Methods

Project B No. 5

JULIA CZMUT
300168

Problem 1.

1. 1. Description of the program

The goal of this task is to find all zeros of the function

$$f(x) = 1.2x \cdot sin(x) - 2ln(x+2)$$

in the interval [2, 12] using:
a) the secant method,
b) the Newton's method.

1. 2. Theoretical background

Secant method

In this method, the two last obtained points are denoted x_{n-1} and x_n and a secant line joins them. A new point in the method is defined by:

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

It may happen that the next interval does not contain the root.

HOW DO WE CHOOSE THE INITIAL POINTS?

The idea of this method is to approximate a root using a secant line, so the initial points are to create a straight line and an interval surrounding the root. An ideal situation is when the root is located on a steep slope of the function. The reason for this will be explained on next pages.

The order of convergence is $p \approx 1.618$

Order of convergence for an iterative method is the largest number $p \geq 1$ such that

$$\lim_{n \to \infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^p} = k < \infty$$

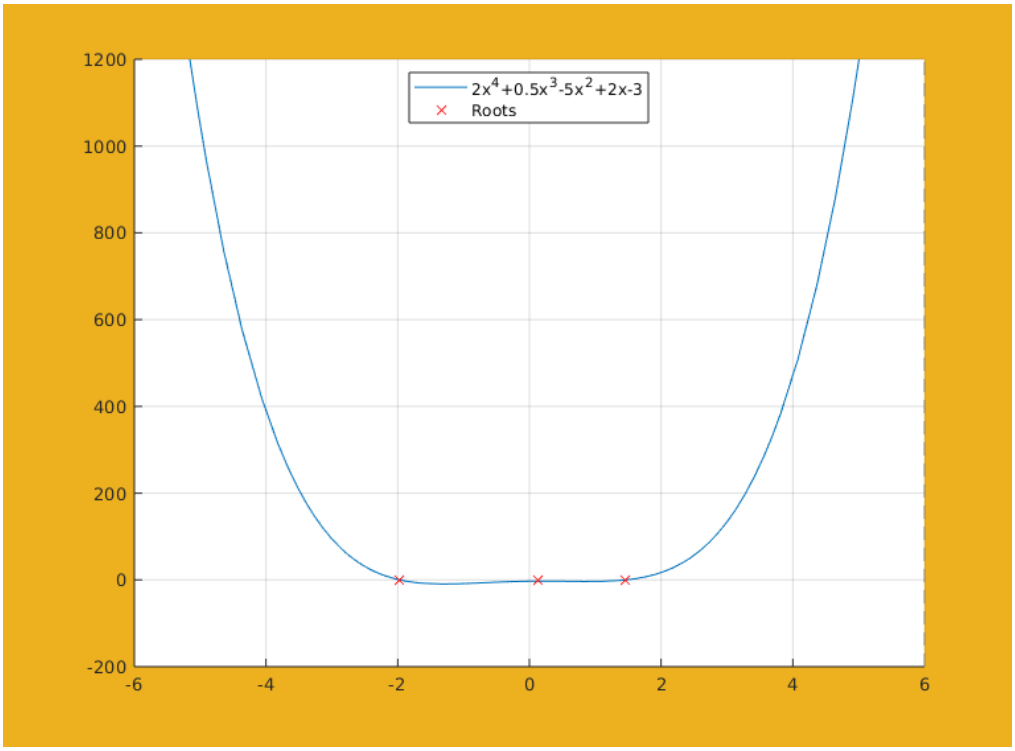
where
α - the root
k – convergence factor.

If p = 1, the method is convergent linearly and if p = 2 the convergence is quadratic.
Secant method is only locally convergent, so a divergence may happen.

WHEN DOES IT DIVERGE?

The secant method diverges when the initial interval is too small, or its ending points are equal to each other and when the derivative at the root is close to zero. The last case means our root is in the minimum or maximum of the function and therefore, the plot at that location is “flat”.
In case of our function, because the roots in the interval [2, 12] are on the steep slopes of the graph, the secant method practically cannot diverge, provided that ending points of the chosen interval are not equal to each other.

However, the polynomial from the next task may be an example of when this method diverges.



As we can see, the roots are located in a flat part of the function. Approximating them using the secant method would probably fail. The derivative of the function near the roots is close to zero.

Newton's method
It is also called the tangent method. It is performed by approximating the function f(x) by the first order part of its expansion into a Taylor series at a current approximation of the root x_n.

The approximation is:

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

Next point x_{n+1} acts a root of the function described above.

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

Finally, the iteration formula is:

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

Newton's method is also locally convergent.

WHEN DOES IT DIVERGE AND HOW DO WE CHOOSE INITIAL POINTS?

If an initial point is outside the root's set of attraction, a divergence may occur, but if it does not and the method converges – Newton's method is very fast, its order of convergence is 2.

The method is especially effective if the function derivative at the root is sufficiently far from zero, so the slope of the function is steep. So it may diverge when the derivative is close to zero, because it is sensitive to numerical errors when close to the root.

Newton's method is faster than secant method, but the secant method has the advantage of needing less information - it does not need the function's derivative like Newton's method does.

WHAT TO DO IF THE METHOD DIVERGES?

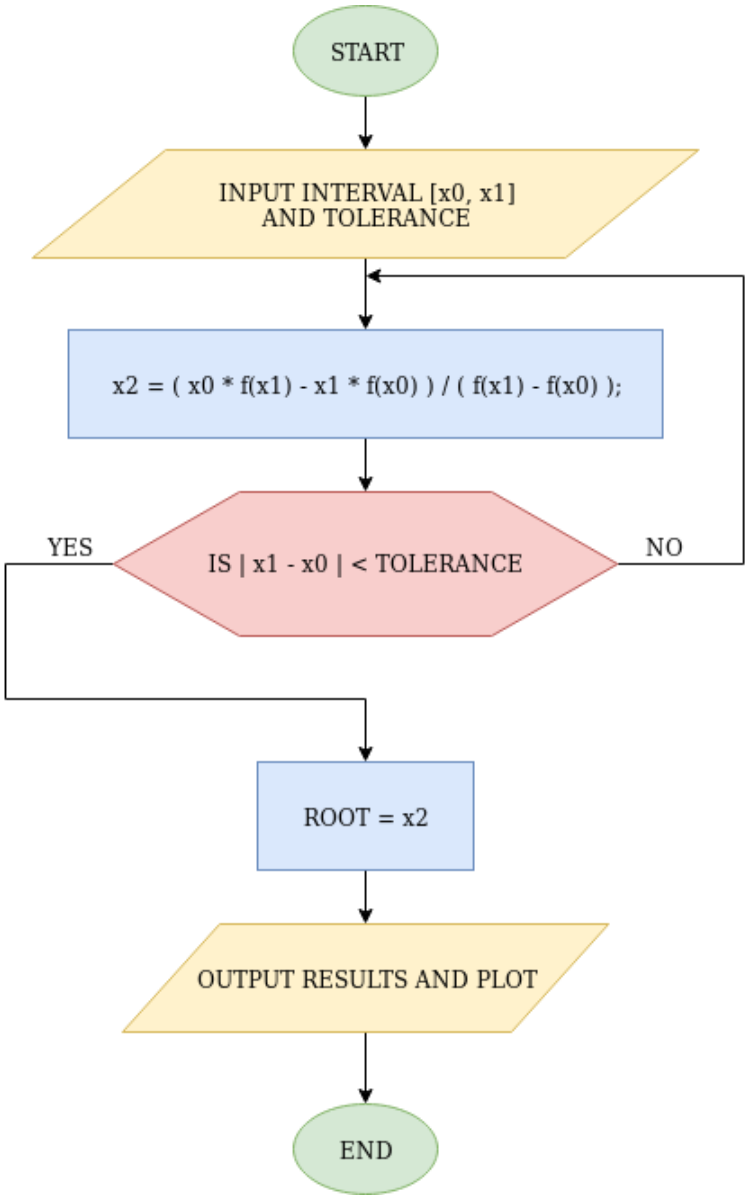
Because **Newton's method** is only **locally convergent**, in a situation when we want to create a really effective algorithm for finding all roots of a function, we can still get the profit from its speed by combining it with another method, for example **bisection method** or **regula falsi method** which are **globally convergent**.

This way, the slow but globally convergent method isolates the root on the domain of the function and then the fast and locally convergent method approximates the root with a high accuracy.

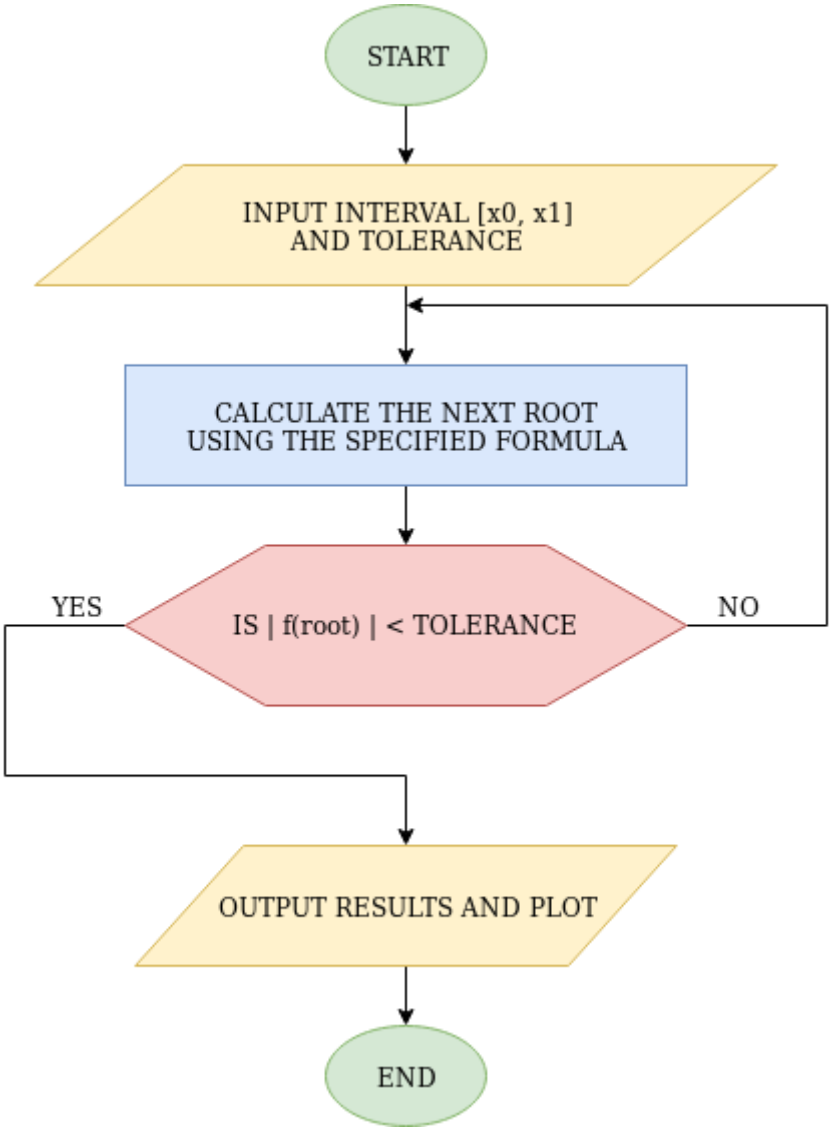
This can also be done for a secant method's divergence as it is also a locally convergent method.

1. 3. Algorithm

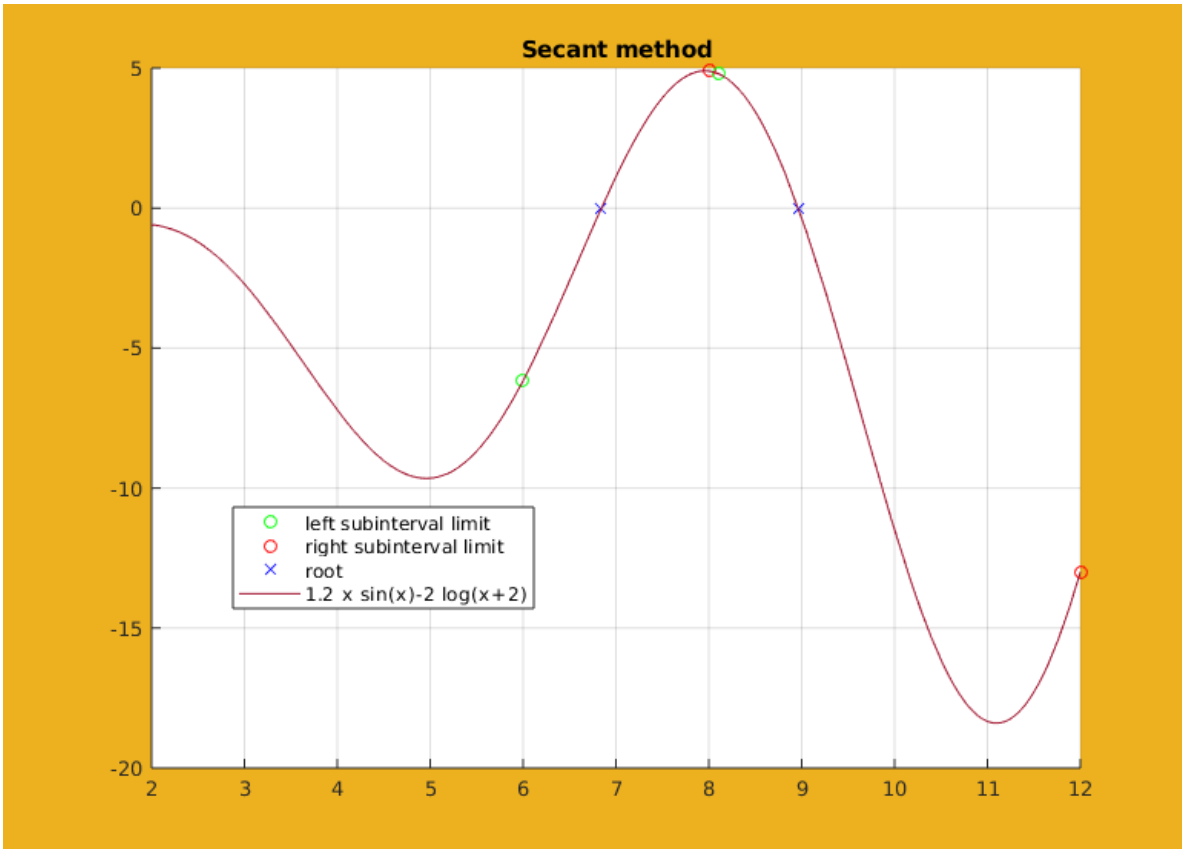
Secant method

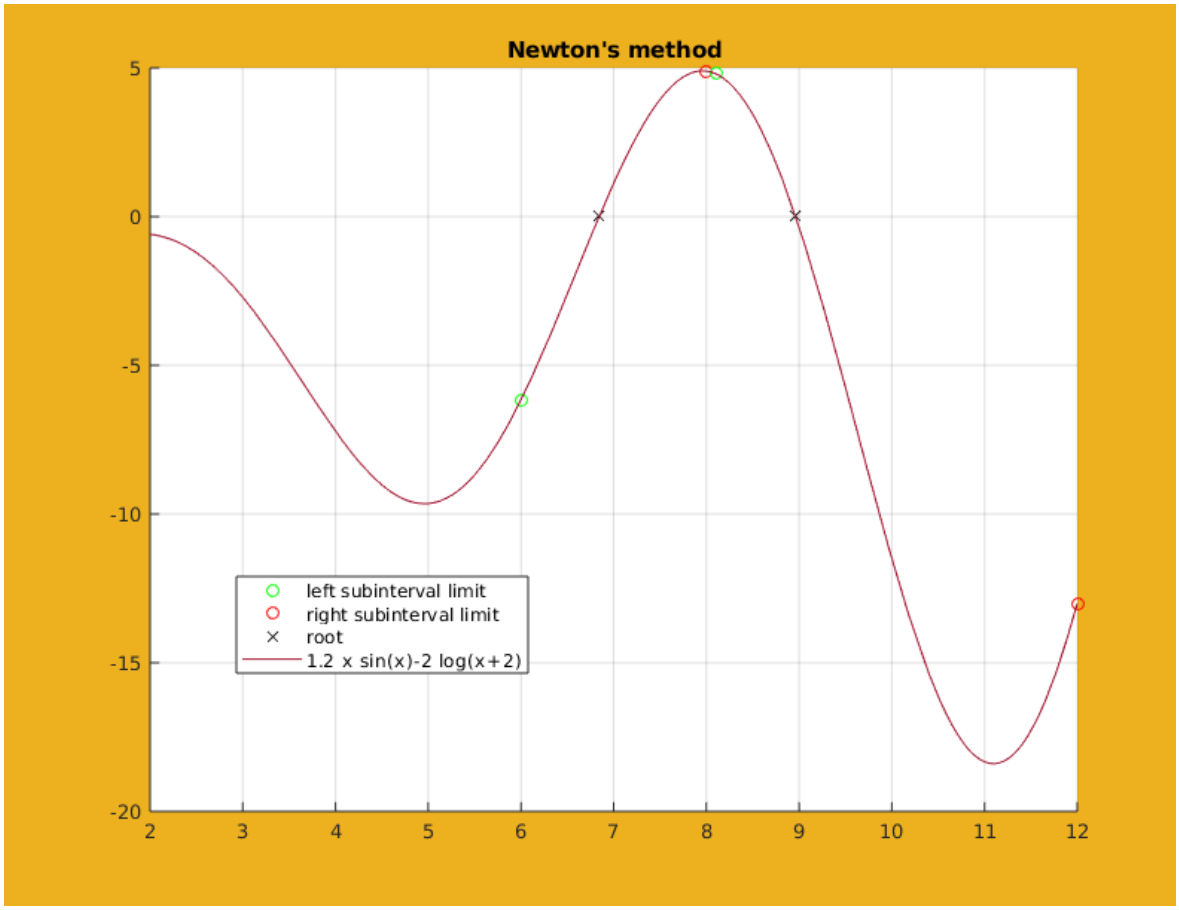


Newton’s method



RESULTS OF APPROXIMATING THE ROOTS





SECANT METHOD

	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST ROOT **6.8428**	1	7.1155	1.8944
	2	6.5567	-2.1683
	3	6.8549	0.089148
	4	6.8432	0.002517
	5	6.8428	-4.846e-06
	6	6.8428	2.5839e-10
	7	**6.8428**	-2.6645e-15
SECOND ROOT **8.9633**	1	9.1518	-1.862
	2	8.6758	2.3528
	3	8.9415	0.20093
	4	8.9663	-0.027644
	5	8.9633	0.0002281
	6	8.9633	2.5259e-07
	7	8.9633	-2.3119e-12
	8	**8.9633**	1.2434e-14

NEWTON'S METHOD

	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST ROOT **6.8428**	1	6.837	-0.0426
	2	6.8428	-3.7716e-05
	3	**6.8428**	-3.0117e-11
SECOND ROOT **8.9633**	1	8.9195	0.40005
	2	8.9641	-0.0071134
	3	8.9633	-2.0303e-06
	4	**8.9633**	-1.6875e-13

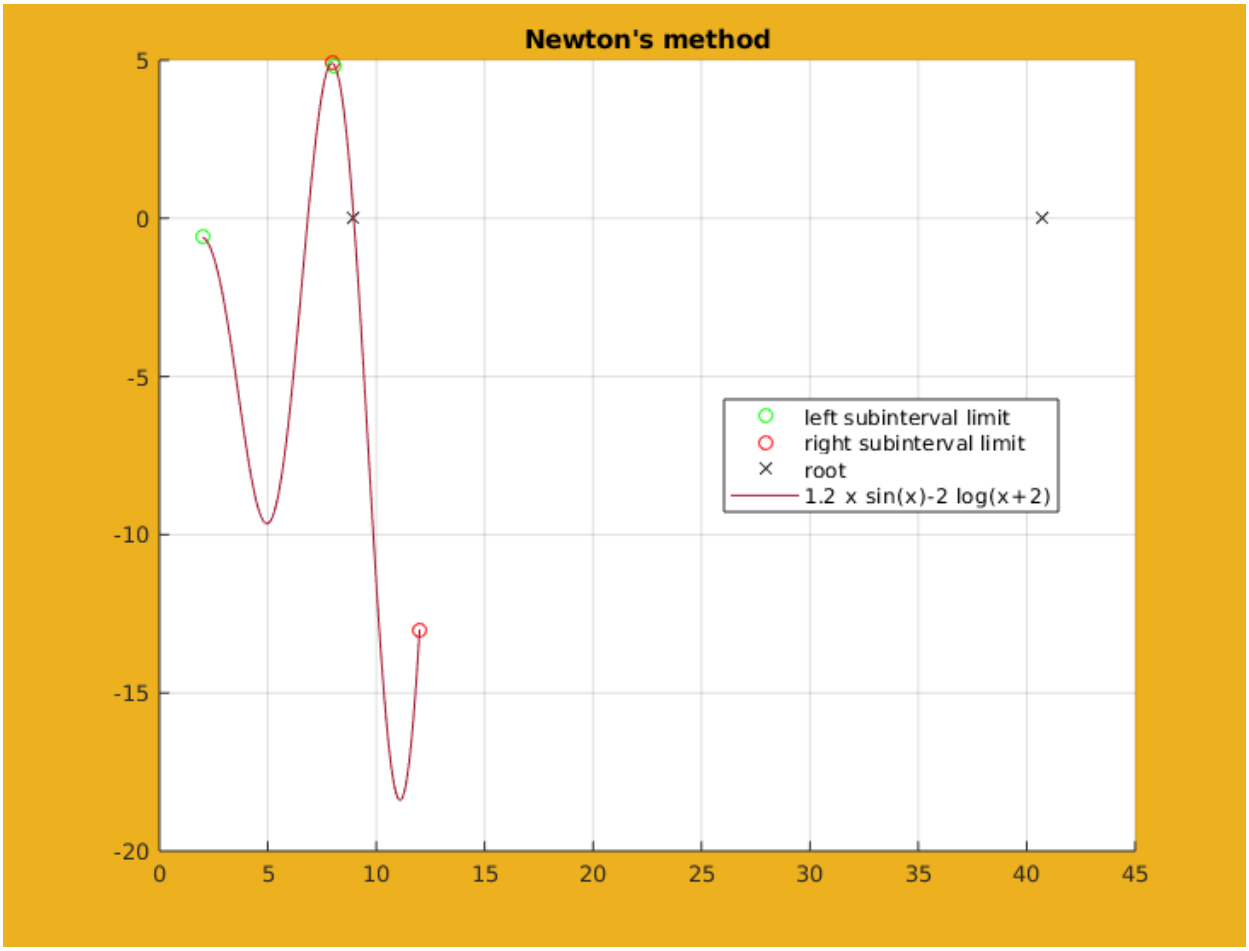
COMPARISON OF BOTH METHODS' RESULTS

	SECANT METHOD	NEWTON'S METHOD
Root from interval [6, 8]	6.8428	6.8428
Iterations needed	7	3
Root from interval [8.1, 12]	8.9633	8.9633
Iterations needed	8	4
Precision used	10^{-9}	10^{-9}

EXAMPLE OF NEWTON'S METHOD DIVERGENCE

Chosen subintervals: **[2, 8]** and **[8.1, 12]**
Found roots:

40.6863
8.9633



COMMENTS

Newton's method is very fast, provided it converges. Unfortunately, divergence happens quite easily. In above case, the initial point in the first subinterval was too far from the root, in other words: it was **outside of the root's set of attraction** and the method approximated the root to be **40.6863**, while the actual root is **6.8428**. As we can see, the divergence is quite significant.

Another reason why previously chosen subintervals were better is that Newton's method works very well when the slope of the function f(x) is steep and in this case the improper initial interval **[2, 8]** has a clear minimum in the middle, which combined with the fact that the initial point inside of this interval is exactly in half, creates a situation where the derivative of the function is close to zero.

1.4. Conclusions

The Newton's method proved to be faster than Secant method. It needed less iterations to obtain the same result with the same accuracy.

The initial intervals were chosen after looking at the graph of our given function f(x). The roots are approximately in the middle of those subintervals. In secant method, a secant line joins two subsequent points, so they were represented by the ending points of the subintervals which additionally, when connected through the secant line, corresponded to the shape of the graph. To compare the methods properly, it made sense to use the same subintervals for Newton's method. It needed only one initial point and because the roots were approximately in the middle of the subintervals, initial root in Newton's method was chosen to be exactly in half of the subinterval, so it managed to be in the set of attraction of the actual root of the function. If the initial point was to be too far from the actual root, the method would diverge and the proper root would not be found, as one example showed.
Both secant and Newton's methods are reliable and both methods established the roots quite well.

SECANT METHOD
ADVANTAGES:
- it does not need the function's derivative

DISADVANTAGES:
- fails to converge when the isolation interval is too small and slope is "flat"
- only locally convergent

NEWTON'S METHOD
ADVANTAGES:
- when converges, it is very fast

DISADVANTAGES:
- it needs the function's derivative
- very sensitive to numerical errors when the derivative at the root is close to zero
- only locally convergent

Problem 2.

2. 1. Description of the program

In this problem we will find all (real and complex) roots of the polynomial

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

$$\begin{bmatrix} a_4 & a_3 & a_2 & a_1 & a_0 \end{bmatrix} = \begin{bmatrix} 2 & 0.5 & -5 & 2 & -3 \end{bmatrix}$$

using the Muller's method implementing both the MM1 and MM2 versions. The results will be compared. Real roots will also be found using the Newton's method and compared with the MM2 version of the Muller's method (using the same initial points).

2. 2. Theoretical background

Muller's method is one of methods developed specially for finding roots of polynomials, both **real** and **complex**. Its idea is to approximate the polynomial locally with a **parabola**. It is done using a quadratic interpolation based on **three different points**. It is interesting to notice that the secant method from the previous task is a version of Muller's: with linear interpolation based on two points. It is **locally convergent** with order of convergence equal to **1.84.**
Muller's method has two versions.

MM1

In this version, we consider three points: x_0, x_1, x_2 together with corresponding polynomial values $f(x_0)$, $f(x_1)$ and $f(x_2)$. A parabola is constructed using those points and than its roots are used for next approximation.

We can assume that x_2 is an actual approximation of the root of the polynomial. So we introduce a new variable:

$$z = x - x_2$$

$$z_0 = x_0 - x_2$$

$$z_1 = x_1 - x_2$$

The interpolating parabola is described using the new variable:

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

Using our three points we can construct such equations:

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

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

$$c = f(x_2)$$

The roots are given by:

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

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

For the next iteration we choose the root which has a smaller absolute value, so it is closer to the current best approximation x_2. $x_3 = x_2 + z_{min}$

$$z_{min} = z_+, \quad if \quad |b + \sqrt{b^2 - 4ac}| \geq |b - \sqrt{b^2 - 4ac}|$$
$$z_{min} = z_-, \quad otherwise$$

MM2

A slightly more numerically effective version of Muller's method. It uses values of a polynomial and its first and second order derivatives at one point.

From the equation for the parabola:

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

We have that:

$$z = x - x_k$$

And for point z = 0:

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

Therefore, the root is calculated using such formula:

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

The approximated root is a root with smaller absolute value

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

And z_{min} is chosen the same way as in MM1.

WHEN DO THEY DIVERGE AND HOW DO WE CHOOSE INITIAL POINTS?

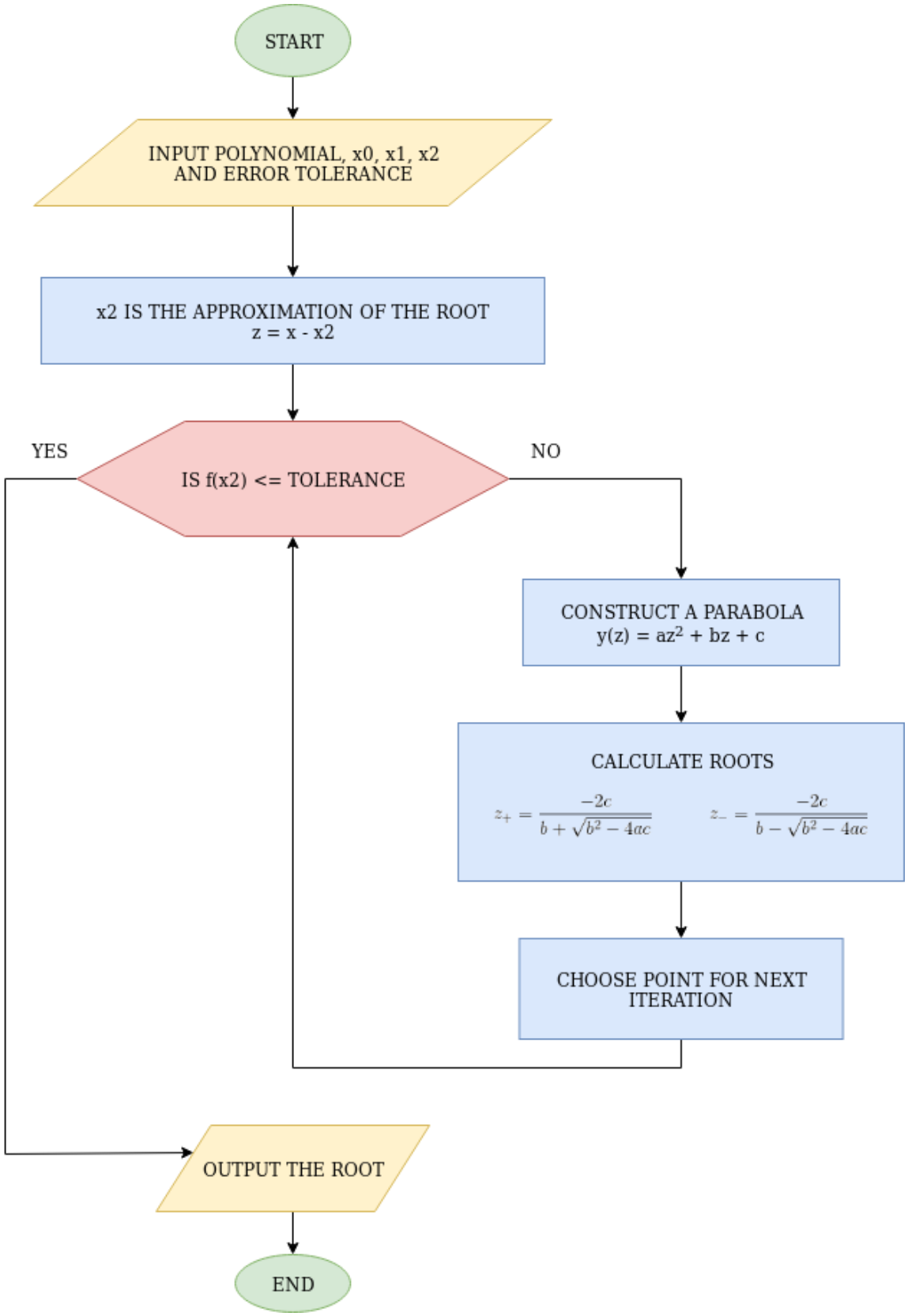
Muller's method is locally convergent, so we need to look at the graph of the considered function when choosing the initial points. We need three of them in MM1 and they contribute to creating a parabola which will help in finding the next iteration's root approximation. In MM2 the initial point should be simply relatively close to the root.
The method will diverge when the points are chosen incorrectly and when they are too far from the root.

WHAT IF WE WANT ALL ROOTS?

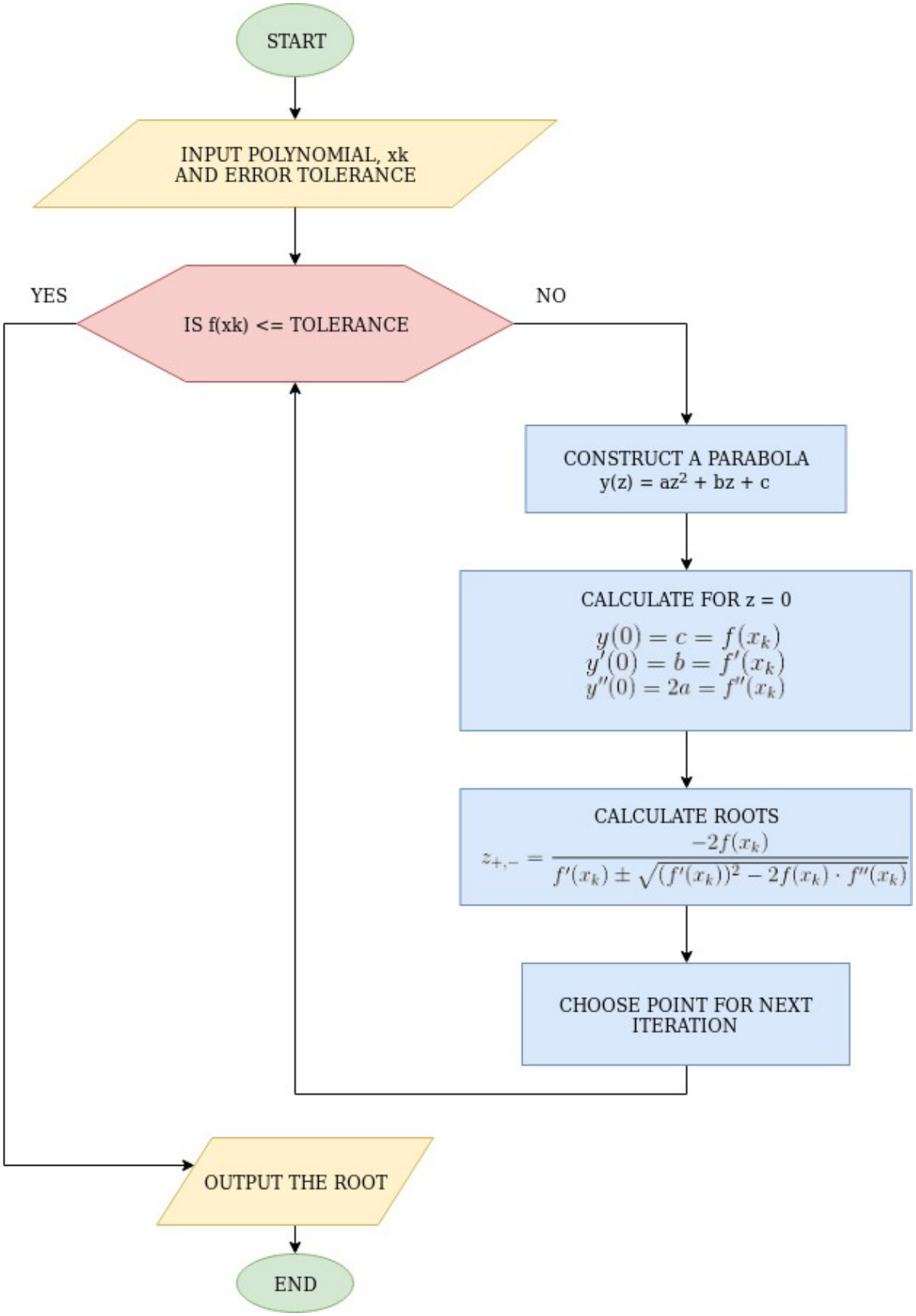
One thing which can help is **deflation by a linear term**.
When we are looking for many or all roots of a polynomial, then to simplify the procedure, after finding one root α, before looking for the next one, the polynomial should be divided by (x – α). This reduces the order of the polynomial and excludes the found root from further calculations. It is done using the standard **Horner's algorithm**, **reversed Horner's algorithm** or c**ombined Horner's scheme**. To increase the precision of calculating roots from deflated polynomials we can use **root polishing**.

2. 3. Algorithm

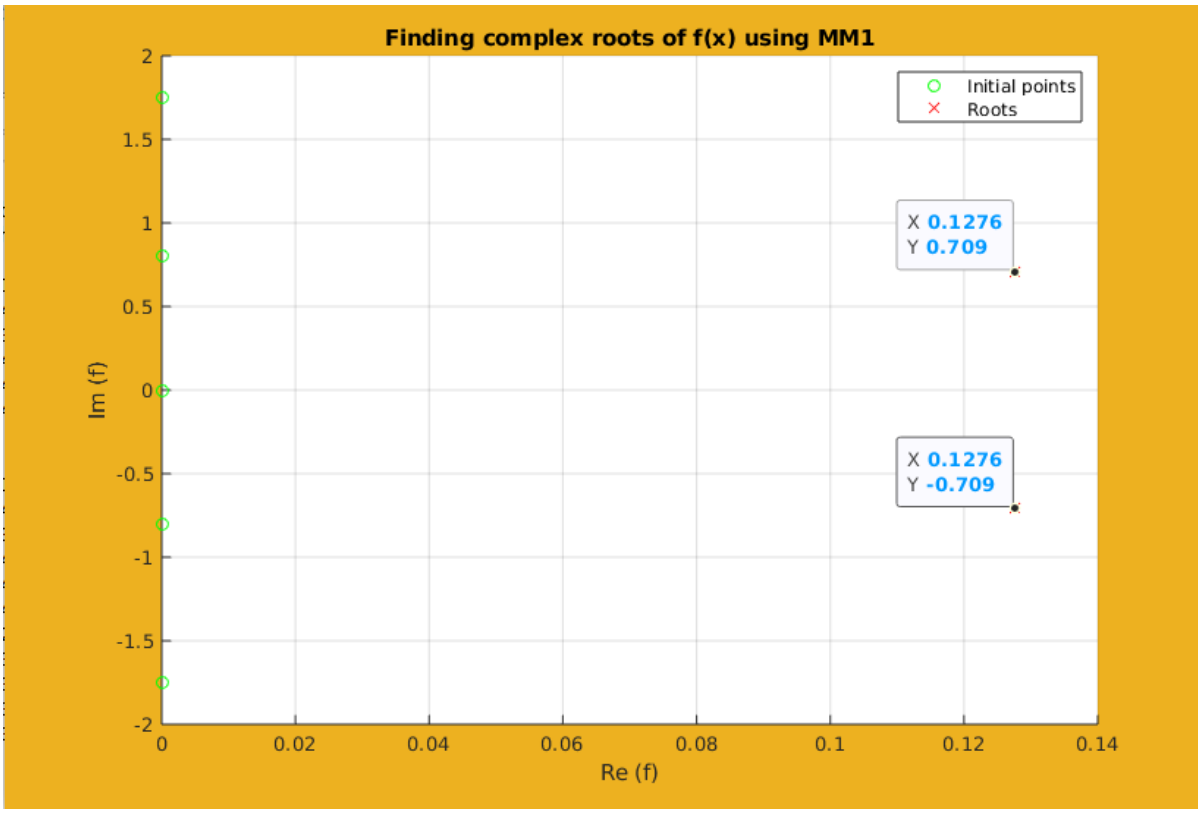
MM1

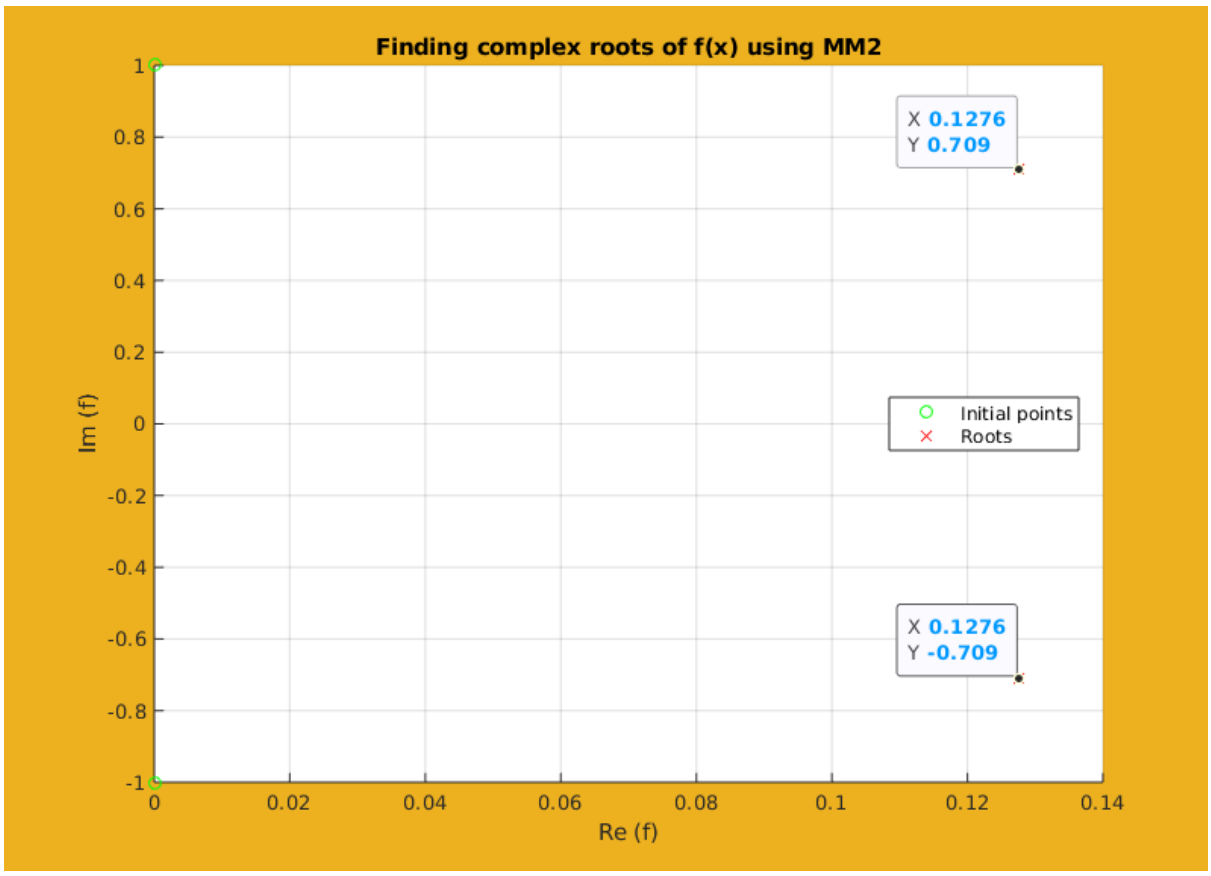


MM2

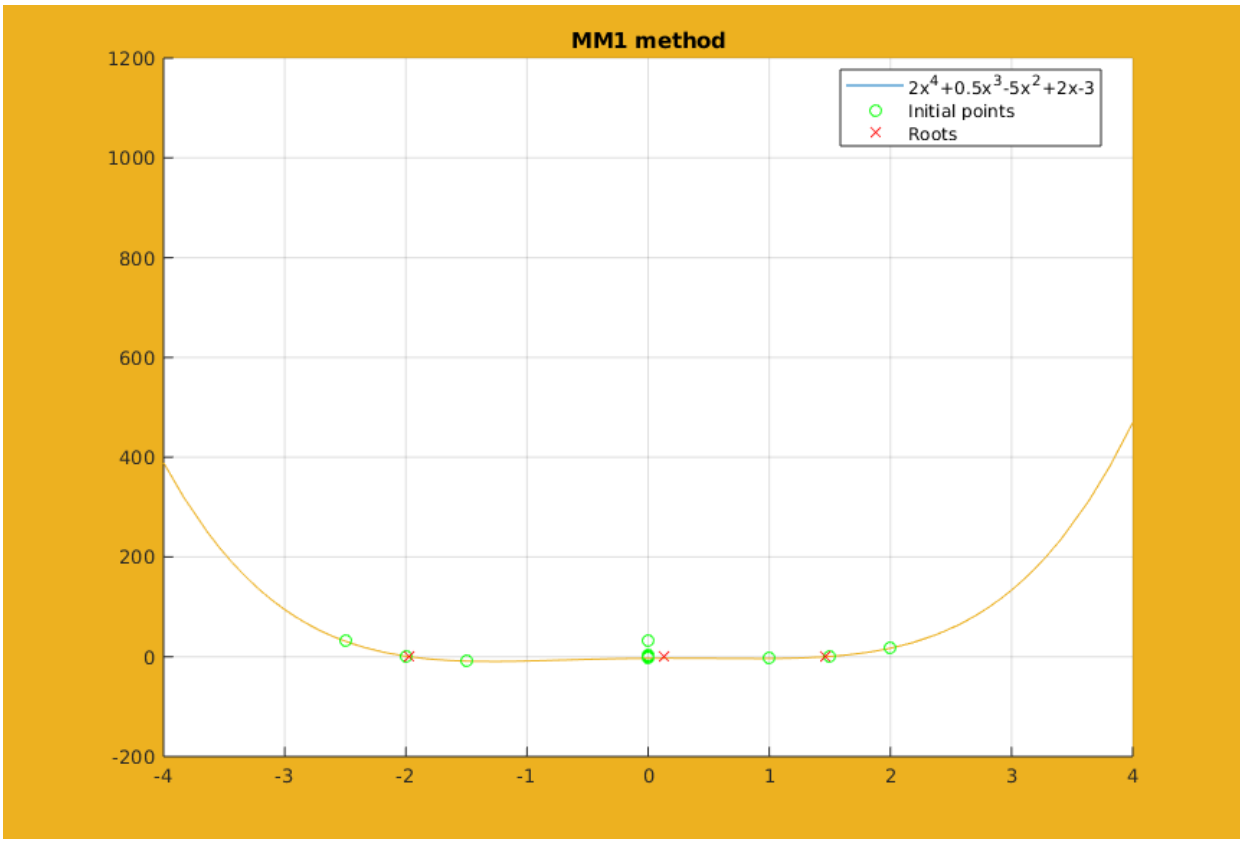

START
INPUT POLYNOMIAL, xk
AND ERROR TOLERANCE
YES
IS f(xk) <= TOLERANCE
NO
CONSTRUCT A PARABOLA
y(z) = az^2 + bz + c
CALCULATE FOR z = 0
$y(0) = c = f(x_k)$
$y'(0) = b = f'(x_k)$
$y''(0) = 2a = f''(x_k)$
CALCULATE ROOTS
$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f(x_k) \cdot f''(x_k)}}$
CHOOSE POINT FOR NEXT
ITERATION
OUTPUT THE ROOT
END

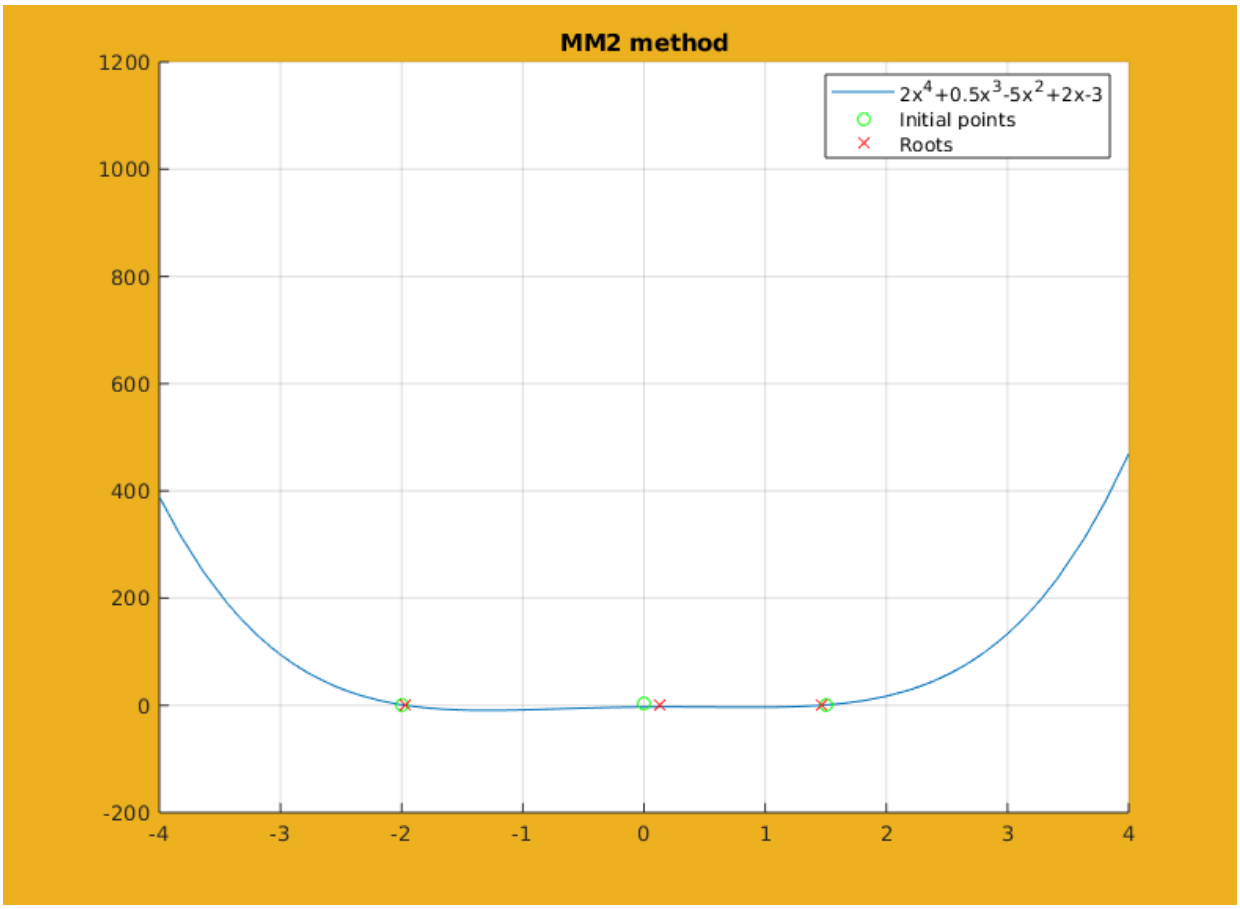
COMPLEX ROOTS OF CONSIDERED POLYNOMIAL





RESULTS OF APPROXIMATING ROOTS USING MULLER'S METHOD





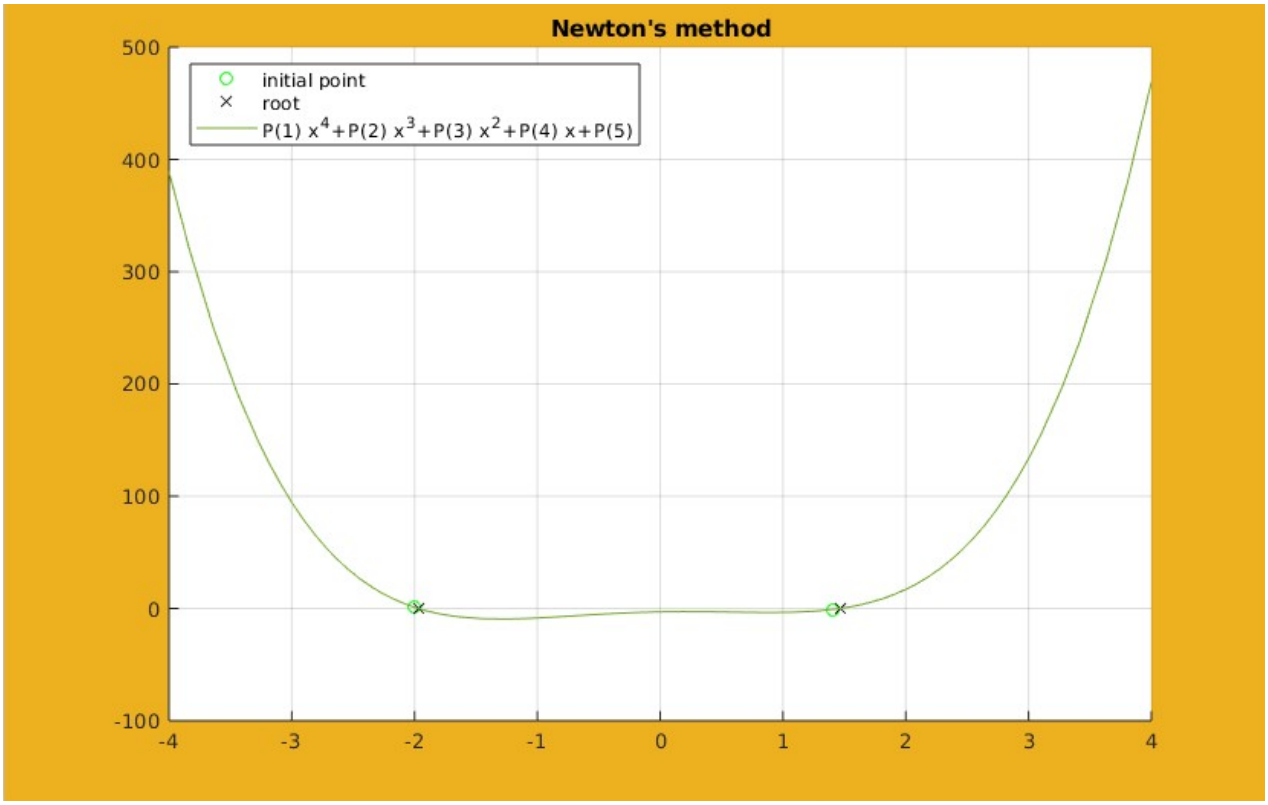
MM1

	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST ROOT **-1.9713**	1	-1.9742	0.099193
	2	-1.9713	0.00056639
	5	**-1.9713**	-2.1628e-08
SECOND ROOT **0.1276-0.7090i**	1	0.085219-0.7399i	0.35352-0.38231i
	2	0.12494-0.70888i	0.0025398-0.025627i
	3	0.12763-0.70905i	0.00013148-2.5435e-05i
	6	**0.12764-0.70904i**	0+5.5511e-16i
THIRD ROOT **0.1276+0.7090i**	1	0.085219+0.7399i	0.35352+0.38231i
	2	0.12494+0.70888i	0.0025398+0.025627i
	3	0.12764+0.70904i	0.00013148+2.5435e-05i
	6	**0.12764+0.70904i**	0-5.5511e-16i
FOURTH ROOT **1.466**	1	1.4716	0.088104
	2	1.4661	0.00098462
	5	**1.466**	3.7303e-14

MM2

	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST ROOT **-1.9713**	1	-1.9713	-0.00036478
	2	-1.9713	1.5099e-14
	3	**-1.9713**	2.6645e-15
SECOND ROOT **0.1276-0.7090i**	1	0.15605-0.70876i	-0.054579-0.26358i
	2	0.12765-0.70903i	-4.2534e-05+0.00012874i
	4	**0.12764-0.70904i**	0-2.2204e-16i
THIRD ROOT **0.1276+0.7090i**	1	0.15605+0.70876i	-0.054579-0.26358i
	2	0.12765+0.70903i	-4.2534e-05-0.00012874i
	4	**0.12764+0.70904i**	0+2.2204e-16i
FOURTH ROOT **1.4660**	1	1.466	-0.00048803
	2	1.466	3.6282e-13
	3	**1.466**	0

RESULTS OF APPROXIMATION USING NEWTON'S METHOD (only real roots)



	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST REAL ROOT **-1.9713**	1	-1.9722	0.030533
	2	-1.9713	3.1559e-05
	3	**-1.9713**	3.383e-11
SECOND REAL ROOT **1.466**	1	1.4733	0.11541
	2	1.4661	0.0011896
	3	1.466	1.3076e-07
	4	**1.466**	2.2204e-15

COMPARISON OF MM2 AND NEWTON'S METHOD (for real roots)

	MM2	NEWTON'S
First root	-1.9713	-1.9713
Iterations needed	3	3
Second root	1.4663	1.466
Iterations needed	3	4
Precision used	10^{-9}	10^{-}

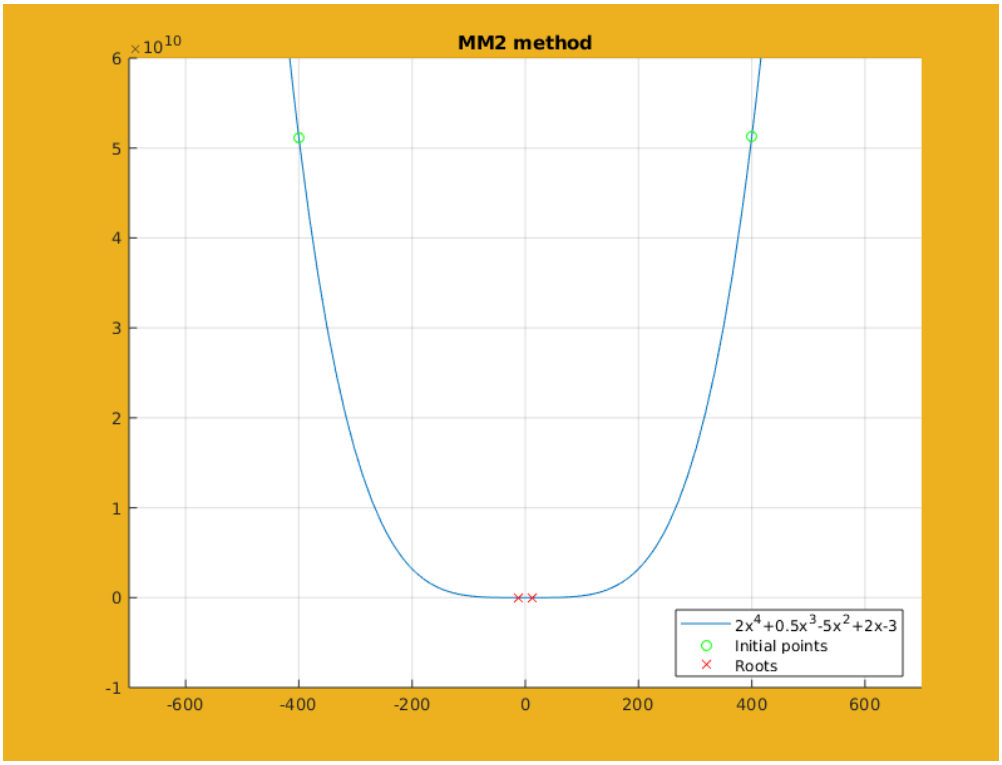
COMPARISON OF MM1 AND MM2 METHODS

	MM1	MM2
First root	-1.9713	-1.9713
Iterations needed	4	3
Second root	0.12764-0.70904i	0.12765-0.70903i
Iterations needed	6	4
Third root	0.12764+0.70904i	0.12765+0.70903i
Iterations needed	6	4
Fourth root	1.466	1.4663
Iterations needed	4	3
Precision used	10^{-9}	10^{-9}

EXAMPLE OF MM2 METHOD DIVERGENCE

Chosen initial points: **400** instead of **1.5**
-400 instead of **-2**

Obtained roots (in 10 iterations) : **12.4728** instead of **1.4663**
-12.596 instead of **-1.9713**



COMMENTS
For MM2 method, the approximated roots started to diverge from the actual roots only after choosing an initial point 30 times bigger than the proper initial point and even then, it would not diverge that much. When the initial points are 200 bigger than previously, then the approximations become worse: about 4 times bigger than the actual root.

---

2. 4. Conclusions

Muller's method provides an effective way of finding both real and complex roots of polynomials.

Although it seems there is not difference between the number of iterations of MM1 and MM2 methods, this criteria is not relevant in this case, because in both versions the precision used was 10^{-9} and the MM2 method converged to precise solutions quicker and stagnated, so generally, its precision should be decreased or an additional condition considering the difference between successive approximations.

Estimation of the real roots of our polynomial with the same initial points from MM2 method using Newton's method gave very accurate solutions. That is because the initial points were well chosen and managed to be in the roots' sets of attraction.

In the previous task, Newton's method was using an initial interval, but for the sake of this comparison, it uses just one initial point.

MM1 METHOD
ADVANTAGES:
- is able to find complex roots

DISADVANTAGES:
- less convenient usage because of having to choose three initial points

---

MM2 METHOD
ADVANTAGES:
- is able to find complex roots
- more convenient to use: only one initial point is to be chosen
- faster than MM1

DISADVANTAGES:
- less progress per iteration than Newton's method because of smaller order of convergence

---

NEWTON'S METHOD
ADVANTAGES:
- big progress per iteration because of quadratic order of convergence
- it is very fast provided it converges

DISADVANTAGES:
- is not able to find complex roots

Problem 3.

3. 1. Description of the program

In this task we will find all (real and complex) roots of the polynomial f(x) from Problem 2. using the Laguerre's method. The results will be compared with the MM2 version of the Muller's method (using the same initial points).

3. 2. Theoretical background

Laguerre's method is regarded as one of the best methods for polynomial root finding. In this method, we start by choosing an initial point x_k.

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

The formula defining the next iteration's approximation is more complex than of any previously described method, because it takes into account the order of the polynomial denoted by n. Therefore, we can draw a hypothesis that it will prove to be the best out of all used in this project.

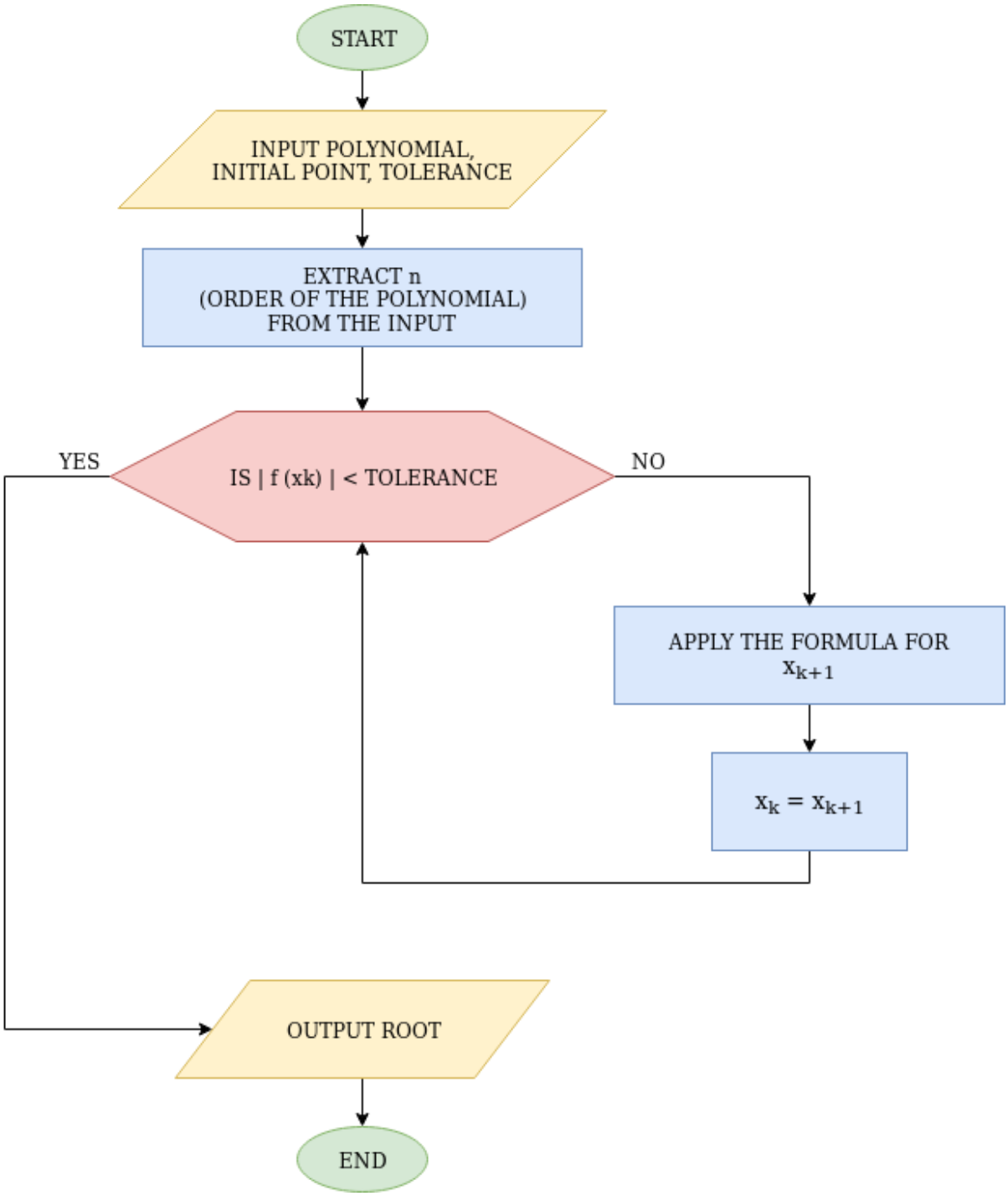
Laguerre's method is **globally convergent**, so it will converge from any initial point.
It converges **cubically** when the initial point is close enough to the root and when it is a single root. On the other hand, when it is a multiple root, the method converges **linearly**.

There is actually little formal analysis for a case with complex roots, however Laguerre's method usually approximates them well despite this fact.

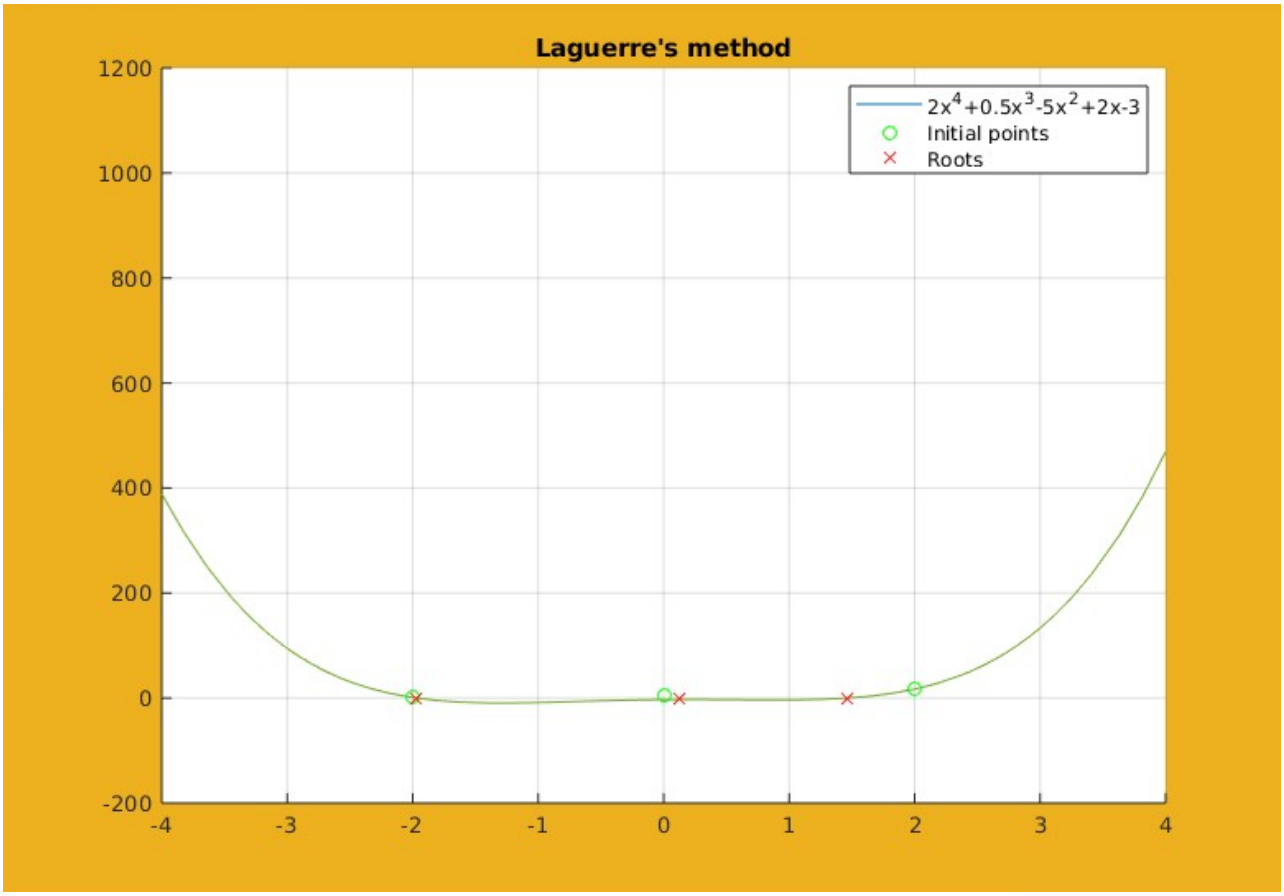
COULD IT EVEN DIVERGE?

Laguerre's method could diverge in complex domain, however it is extremely rare. If someone decides not to use this method in favour of another globally convergent method, it is probably just because Laguerre's method does not have much solid theory on complex roots case.

3. 3. Algorithm



RESULTS OF APPROXIMATING USING LAGUERRE'S METHOD



	ITERATION	ARGUMENT	FUNCTION VALUE
FIRST ROOT **-1.9713**	1	-1.9713	-1.1142e-05
	2	**-1.9713**	2.6645e-15
SECOND ROOT **0.1276-0.7090i**	1	0.13345-0.70944i	-0.0054337+0.055193i
	2	**0.12764-0.70904i**	1.2575e-07+3.2437e-07i
THIRD ROOT **0.1276+0.7090i**	1	0.13345+0.70944i	-0.0054337-0.055193i
	2	**0.12764+0.70904i**	1.2575e-07-3.2437e-07i
FOURTH ROOT **1.4660**	1	1.463	-0.048495
	2	**1.466**	3.1372e-08

COMPARISON OF LAGUERRE'S AND MM2 METHODS

	MM2	LAGUERRE
First root	-1.9713	-1.9713
Iterations needed	3	2
Second root	0.12765-0.70903i	0.12764-0.70904i
Iterations needed	4	2
Third root	0.12765+0.70903i	0.12764+0.70904i
Iterations needed	4	2
Fourth root	1.4663	1.466
Iterations needed	3	2
Precision used	10^{-9}	10^{-9}

3. 4. Conclusions

The hypothesis drawn in the beginning of this task has been confirmed by the results. Laguerre's method obtained the right solutions right away.

The comparison of MM2 and Laguerre's methods showed that the results obtained were the same and in the case of Laguerre's method – quicker. It has to be stressed that initial points for comparison were the same.
It also obtained the values of complex roots quicker than MM2 method which in itself is tailored for finding them.
These observations show that the Laguerre's method is very reliable, especially for finding real roots.

LAGUERRE'S METHOD
ADVANTAGES:
- globally convergent
- fastest out of considered methods when close to a single root

DISADVANTAGES:
- not much solid theory about complex roots case
- most information about the function needed

SOURCE CODE

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

iterations = 0;

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

for i=1:imax
    x2 = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0));
    x0 = x1;
    x1 = x2;
    root = x2;
    iterations = iterations + 1;

    if abs(x1 - x0) < tolerance
        break
    end
end
if i >= imax
    disp("Maximal number of iterations reached")
end

plot(root, f(root), 'bx')
legend('left subinterval limit', 'right subinterval limit', 'root')
legend('Location', 'best')
hold off

end
```

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

hold on
plot(x0, f(x0), 'go')
plot(x1, f(x1), 'ro')
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;

    % stop test
    if abs(f(root)) <= tolerance
       break;
    end
end
if i >= imax
    disp("Maximal number of iterations reached")
end

plot(root, f(root), 'kx')
legend('left subinterval limit', 'right subinterval limit', 'root')
legend('Location', 'best')
hold off

end
```

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

hold on
plot(x, f(x), 'go')
grid on

root = x;
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;

    % stop test
    if abs(f(root)) <= tolerance
       break;
    end
end
if i >= imax
    disp("Maximal number of iterations reached")
end

plot(root, f(root), 'kx')
legend('initial point', 'root')
legend('Location', 'best')
hold off

end
```

```
function [root, iterations] = MM1(P, 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 = polyval(P, x2);
    b = (polyval(P, x2) * (z1 + z0) * (z1 - z0) + z0 * z0 * polyval(P, x1) - z1
* z1 * polyval(P, x0))/((z0 - z1) * z0 * z1);
    a = (polyval(P, x0) - polyval(P, x2))/(z0*z0) - b/z0;
    delta = b*b - 4*a*c;

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

    d0 = abs(root -x0);
    d1 = abs(root -x1);
    d2 = abs(root -x2);

    if d0 >= d1 && d0 >= d2
        x0 = x2;
        x2 = root;
    elseif d1 >= d0 && d1 >= d2
        x1 = x2;
        x2 = root;
    elseif d2 >= d0 && d2 >= d1
        x2 = root;
    end

    iterations = iterations +1;
end
if iterations >= imax
    disp("Maximal number of iterations reached")
end

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

    % choosing root with smaller absolute value
    if abs(root1) <= abs(root2)
        zmin = root1;
    else
        zmin = root2;
    end

    xk = zmin + xk;
    iterations = iterations + 1;
end
if iterations >= imax
    disp("Maximal number of iterations reached")
end
end
```

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

iterations = 0;

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

while e > tolerance && iterations < imax

    e = polyval(P, xk);

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

    % choosing largest absolute value for the denominator
    if abs(polyval(dP, xk) + expFromDenominator) > abs(polyval(dP, xk) -
expFromDenominator)
        denominator = polyval(dP, xk) + expFromDenominator;
    else
        denominator = polyval(dP, xk) - expFromDenominator;
    end

    % formula for Laguerre's method
    xk = xk - (n * polyval(P, xk) / denominator);
    iterations = iterations + 1;
end
if iterations >= imax
    disp("Maximal number of iterations reached")
end
root = xk;

end
```