Numerical Methods

Project B (no. 31)

Report

Author: Sotaro Suzuki

317340

TABLE OF CONTENTS

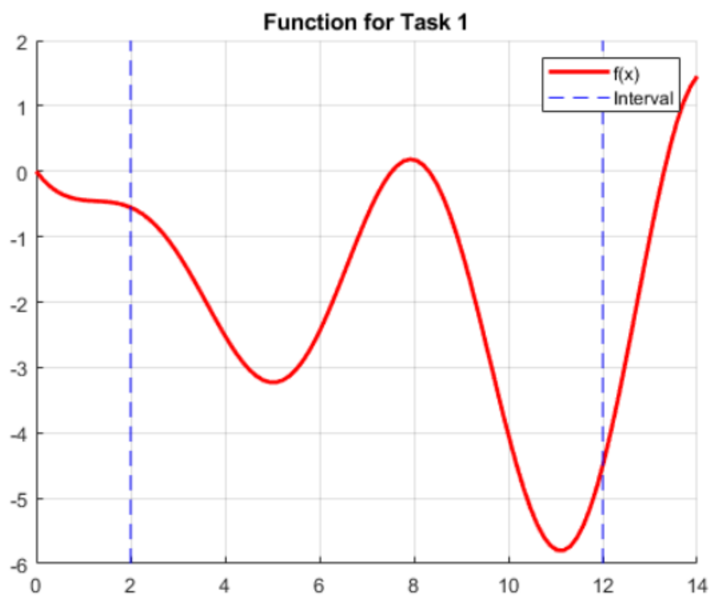
TASK 1 SECANT AND NEWTON'S METHOD

1.1 Description of the task

The aim of this task is to create a MATLAB program calculating roots of a function using the Secant and the Newton's method. The function given in the task is

$$f(x) = 0{,}3x \times \sin(x) - \ln(x + 1)$$

And the interval to find the roots in is $x \in [2, 12]$



As we can see, there are two roots in the interval, both around the point *x* = *8*, and a few outside the interval. As both of algorithms we're going to use can find only one root, we will have to subdivide our interval.

1.2 Theory

In this task we are finding *roots* of a function, that is solving a nonlinear equation of the form $f(x) = 0$. To find a root, it is crucial to determine an interval where it is located - it is called *root bracketing*. In the case of this task root bracketing will be performed with user interaction – which means, I will first plot the graph and then visually estimate the intervals in which roots are located.

Having found the interval, we initiate our chosen *iterative method*. *Iterative* means that we firstly approximate the root as some value x_0, for example as the middle or end point of the interval, and then improve this estimation with every iteration, to finally arrive at x_n, for which $f(x_n) = 0$. (It is worth noting, that due to numerical errors, we won't always find the exact value of x_n, and the value of $f(x_n)$ might differ from 0.)

Most iterative methods for nonlinear problems are only *locally convergent*, that is, their convergence depends on the starting point (first approximation) of iteration. That means, that for some initial estimations of x_0 the algorithm will not find a root.

1.3 Algorithms used..

The first algorithm used in this task is the *Secant Method*. In this method, a secant line always joins the last two obtained points. If we denote the two last obtained points as x_{n-1} and x_n, then the new point is calculated with this formula:

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

This method is only *locally convergent*, so it is crucial to first properly isolate the interval in which a root is located. It converges slightly slower than the *Newton's Method*, but is faster than for example the *Bisection Method* or *Regula Falsi* method.

The second algorithm in this task is the *Newton's Method*, also called the *tangent method*. Its principle of operation is estimating the function $f(x)$ as its first order part of expansion into a Taylor series at point x_n, that is the current estimate of the root:

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

The next point is then obtained from a root of the obtained linear function:

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

Which transforms into

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

This method, like the previous one, is also *locally convergent*. If the supplied first approximation is outside a set of attraction of any root, it may diverge. However, if convergent, it is very fast, as its convergence is quadratic. It is particularly effective when the slope is very steep (derivative is far from 0). On the other hand, if the derivative is close to 0, the function is prone to numerical errors, so it is not recommended. It also requires calculating the derivative, which can be costly in the case of complex functions.

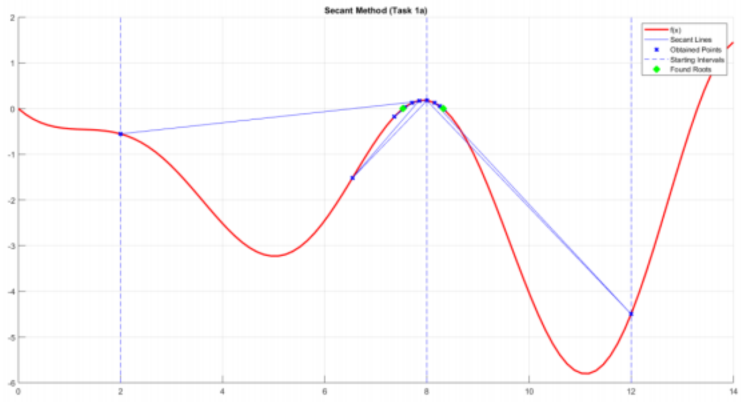
1.4 Implementation and results

1.4.1 Selecting sub-intervals

For the Secant Method, the interval was subdivided from [2, 12] to [2, 8] and [8, 12], as can be seen on the following graph. The conditions for stopping the algorithm were:

$$f(x_n) = 0 \lor x_n = x_{n-1} \lor iterations = 100$$

The results and iterations are presented and compared in 1.4.2, and the complete code can be read in the appendix.



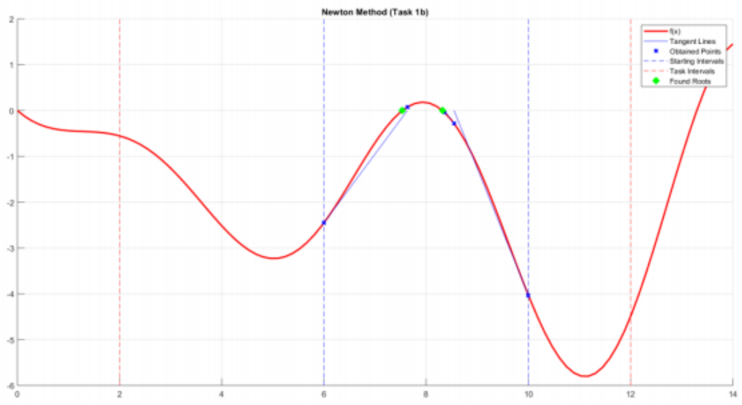
For the Newton's Method, starting points of *x* = *6* and *x* = *10* were selected, as points closer to the endpoints of [2, 12] caused the algorithm to find roots outside our interval. This algorithm needs a derivative of *f(x)*, which was calculated using the following commands in MATLAB:

```
>> syms x
>> y = 0.3*x*sin(x)-log(x+1);
>> diff(y)
```

Which gives us the result as follows:

$$f'(x) = \frac{3\sin(x)}{10} - \frac{1}{x+1} + \frac{3x\cos(x)}{10}$$

The stopping conditions in this algorithm were exactly the same as in the case of the *Secant Method.* The operation of the algorithm can be seen on the following graph, while the results are compared in the 1.4.2. Full code is available in the appendix.



1.4.2 Comparison of results

Variable	**Secant Method**	**Newton's Method**
First Root	7,53239022513267	7,53239022513267
Iterations	10	6
Second Root	8,31757258979292	8.317572589792919
Iterations	9	6

Iterations: Secant Method

i	x_i	$f(x_i)$
	2	-0,553033832572701
	8	0,177235214559897
1	6,54380889956261	-1,51485845535532
2	7,84747396275103	0,174060347836469
3	7,71311795112277	0,126186399878821
4	7,35898168008772	-0,180634478294791
5	7,56747212227817	0,0297251240088818
6	7,53801112225293	0,00492504034864183
7	7,53216047207677	-0,000202616149374446
8	7,53239165687054	1,26230936858818e-06
9	7,53239022549455	3,19058557352037e-10
10	7,53239022513267	0

i	x_i	$f(x_i)$
	8	0,177235214559897
	12	-4,49661186226310
1	8,15168251048588	0,123997572996660
2	8,25495504098348	0,0548962166695106
3	8,33699787648680	-0,0190160501420746
4	8,31589000439696	0,00160297807187604
5	8,31753098649140	3,97199027548645e-05
6	8,31757268123543	-8,73076180241128e-08
7	8,31757258978796	4,73310279858197e-12
8	8,31757258979292	-4,44089209850063e-16
9	8,31757258979292	-4,44089209850063e-16

Iterations: Newton's Method

i	x_i	$f(x_i)$
	6	-2,44885804581338
1	7,63080562489320	0,0771293346966577
2	7,51793932212994	-0,0129426582118914
3	7,53216920757560	-0,000194910556682082
4	7,53239017140866	-4,73665133782220e-08
5	7,53239022513266	-1,77635683940025e-15
6	7,53239022513267	0

i	x_i	$f(x_i)$
	10	-4,02995860546648
1	8,54583951067487	-0,281857329192520
2	8,35909656004477	-0,0417878323437413
3	8,31959292331091	-0,00193405542866421
4	8,31757788067166	-5,05166719877437e-06
5	8,31757258982940	-3,48321371745897e-11
6	8,31757258979292	-4,44089209850063e-16

1.5 Conclusions ....

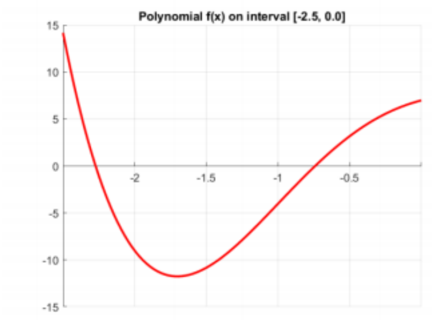
As we can see, both of the methods find the same results, however, the Newton's method takes 30% less iterations. Before calling it ultimately superior, though, it is important to point out that in this case, the problem was well conditioned for this algorithm – the slope of the function is steep and the derivative is not costly to calculate.

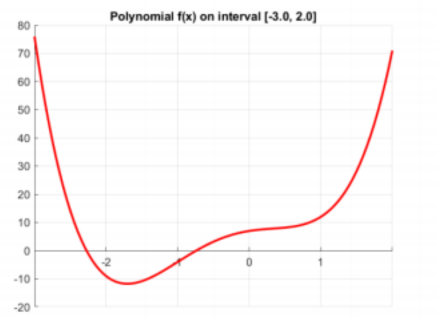
2 MULLER'S METHOD

2.1 Description of the task ....

The aim of this task is to create a MATLAB program for finding roots (real and complex) of a polynomial using the MM1 and MM2 versions of the Muller's Method. The polynomial given in the task is:

$$f(x) = 2x^4 + 3x^3 - 6x^2 + 4x + 7$$





From the graphs of this polynomial, we can notice that it has two real roots, in the intervals [-2.5, 0] and [-1, -0.5]. We also know, from the fundamental theorem of algebra, that a polynomial with real coefficients of degree *n* has exactly *n* complex roots, and if z_0 is a root then so is $\bar{z}_0$. That means that we only need to find three roots: two real and one complex, since we can obtain the fourth one by taking the conjugate.

2.2 Theory ....

We consider polynomials of the following form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

It has exactly *n* roots, which can be real or complex (in case of complex conjugate is also a root) and single or multiple. As all polynomials are continuous and differentiable nonlinear functions, it is a special case of the last task, which means the methods presented in

Task 1 also work for finding polynomial roots (though only real). This time, though, we will use algorithms designed specially for polynomials, which are faster and capable of finding complex roots.

2.3 Algorithms used

Both of the methods used in this tasks are versions of the Muller's Method – called MM1 and MM2. The basic idea of the Muller's Method is similar to the Secant Method – it interpolates a quadratic function instead of a linear one in order to approximate the root. Both MM1 and MM2 are *locally convergent* and are more effective than the Secant Method, and almost as effective as the Newton's Method – although their upside is being able to find complex roots.

2.3.1 MM1

In the MM1 method we take three points – x_0, x_1, x_2 and their polynomial values – $f(x_0)$, $f(x_1)$, $f(x_2)$. We then assume that x_2 is the approximation of the root and introduce a new variable $z = x - x_2$. Then, the interpolating parabola is defined as

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

Considering $z_0 = x_o - x_2$ and $z_1 = x_1 - x_2$ we have

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

We can then derive a following system of equations

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

Which we can transform into a form

$$b = \frac{f(x_2)(z_1 + z_0)(z_1 - z_0) + z_0^2 f(x_1) - z_1^2 f(x_0)}{(z_0 - z_1)z_0 z_1}$$

$$a = \frac{f(x_0) - f(x_2)}{z_0^2} - \frac{b}{z_0}$$

We then calculate the roots of the parabola as follows

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

For the next iterations we choose the root that has a smaller absolute value, that is:

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

where

$$z_{min} = z_+, \qquad if \ \left|b + \sqrt{b^2 - 4ac}\right| \geq \left|b - \sqrt{b^2 - 4ac}\right|$$
$$z_{min} = z_-, \qquad if \ \left|b + \sqrt{b^2 - 4ac}\right| < \left|b - \sqrt{b^2 - 4ac}\right|$$

Then, for the next iteration, we take the point x_3 and choose two points from x_0, x_1, x_2 that are the closest to x_3.

2.3.2 MM2 ....

The MM2 version, instead of using three different points, uses only one, together with the value of the polynomial and its first and second derivative at this point. It is slightly more effective numerically than MM1.

We can derive from the parabola

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

That at the point $z = 0$

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

Which leads for a root formula:

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

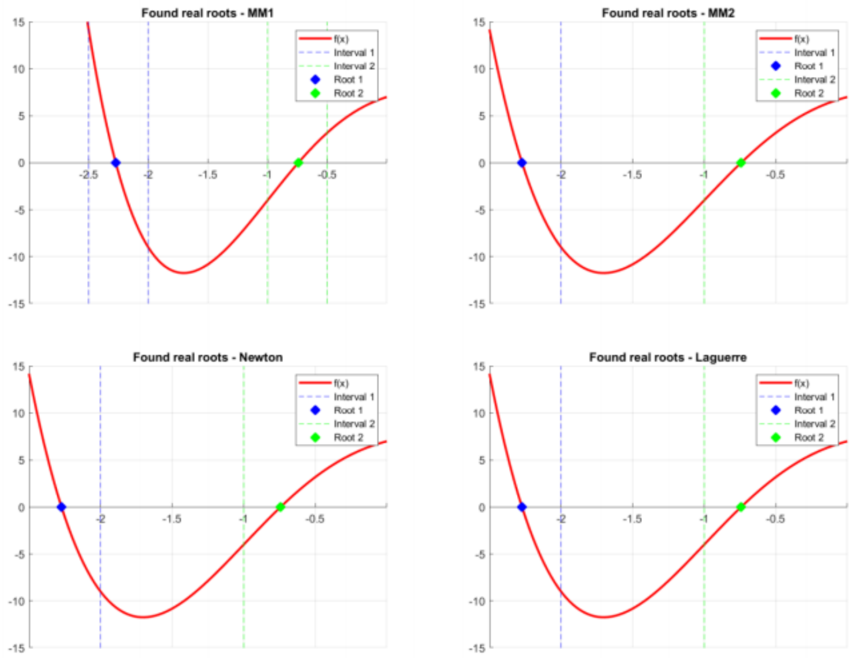
We then calculate the next approximation as

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

Where z_{min} is selected the same was as it was done in MM1.

2.4 Implementation and results

Graphs created by MM1, MM2, Newton's Method (task 1) and Laguerre's Method (task 3):



The MM2 method requires calculating the first and second derivative, which are as follows:

$$f(x) = 2x^4 + 3x^3 - 6x^2 + 4x + 7$$
$$f'(x) = 8x^3 + 9x^2 - 12x + 4$$
$$f''(x) = 24x^2 + 18x - 12$$

The results of the three algorithms are presented below:

Variable	**MM1**	**MM2**	**Newton's Method**
First Root	-2,27271005907168	-2,27271005907168	-2,27271005907168
Iterations	5	4	6
Second Root	-0,742267230272086	-0,742267230272086	-0,742267230272086
Iterations	6	4	4
Third Root	0,840821978005217 + 0,822300758043198i	0,840821978005217 + 0,822300758043198i	not possible
Iterations	10	5	
Fourth Root	0,840821978005217 - 0,822300758043198i	0,840821978005217 - 0,822300758043198i	
Iterations	conjugate	conjugate	

Comparison of iterations: MM2 vs Newton

i	**x_i – MM2**	**f(x_i) – MM2**	**x_i – Newton**	**f(x_i) – Newton**
0	-2	-9	-2	-9
1	-2,28245740735894	0,469796985784454	-2,45000000000000	10,4505187500000
2	-2,27270960463872	-2,16370274292999e-05	-2,30289971988993	1,49248403539498
3	-2,27271005907168	-7,99360577730113e-15	-2,27379622774108	0,0517865068402452
4	-2,27271005907168	-7,99360577730113e-15	-2,27271153557946	7,03014957474935e-05
5			-2,27271005907442	1,30160771050214e-10
6			-2,27271005907168	-7,99360577730113e-15

2.5 Conclusions ....

We can see that MM2 is faster and less complicated numerically than MM1. Newton's method is similar in speed to MM2, but it cannot calculate complex roots.

3 LAGUERRE'S METHOD

3.1 Description of the task ....

The aim of this task is to calculate roots of a polynomial using Laguerre's Method. The supplied polynomial is the same as in Task 2 (see 2.1).

3.2 Theory ....

As in the previous point, the theory is the same as in Task 2 (see 2.2).

3.3 Algorithm used..

The algorithm in this task is the Laguerre's Method. It is similar to MM2 (see 2.3) and is defined by this formula:

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

Where *n* denotes the order of the polynomial and the sign of the denominator is chosen as in MM1 and MM2. The formula is very similar to MM2 and can be viewed as an improved version. For real roots Laguerre's method is globally convergent, and it is regarded as one of the best methods for finding polynomial roots.

3.4 Implementation and results

The graph for this method can be seen in 2.4. The results, compared to MM2, are:

Variable	**Laguerre's**	**MM2**
First Root	-2,27271005907168	-2,27271005907168
Iterations	4	4
Second Root	-0,742267230272086	-0,742267230272086
Iterations	3	4
Third Root	0,840821978005217 + 0,822300758043198i	0,840821978005217 + 0,822300758043198i
Iterations	6	5
Fourth Root	0,840821978005217 - 0,822300758043198i	0,840821978005217 - 0,822300758043198i
Iterations	conjugate	conjugate

i	**x_i – Laguerre's**	**$f(x_i)$ – Laguerre's**	**x_i – MM2**	**$f(x_i)$ – MM2**
0	-2	-9	-2	-9
1	-2,27155469495497	-0,0549310079755641	-2,28245740735894	0,469796985784454
2	-2,27271005901715	-2,59628407661694e-09	-2,27270960463872	-2,16370274292999e-05
3	-2,27271005907168	-7,99360577730113e-15	-2,27271005907168	-7,99360577730113e-15
4	-2,27271005907168	-7,99360577730113e-15	-2,27271005907168	-7,99360577730113e-15

3.5 Conclusions

For this set of data, we can see that Laguerre's Method is slightly faster than MM2. It is also worth noting, that for real roots, it is globally convergent, which is a huge upside.

4 APPENDIX

ftask1.m
```
function [y] = ftask1(x)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
y = 0.3*x*sin(x)-log(x+1);
end
```

ftask1der.m
```
function [y] = ftask1der(x)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
y = (3*sin(x))/10 - 1/(x + 1) + (3*x*cos(x))/10;
end
```

secant1a.m
```
clear;

%   Functions

interval1 = 2;
interval2 = 12;
interval3 = 8;

x = linspace(0, 14);
y = zeros(1, 100);
for i = 1:1:100
    y(i) = ftask1(x(i));
end

limy = linspace(-6, 2);
lim1 = ones(1, 100)*interval1;
lim2 = ones(1, 100)*interval2;
lim3 = ones(1, 100)*interval3;

%   Secant method
figure(1);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
%first zero

xprev = interval1;
xnext = interval3;
xlims1(1) = xprev;
xlims1(2) = xnext;
iterations1 = 0;
```

```
while xnext ~= xprev && ftask1(xnext) ~= 0 && iterations1
< 100
    xnew = (xprev * ftask1(xnext) - xnext *
ftask1(xprev))/(ftask1(xnext) - ftask1(xprev));
    xprev = xnext;
    xnext = xnew;
    iterations1 = iterations1 + 1;
    xlims1(iterations1+2) = xnew;
end

secantzero1 = xnew;

linx = linspace(xlims1(1), xlims1(2));
linp(1) = (ftask1(xlims1(2))-
ftask1(xlims1(1)))/(xlims1(2)-xlims1(1));
linp(2) = ftask1(xlims1(1)) - linp(1)*xlims1(1);
liny = polyval(linp, linx);
plot(linx, liny, 'b', 'DisplayName', 'Secant Lines');

legend('AutoUpdate', 'off');
for i = 2:1:length(xlims1)-1
   linx = linspace(xlims1(i), xlims1(i+1));
   linp(1) = (ftask1(xlims1(i+1))-
ftask1(xlims1(i)))/(xlims1(i+1)-xlims1(i));
   linp(2) = ftask1(xlims1(i)) - linp(1)*xlims1(i);
   liny = polyval(linp, linx);
   plot(linx, liny, 'b');
end

ylims1 = zeros(1, length(xlims1));
for i = 1:1:length(xlims1)
    ylims1(i) = ftask1(xlims1(i));
end

plot(xlims1, ylims1, 'bx', 'LineWidth', 2);

%second zero

xprev = interval3;
xnext = interval2;
xlims2(1) = xprev;
xlims2(2) = xnext;
iterations2 = 0;

while xnext ~= xprev && ftask1(xnext) ~= 0 && iterations2
< 100
```

```
    xnew = (xprev * ftask1(xnext) - xnext *
ftask1(xprev))/(ftask1(xnext) - ftask1(xprev));
    xprev = xnext;
    xnext = xnew;
    iterations2 = iterations2 + 1;
    xlims2(iterations2+2) = xnew;
end

secantzero2 = xnew;

for i = 1:1:length(xlims2)-1
   linx = linspace(xlims2(i), xlims2(i+1));
   linp(1) = (ftask1(xlims2(i+1))-
ftask1(xlims2(i)))/(xlims2(i+1)-xlims2(i));
   linp(2) = ftask1(xlims2(i)) - linp(1)*xlims2(i);
   liny = polyval(linp, linx);
   plot(linx, liny, 'b');
end

ylims2 = zeros(1, length(xlims2));
for i = 1:1:length(xlims2)
    ylims2(i) = ftask1(xlims2(i));
end

legend('AutoUpdate', 'on');
plot(xlims2, ylims2, 'bx', 'LineWidth', 2, 'DisplayName',
'Obtained Points');

%   Plots
plot(lim3, limy, 'b--', 'DisplayName', 'Starting
Intervals');
plot(secantzero1, ftask1(secantzero1), 'gx', 'LineWidth',
7, 'DisplayName', 'Found Roots');
legend('AutoUpdate', 'off');
plot(lim1, limy, 'b--', lim2, limy, 'b--' );
plot(secantzero2, ftask1(secantzero2), 'gx', 'LineWidth',
7);
grid on;
box off;
legend('show');
title('Secant Method (Task 1a)');
```

newton1b.m

```
clear;

%   Functions
```

```
interval1 = 6;
interval2 = 10;
tasklim1 = 2;
tasklim2 = 12;

x = linspace(0, 14);
y = zeros(1, 100);
for i = 1:1:100
    y(i) = ftask1(x(i));
end

limy = linspace(-6, 2);
lim1 = ones(1, 100)*interval1;
lim2 = ones(1, 100)*interval2;
limt1 = ones(1, 100)*tasklim1;
limt2 = ones(1, 100)*tasklim2;

%   Newton method
figure(1);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
%first zero

xprev = interval1;
xnext = xprev - ftask1(xprev)/ftask1der(xprev);
xlims1(1) = xprev;
xlims1(2) = xnext;
iterations1 = 0;

while xnext ~= xprev && ftask1(xnext) ~= 0 && iterations1
< 100
    xnew = xnext - ftask1(xnext)/ftask1der(xnext);
    xprev = xnext;
    xnext = xnew;
    iterations1 = iterations1 + 1;
    xlims1(iterations1+2) = xnew;
end

newtonzero1 = xnew;

linx = linspace(xlims1(1), xlims1(2));
linp(1) = (-ftask1(xlims1(1)))/(xlims1(2)-xlims1(1));
linp(2) = ftask1(xlims1(1)) - linp(1)*xlims1(1);
liny = polyval(linp, linx);
plot(linx, liny, 'b', 'DisplayName', 'Tangent Lines');

legend('AutoUpdate', 'off');
for i = 2:1:length(xlims1)-1
```

```
   linx = linspace(xlims1(i), xlims1(i+1));
   linp(1) = (-ftask1(xlims1(i)))/(xlims1(i+1)-xlims1(i));
   linp(2) = ftask1(xlims1(i)) - linp(1)*xlims1(i);
   liny = polyval(linp, linx);
   plot(linx, liny, 'b');
end

ylims1 = zeros(1, length(xlims1));
for i = 1:1:length(xlims1)
    ylims1(i) = ftask1(xlims1(i));
end

plot(xlims1, ylims1, 'bx', 'LineWidth', 2);

%second zero

xprev = interval2;
xnext = xprev - ftask1(xprev)/ftask1der(xprev);
xlims2(1) = xprev;
xlims2(2) = xnext;
iterations2 = 0;

while xnext ~= xprev && ftask1(xnext) ~= 0 && iterations2
< 100
    xnew = xnext - ftask1(xnext)/ftask1der(xnext);
    xprev = xnext;
    xnext = xnew;
    iterations2 = iterations2 + 1;
    xlims2(iterations2+2) = xnew;
end

newtonzero2 = xnew;

for i = 1:1:length(xlims2)-1
   linx = linspace(xlims2(i), xlims2(i+1));
   linp(1) = (-ftask1(xlims2(i)))/(xlims2(i+1)-xlims2(i));
   linp(2) = ftask1(xlims2(i)) - linp(1)*xlims2(i);
   liny = polyval(linp, linx);
   plot(linx, liny, 'b');
end

ylims2 = zeros(1, length(xlims2));
for i = 1:1:length(xlims2)
    ylims2(i) = ftask1(xlims2(i));
end

legend('AutoUpdate', 'on');
```

```
plot(xlims2, ylims2, 'bx', 'LineWidth', 2, 'DisplayName',
'Obtained Points');

%   Plots
plot(lim1, limy, 'b--', 'DisplayName', 'Starting
Intervals');
plot(limt1, limy, 'r--', 'DisplayName', 'Task Intervals');
plot(newtonzero1, ftask1(newtonzero1), 'gx', 'LineWidth',
7, 'DisplayName', 'Found Roots');
legend('AutoUpdate', 'off');
plot(lim2, limy, 'b--' );
plot(limt2, limy, 'r--' );
plot(newtonzero2, ftask1(newtonzero2), 'gx', 'LineWidth',
7);
grid on;
box off;
legend('show');
title('Newton Method (Task 1b)');
```

polyder.m

```
function [derivative] = polyder(polynomial)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

degree = length(polynomial) - 1;
derivative = zeros(1, degree);

for i = 1:1:degree
    derivative(i) = polynomial(i) * (degree - i + 1);
end
```

mm1.m

```
clear;

p = [3 4 -6 4 7];
r = roots(p);

interval1 = -2.6;
interval2 = 0;

x = linspace(interval1,interval2);
y = polyval(p, x);
figure(1)
ylim([-15 15]);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
ax = gca;
```

```
ax.XAxisLocation = 'origin';
title('Found real roots - MM1');
grid on;
box off;

limy = linspace(-15,15);
lim11 = ones(1,100)*-2.5;
lim12 = ones(1,100)*-2;
plot(lim11, limy, 'b--', 'DisplayName', 'Interval 1');
legend('AutoUpdate', 'off');
plot(lim12, limy, 'b--');

lim21 = ones(1,100)*-1;
lim22 = ones(1,100)*-0.5;
legend('AutoUpdate', 'on');
plot(lim21, limy, 'g--', 'DisplayName', 'Interval 2');
legend('AutoUpdate', 'off');
plot(lim22, limy, 'g--');

x0 = -2.5;
x1 = -2;
x2 = (x0+x1)/2;

iterations1 = 0;

while iterations1 < 100 && polyval(p, x2) ~= 0 && x0 ~= x1
&& x1 ~= x2 && x0 ~= x2
    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))
        xnew = x2 - (2*c)/(b + sqrt(delta));
    else
        xnew = x2 - (2*c)/(b - sqrt(delta));
    end

    d0 = abs(xnew - x0);
    d1 = abs(xnew - x1);
    d2 = abs(xnew - x2);
    if d0 >= d1 && d0 >= d2
        x0 = x2;
        x2 = xnew;
    elseif d1 >= d0 && d1 >= d2
        x1 = x2;
        x2 = xnew;
    elseif d2 >= d0 && d2 >= d1
        x2 = xnew;
    end

    iterations1 = iterations1 +1;
end

root1 = xnew;

legend('AutoUpdate', 'on');
plot(root1, polyval(p, root1), 'bx', 'LineWidth', 7,
'DisplayName', 'Root 1');
legend('AutoUpdate', 'off');

% second

x0 = -0.5;
x1 = -1;
x2 = (x0+x1)/2;

iterations2 = 0;

while iterations2 < 100 && polyval(p, x2) ~= 0 && x0 ~= x1
&& x1 ~= x2 && x0 ~= x2
    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))
        xnew = x2 - (2*c)/(b + sqrt(delta));
    else
        xnew = x2 - (2*c)/(b - sqrt(delta));
    end

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

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

    iterations2 = iterations2 +1;
end

root2 = xnew;

legend('AutoUpdate', 'on');
plot(root2, polyval(p, root2), 'gx', 'LineWidth', 7,
'DisplayName', 'Root 2');
legend('AutoUpdate', 'off');

saveas(1, "./plots/mm1.fig");
saveas(1, "./plots/mm1.png");
```

mm2.m

```
clear;

p = [3 4 -6 4 7];
pd = polyder(p);
pdd = polyder(pd);

x = -2;
xp = 0;
iterations1 = 0;
xpoints(1) = x;

while iterations1 < 100 && polyval(p, x) ~= 0 && x ~= xp
    delta = polyval(pd, x)^2 - 2*polyval(p,
x)*polyval(pdd, x);
    if abs(polyval(pd, x) + sqrt(delta)) >=
abs(polyval(pd, x) - sqrt(delta))
        xnew = x - (2*polyval(p, x))/(polyval(pd, x) +
sqrt(delta));
    else
        xnew = x - (2*polyval(p, x))/(polyval(pd, x) -
sqrt(delta));
```

```
    end

    xp = x;
    x = xnew;
    xpoints(iterations1+2) = x;
    iterations1 = iterations1 + 1;
end

root1 = xnew;
ypoints = polyval(p, xpoints);

x = -1;
xp = 0;
iterations2 = 0;

while iterations2 < 100 && abs(polyval(p, x)) >= 10e-16 &&
x ~= xp
    delta = polyval(pd, x)^2 - 2*polyval(p,
x)*polyval(pdd, x);
    if abs(polyval(pd, x) + sqrt(delta)) >=
abs(polyval(pd, x) - sqrt(delta))
        xnew = x - (2*polyval(p, x))/(polyval(pd, x) +
sqrt(delta));
    else
        xnew = x - (2*polyval(p, x))/(polyval(pd, x) -
sqrt(delta));
    end

    xp = x;
    x = xnew;
    iterations2 = iterations2 + 1;
end

root2 = xnew;

y1 = polyval(p, root1);
y2 = polyval(p, root2);

interval1 = -2.5;
interval2 = 0;

figure(1)
x = linspace(interval1,interval2);
y = polyval(p, x);
figure(1)
ylim([-15 15]);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
```

```
ax = gca;
ax.XAxisLocation = 'origin';
title('Found real roots - MM2');
grid on;
box off;

limy = linspace(-15,15);
lim2 = ones(1,100)*-1;
lim1 = ones(1,100)*-2;

plot(lim1, limy, 'b--', 'DisplayName', 'Interval 1');
plot(root1, polyval(p, root1), 'bx', 'LineWidth', 7,
'DisplayName', 'Root 1');
plot(lim2, limy, 'g--', 'DisplayName', 'Interval 2');
plot(root2, polyval(p, root2), 'gx', 'LineWidth', 7,
'DisplayName', 'Root 2');

legend('show');

saveas(1, "./plots/mm2.fig");
saveas(1, "./plots/mm2.png");
```

polynewton.m

```
clear;

p = [3 4 -6 4 7];
pd = polyder(p);

xprev = -2;
xnext = xprev - polyval(p, xprev)/polyval(pd, xprev);
iterations1 = 0;
xpoints(1) = xprev;
xpoints(2) = xnext;

while xnext ~= xprev && polyval(p, xnext) ~= 0 &&
iterations1 < 100
    xnew = xnext - polyval(p, xnext)/polyval(pd, xnext);
    xprev = xnext;
    xnext = xnew;

    xpoints(iterations1 + 3) = xnext;
    iterations1 = iterations1 + 1;
end

root1 = xnext;
ypoints = polyval(p, xpoints);

xprev = -1;
```

```
xnext = xprev - polyval(p, xprev)/polyval(pd, xprev);
iterations2 = 0;

while xnext ~= xprev && abs(polyval(p, xnext)) >= 10e-16
&& iterations2 < 100
    xnew = xnext - polyval(p, xnext)/polyval(pd, xnext);
    xprev = xnext;
    xnext = xnew;
    iterations2 = iterations2 + 1;
end

root2 = xnext;

interval1 = -2.5;
interval2 = 0;

figure(1)
x = linspace(interval1,interval2);
y = polyval(p, x);
figure(1)
ylim([-15 15]);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
ax = gca;
ax.XAxisLocation = 'origin';
title('Found real roots - Newton');
grid on;
box off;

limy = linspace(-15,15);
lim2 = ones(1,100)*-1;
lim1 = ones(1,100)*-2;

plot(lim1, limy, 'b--', 'DisplayName', 'Interval 1');
plot(root1, polyval(p, root1), 'bx', 'LineWidth', 7,
'DisplayName', 'Root 1');
plot(lim2, limy, 'g--', 'DisplayName', 'Interval 2');
plot(root2, polyval(p, root2), 'gx', 'LineWidth', 7,
'DisplayName', 'Root 2');

legend('show');

saveas(1, "./plots/new.fig");
saveas(1, "./plots/new.png");
```

laguerre.m

```
clear;
```

```
p = [3 4 -6 4 7];
pd = polyder(p);
pdd = polyder(pd);
n = length(p) - 1;

x = -2;
xp = 0;
iterations1 = 0;
xpoints(1) = x;

while iterations1 < 100 && polyval(p, x) ~= 0 && x ~= xp
    delta = (n-1)*((n-1)*polyval(pd, x)^2 - n*polyval(p,
x)*polyval(pdd, x));
    if abs(polyval(pd, x) + sqrt(delta)) >=
abs(polyval(pd, x) - sqrt(delta))
        xnew = x - (n*polyval(p, x))/(polyval(pd, x) +
sqrt(delta));
    else
        xnew = x - (n*polyval(p, x))/(polyval(pd, x) -
sqrt(delta));
    end

    xp = x;
    x = xnew;
    xpoints(iterations1 + 2) = x;
    iterations1 = iterations1 + 1;
end

root1 = x;
ypoints = polyval(p, xpoints);

x = -1;
xp = 0;
iterations2 = 0;

while iterations2 < 100 && abs(polyval(p, x)) >= 10e-16 &&
x ~= xp
    delta = (n-1)*((n-1)*polyval(pd, x)^2 - n*polyval(p,
x)*polyval(pdd, x));
    if abs(polyval(pd, x) + sqrt(delta)) >=
abs(polyval(pd, x) - sqrt(delta))
        xnew = x - (n*polyval(p, x))/(polyval(pd, x) +
sqrt(delta));
    else
        xnew = x - (n*polyval(p, x))/(polyval(pd, x) -
sqrt(delta));
    end
    xp = x;
    x = xnew;
    iterations2 = iterations2 + 1;
end

root2 = x;

y1 = polyval(p, root1);
y2 = polyval(p, root2);

interval1 = -2.5;
interval2 = 0;

figure(1)
x = linspace(interval1,interval2);
y = polyval(p, x);
figure(1)
ylim([-15 15]);
hold on;
plot(x, y, 'r', 'LineWidth', 2, 'DisplayName', 'f(x)');
ax = gca;
ax.XAxisLocation = 'origin';
title('Found real roots - Laguerre');
grid on;
box off;

limy = linspace(-15,15);
lim2 = ones(1,100)*-1;
lim1 = ones(1,100)*-2;

plot(lim1, limy, 'b--', 'DisplayName', 'Interval 1');
plot(root1, polyval(p, root1), 'bx', 'LineWidth', 7,
'DisplayName', 'Root 1');
plot(lim2, limy, 'g--', 'DisplayName', 'Interval 2');
plot(root2, polyval(p, root2), 'gx', 'LineWidth', 7,
'DisplayName', 'Root 2');

legend('show');

saveas(1, "./plots/lagurr.fig");
saveas(1, "./plots/lagurr.png");
```