ENUME

PROJECT B NO.26

KORKMAZ BARAN (302809)

Table of contents

Task 1

In this task, roots of the function $\mathbf{1.2sinx + 2\ln(x+2) - 5}$ will be calculated by **False Position Method** and **Newton's Method** in the interval **[2, 12].** According to the results, a comparison between methods will be made.

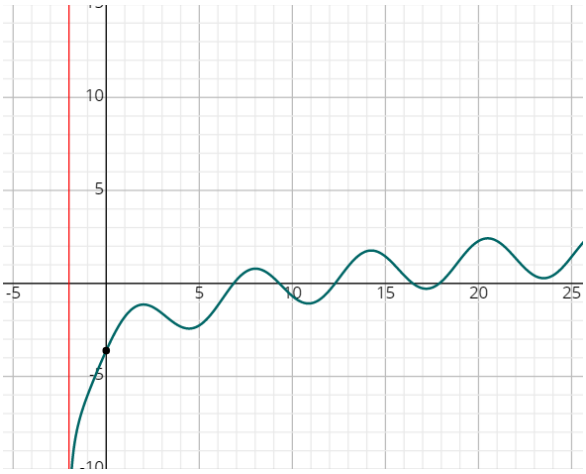
Theory

Solutions of nonlinear equations using iterative methods are called **zeros**, or **roots of the function**. To find a root with an iterative method, an interval which these roots located must be identified beforehand. This procedure is called **root bracketing** and can be performed in an interactive user-computer process. Once root bracketing is done, an approximate function f(x) is drawn and isolation intervals of all roots of the function is selected by the user.

If the roots wanted to be found without user interference, then we must check if a function **f(x)** is continuous on a closed interval $[\boldsymbol{a, b}]$ and $\boldsymbol{f(a) * f(b) < 0}$. If both cases are satisfied, then we can say that at least one root of $\boldsymbol{f(x)}$ is located within $[\boldsymbol{a, b}]$**.** However, if it does not include the root, interval can be expanded.

For our task, the function is $\mathbf{1.2sinx + 2\ln(x+2) - 5}$ and interval is **[2, 12].** Checking the conditions:

1. $\boldsymbol{f(12) = 0.5276\ f(2) = -1.1362}$ then, $\boldsymbol{f(a) * f(b) < 0}$
2. We can see from the graph of the function, it is **continuous** in the interval **[2, 12].**



False Position Method

The False Position Method is an iterative method for estimating the roots of a polynomial **f(x)**. First, we define a point c as:

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

If f(c) is equal to 0 then then it is the root. However if f(c) and f(a) have opposite signs the we determine that root lies on the interval [a,c] and step variable is b-c and b = c.

If else f(c) and f(b) have opposite signs, then the root is on the interval [c, b] and step variable is b-c and b = c.

Newton's Method

Newton's Method operates by approximation of function f(x) by the first order of its expansion into Taylor series at current point x_n

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

The next point x_{n+1} obtained by linear function:

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

This leads to iteration formula:

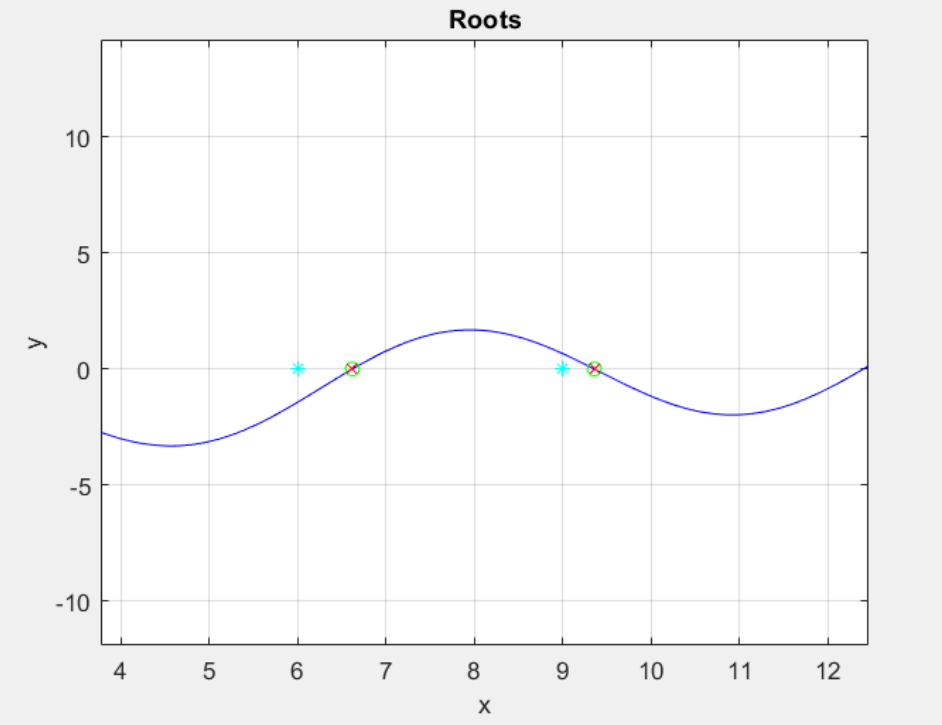
$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$$

Newton's method is **locally convergent**. When it converges, it is fast since its convergence is quadratic. Method is very effective when the function derivative is far from 0 but when it gets close to 0 then numerical errors arises.

Results

Estimated Roots found with estimation function are **6** and **9.** These values used for initial interval.

Initial Points = [6 , 9]



Roots = 6.619014036059766 and 9.357986557021809

Cyan dots represent the **initial points**, red crosses and green circles represent the roots found with **Newton's Method** and with **False Position Method,** respectively.

```
Newton's Method
1    6.630037
2    6.618994
3    6.619014
4    6.619014
Iterations:
     4

Root:
   6.619014036059766

1    9.381876
2    9.358005
3    9.357987
4    9.357987
Iterations:
     4

Root:
   9.357986557021810
```

```
False Position Method
1    6.648448
2    6.620269
3    6.619064
4    6.619016
5    6.619014
6    6.619014
7    6.619014
8    6.619014
9    6.619014
10   6.619014
11   6.619014
Iterations:
    11

Root:
   6.619014036059766

1    9.360570
2    9.357886
3    9.357987
4    9.357987
Iterations:
     4

Root:
   9.357986557021809
```

Conclusions

We can observe from the results that while both methods found the roots, Newton's Method found the first root faster and more efficiently. This is because False Position Method checks all intervals before reaching the error limit.

Task 2

In this task, I will find the roots of the polynomial $-2x^4 + 5x^3 + 5x^2 + 2x + 1$ using **Muller's Theory** implementing both the **MM1** and **MM2** versions. Then results of **MM1**, **MM2** and **Newton' Method** will be compared.

Theory

Muller's Method

Muller function approximates the root of a polynomial by taking the root of an approximate quadratic function.

There are two versions, **MM1** and **MM2**. MM1 takes **3** points from the polynomial f(x). MM2 takes one point, calculates the first and second derivative to get the second and third points.

a) MM1

A parabola is constructed which passes through **f(x0), f(x1)** and **f(x2).** The two roots of this parabola are calculated, and the root with the smaller absolute value is taken, together with the two closest points from the selection of **f(x0), f(x1)** and **f(x2).**

The three points are defined as:

$$z = x - x^2$$

$$z_0 = x_0 - x_2$$

$$z_1 = x_1 - x_2$$

The interpolating parabola defined in the variable z is considered,

$$y(z) = az_2 + \mathbf{bz} + \mathbf{c}$$

Considering the three given points, we have:

$$az_0^2 + bz_0 + c = y(z_0) = \mathbf{f}(x_0)$$

$$az_1^2 + bz_1 + c = y(z_1) = \mathbf{f}(x_1)$$

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

b) MM2

This method uses the original, first and second derivative of the function at the given point.

It is slightly more expensive to calculate values of polynomial at three points, verses to calculate the values of polynomial, 1st derivative and 2nd derivatives at one point only.

Using the same definition of the parabola,

$$y(z) = az_2 + \mathbf{bz} + \mathbf{c}$$

$$z = x - x_k$$

at the point **z = 0**:

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

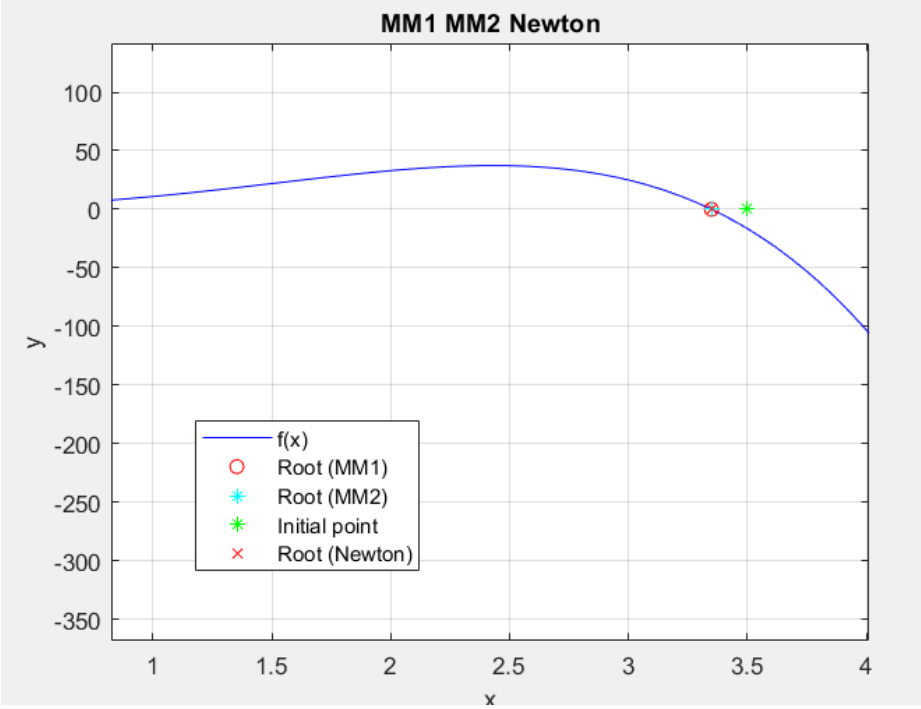
$$y''^{(0)} = 2a = f''(x_k)$$

which leads to this formula for the roots:

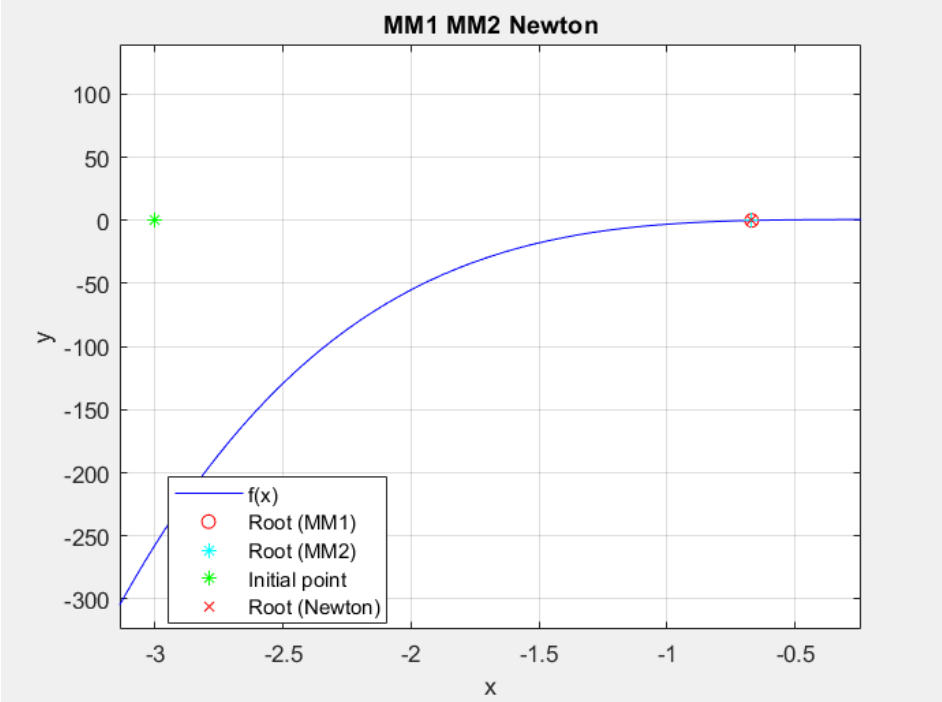
$$Z_{+,-} = \frac{-f(x_k)}{f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}$$

Results

Initial Point = 3.5 Real root = 3.34897



Initial Point = -3 Real root = - 0.669473



MM1 Method Results

```
MM1 Method
initial point = [2.5,3,3.5]
Number of iterations: 5
Iterations
   3.362814767087420
   3.349407766503623
   3.348971270878388
   3.348971152643224
   3.348971152643363

Root =
   3.348971152643363


MM1 Method
initial point = [-1,-2,-3]
Number of iterations: 12
Iterations
  -2.231930648548500
  -1.795353676003239
  -1.438208267186043
  -1.162717170980132
  -0.955356706679974
  -0.807928810518433
  -0.716571608735954
  -0.676798775479943
  -0.669648330081252
  -0.669472863938404
  -0.669472854180458
  -0.669472854180487

Root =
  -0.669472854180487
```

```
MM1 Method
initial point = [0-1.5i,0-1i,0-0.5i]
Number of iterations: 7
Iterations
 -0.065112704835693 - 0.415937164183935i
 -0.113907400500695 - 0.463750549524321i
 -0.089650942840653 - 0.461191919685775i
 -0.089757553899117 - 0.463659243074044i
 -0.089749153498421 - 0.463633208082598i
 -0.089749149231438 - 0.463633208095319i
 -0.089749149231438 - 0.463633208095319i

Root =
 -0.089749149231438 - 0.463633208095319i


MM1 Method
initial point = [0+0.5i,0+1i,0+1.5i]
Number of iterations: 10
Iterations
 -0.026535385037672 + 1.071723580859060i
 -0.049579527070339 + 0.820865704395742i
 -0.075693311647139 + 0.638222254996259i
 -0.091847607170833 + 0.525855347482116i
 -0.093602170796860 + 0.474278708017865i
 -0.090148087503476 + 0.463811519922944i
 -0.089749115080303 + 0.463632625248743i
 -0.089749149235864 + 0.463633208094837i
 -0.089749149231438 + 0.463633208095319i
 -0.089749149231438 + 0.463633208095319i

Root =
 -0.089749149231438 + 0.463633208095319i
```

MM2 Method Results

```
MM2 Method
initial point = 3.5
Number of iterations: 3
Iterations
   3.348149302465597
   3.348971152768350
   3.348971152643363

Root =
   3.348971152643363


MM2 Method
initial point = -3
Number of iterations: 7
Iterations
 -1.908783783783784 + 0.738737880410400i
 -1.410902521797849 + 0.003772997875808i
 -0.909997588489656 - 0.303265318909419i
 -0.749992471059090 - 0.014233783977020i
 -0.668275640619203 + 0.000739207704582i
 -0.669472853606977 - 0.000000006483951i
 -0.669472854180487 - 0.000000000000000i

Root =
 -0.669472854180487 - 0.000000000000000i
```

```
MM2 Method
initial point = 0-0.5i
Number of iterations: 3
Iterations
 -0.089665055336282 - 0.464843242290502i
 -0.089749148611764 - 0.463633205624232i
 -0.089749149231438 - 0.463633208095319i

Root =
 -0.089749149231438 - 0.463633208095319i
```

```
MM2 Method
initial point = 0+1.5i
Number of iterations: 5
Iterations
 -0.401737159698994 + 0.875257309054853i
 -0.061969712552161 + 0.595859238174261i
 -0.090859790169844 + 0.459721853477730i
 -0.089749114421427 + 0.463633297365805i
 -0.089749149231438 + 0.463633208095319i

Root =
 -0.089749149231438 + 0.463633208095319i
```

Newton Method Results

```
Newton's Method
Initial point = 3.5
1    3.365031
2    3.349179
3    3.348971
4    3.348971
Iterations:
     4

Root:
   3.348971152643364

Newton's Method
Initial point = -3
1    -2.204334
2    -1.627642
3    -1.217256
4    -0.936500
5    -0.763951
6    -0.686079
7    -0.670094
8    -0.669474
9    -0.669473
Iterations:
     9

Root:
  -0.669472854182414
```

Conclusions

For the initial point -3,

Method	**MM2**	**Newton**
Iterations	**7**	**9**
Root	**-0.669472854182414**	**-0.669472854182414**

For the initial point 3.5,

Method	**MM2**	**Newton**
Iterations	**3**	**4**
Root	**3.348971152643363**	**3.348971152643363**

MM2 Method found every real and imaginary root in less iterations than **MM1**, so we can conclude that it is the more efficient method than **MM1**.

Task 3

In this task, I will find the roots of the polynomial $-2x^4 + 5x^3 + 5x^2 + 2x + 1$ using **Laguerre's Method**. According to the results while using the same initial points, a comparison between **MM2 version of the Müller's method** and **Laguerre's Method** will be made.

Theory

Laguerre's Method

Another method for finding the roots of a polynomial is Laguerre's Method. The Laguerre's formula is slightly more complex since, it takes into account of the order of the polynomial. In the case of polynomials with real roots only, the Laguerre's method is convergent starting from any real initial point, thus it is **globally convergent**. The Laguerre's method is regarded as one of the most efficient methods for polynomial root finding.

Laguerre's Method defined by the formula:

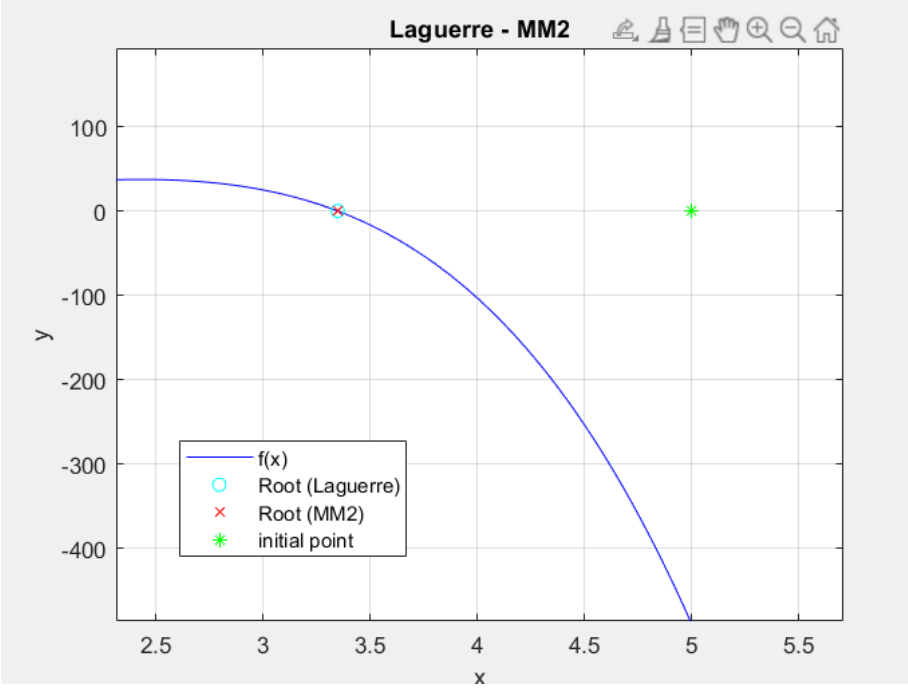
$$x_{k+1} = x_k - \frac{nf(x)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)](f'(x_k))^2 - nf(x_k)f''(x_k)}}$$

Where n denotes the order of the polynomial and the sign of the denominator is chosen so

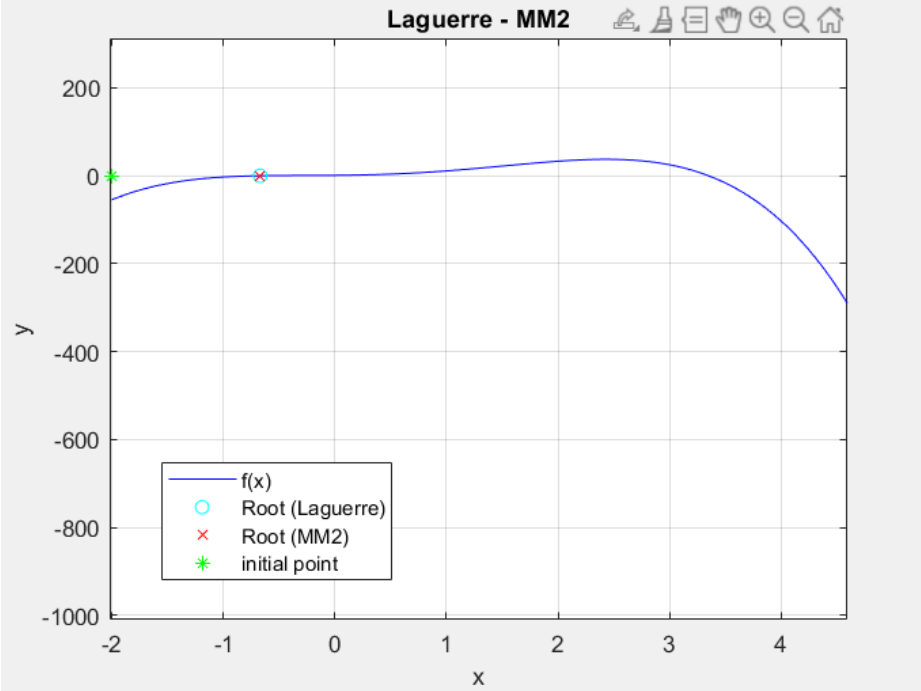
that there is a larger absolute value of the denominator.

Results

Initial Point = 5, Root = 3.34897



Initial Point = -2, Root = - 0.669473



```
Initial point = -2
Laguerre's method
Number of iterations: 4
Iterations
  -0.743013265687409
  -0.669336563679490
  -0.669472854181736
  -0.669472854180487

Root =
  -0.669472854180487

Muller's MM2 method
Number of iterations: 6
Iterations
 -1.273972602739726 + 0.475719324772201i
 -0.970683683891470 + 0.007586722572693i
 -0.646647963675905 - 0.118576205389591i
 -0.667368448816885 - 0.003105941216908i
 -0.669472733755362 + 0.000000026600251i
 -0.669472854180487 - 0.000000000000000i

Root =
 -0.669472854180487 - 0.000000000000000i
```

```
Initial point = 5
Laguerre's method
Number of iterations: 3
Iterations
   3.347653144004552
   3.348971152645730
   3.348971152643363

Root =
   3.348971152643363

Muller's MM2 method
Number of iterations: 5
Iterations
  3.697727272727273 + 0.725818859309212i
  3.426304631526288 + 0.049493448210336i
  3.348998870355031 - 0.001762908388800i
  3.348971152643940 - 0.000000000001142i
  3.348971152643363 + 0.000000000000000i

Root =
   3.348971152643363
```

```
Initial point = 0-1i
Laguerre's method
Number of iterations: 3
Iterations
 -0.126913963286785 - 0.487878960055046i
 -0.089710895455554 - 0.463643979755069i
 -0.089749149231412 - 0.463633208095312i

Root =
 -0.089749149231412 - 0.463633208095312i

Muller's MM2 method
Number of iterations: 4
Iterations
 -0.281555385156155 - 0.575529447574672i
 -0.097475216665663 - 0.476821343993981i
 -0.089744549188580 - 0.463635538167398i
 -0.089749149231438 - 0.463633208095319i

Root =
 -0.089749149231438 - 0.463633208095319i
```

```
Initial point = 0+1i
Laguerre's method
Number of iterations: 3
Iterations
 -0.126913963286785 + 0.487878960055046i
 -0.089710895455554 + 0.463643979755069i
 -0.089749149231412 + 0.463633208095312i

Root =
 -0.089749149231412 + 0.463633208095312i

Muller's MM2 method
Number of iterations: 4
Iterations
 -0.281555385156155 + 0.575529447574672i
 -0.097475216665663 + 0.476821343993981i
 -0.089744549188580 + 0.463635538167398i
 -0.089749149231438 + 0.463633208095319i

Root =
 -0.089749149231438 + 0.463633208095319i
```

Conclusions

For the initial point -2,

Method	**Laguerre**	**MM2**
Iterations	**4**	**6**
Root	**-0.669472854182414**	**-0.669472854182414**

For the initial point 5,

Method	**Laguerre**	**MM2**
Iterations	**3**	**5**
Root	**3.348971152643363**	**3.348971152643363**

For the initial point -i,

Method	**Laguerre**	**MM2**
Iterations	**3**	**4**
Root	**-0.089749149231412 - 0.463633208095312i**	**-0.089749149231438 - 0.463633208095319i**

For the initial point +i,

Method	**Laguerre**	**MM2**
Iterations	**3**	**4**
Root	**-0.089749149231412 + 0.463633208095312i**	**-0.089749149231438 + 0.463633208095319i**

We can see from the tables that both **Laguerre's** and Muller's **MM2 Methods** found the real roots identically while there is very small difference in imaginary roots. Furthermore, **Laguerre's Method** outperformed **MM2** by a slight margin when finding real roots.

Codes

Task 1.

```
% Main for Task 1

a = 2;
b = 12;
syms x real;
f = 2.1*sin(x)+2*log(x+2)-5;
R = esti(f,a,b);% Estimating the roots
n = length(R);
% Calculate estimated roots to use as initial interval
disp('Estimated roots : ');
disp(R);

falseR = zeros(n,1);
newtonR = zeros(n,1);

disp('False Position Method :');
for k=1:n
falseR(k)=falseP(f,R(k),(R(k)+1));
end

disp('Newton''s Method')
for j=1:n
newtonR(j)=newton(f,R(j),(R(j)+1));
end

f =inline(f);
j=1;
for i=-2:0.1:14
y(j)=feval(f,i);
j=j+1;
end

% Plotting the graph
figure
i=-2:0.1:14;
plot(i,y,'b');
axis ([-2 12 -12 30])
hold on
plot(R,0,'*c',falseR,0,'go',newtonR,0,'rx')
title('Roots');
xlabel('x');
ylabel('y');
grid on
```

```
function [ x ] = newton(f,x,df)
%Function for calculating roots by using Newton's Method

%Defining functions
df = diff(f);
f = inline(f);
df = inline(df);

err = 1;
max_it = 100; % iteration limit
iter = 0;

%Loop stops when error is negligible or number of
%iterations exceeds the limit.
while(iter<max_it)&&(err>10e-7)

%Newton's algorithm
x1 = x - feval(f, x)/feval(df, x);
err = abs(x1 - x);
x = x1;
iter = iter + 1;

%Iterations are printed along with current zero
fprintf('%d\t%f \n',iter,x);

end

%Printing of the results
disp('Iterations:');
disp(iter);
disp('Root:');
disp(x);

end
```

```
function [ c ]= falseP(f,a,b)
%Function for calculating roots by using False Position Method

f = inline(f);
%Finding f(a), f(b)
ya = feval(f,a);
yb = feval(f,b);
iter = 0;

%Loop stops when error is negligible or number of
%iterations exceeds the limit.
while (iter < 1000) && (b-a)>10e-12

%Find false position
c = ((a * yb) - (b * ya))/(yb - ya);
yc = feval(f,c);

%Root is found if f(c) is equal to zero
if(yc==0)
a=c;
b=c;
break

elseif(yb*yc>0)
b=c;
yb=yc;

else
a=c;
ya=yc;

end

iter = iter + 1;

%Printing iterations
fprintf('%d\t%f \n',iter,c);
end

% Evaluating C
c = (a*yb - b *ya)/(yb - ya);

% Printing iterations
disp('Iterations:');
disp(iter);
disp('Root:');
disp(c);
end
```

```
function [ R ] = esti (f ,a ,b )
% Function for estimating the roots of a function

%Defining functions
f = inline(f);
n = 1;

for i = a : b-1 %b-1 for not exceeding interval
%Evaluate y and next y
y = feval(f,i);
y1 = feval(f,i+1);

%check if the root has been found
if(y<=0 && y1 >=0)||(y>=0 && y1 <=0)
R(n) = i;
n = n+1;
end

end
end
```

Task 2

```
% Main for Task 2

format long;
p = [-2, 5, 5, 2, 1];
% Finding derrivatives
dP = polyder(p);
dP2 = polyder(dP);
f = @(x) polyval(p,x);
df = @(x) polyval(dP,x);
syms x real;
fn = -2*x^4 + 5*x^3 + 5*x^2 + 2*x + 1;

interval = [2.5 3 3.5 ; -1 -2 -3; -1.5i -1i -0.5i ; 0.5i 1i 1.5i ];
n = length(interval);

for i = 1:n

x0 = interval(i,1);
x1 = interval(i,2);
x2 = interval(i,3);

disp('MM1 Method');
disp(['initial point = [' num2str(x0), ',', num2str(x1),',',num2str(x2),']']);
muller1R = mm1(p,x0,x1,x2);

disp('MM2 Method');
disp(['initial point = ' num2str(x2)]);
muller2R = mm2(p,dP,dP2,x2);
```

```
if i<3
disp('Newton''s Method');
disp(['Initial point = ', num2str(x2)]);
newtonX = newton(fn,x2);
end

if not(isreal(muller1R))
continue;
end

% Plotting the graph
X = -4:0.01:6;
figure;
plot(real(X),polyval(p,X),'-b',muller1R,0,'ro',muller2R,0,'c*',x2,0,'g*',newtonX,0,'rx');
legend('f(x)','Root (MM1)','Root (MM2)','Initial point','Root (Newton)');
xlabel('x');
ylabel('y');
title('MM1 MM2 Newton');
grid on;

end
```

```
function x = mm2(p,dP,dP2,x)
% Function for finding roots using Muller's MM2 method
approx = [];
c = polyval(p,x);
max_it = 100;  % Limit for iterations

for i = 1:max_it
a = 0.5 * polyval(dP2,x);
b = polyval(dP,x);
sqrtdel = sqrt(b*b - 4*a*c);
x1 = -2 * c/(b + sqrtdel);
x2 = -2 * c/(b - sqrtdel);

if abs(x1) < abs(x2)
x = x + x1;
else
x = x + x2;
end

approx = [approx; x];
c = polyval(p,x);

% Plotting the graph
if abs(c) < 1e-12
disp(['Number of iterations: ',num2str(i)]);
disp('Iterations ');
disp(approx);
disp('Root = ');
disp(x);
return;

end
end
```

```
function x2 = mm1(p,x0,x1,x2)
% Function for finding roots using Muller's MM1 method
approx = [];
px0 = polyval(p,x0);
px1 = polyval(p,x1);
max_it = 100; % Limit for iterations

% MM1 Method
for i = 1:max_it
px2 = polyval(p,x2);
z1 = x1 - x0;
z2 = x2 - x1;
del1 = (px1 - px0)/z1;
del2 = (px2 - px1)/z2;
d = (del2 - del1)/(x2 - x0);
b = del2 + z2 * d;
D = sqrt(b*b + 4*px2*d);

if abs(b - D) < abs(b + D) %checking which denominator is bigger
E = b + D;
else
E = b - D;
end

z = -2 * px2 / E;
x0 = x1;
x1 = x2;
x2 = x2 + z;
approx = [approx; x2];
```

```
if abs(z) < 1e-12
disp(['Number of iterations: ',num2str(i)]);
disp('Iterations ');
disp(approx);
disp('Root = ');
disp(x2);
return;

end
px0 = px1;
px1 = px2;

end
error('Number of iterations exceeded');
```

Task 3.

```
% Main for Task 3

format long;
p = [-2, 5, 5, 2, 1];
% Finding derrivatives
dP = polyder(p);
dP2 = polyder(dP);

for x2 = [-2, 5, -1i, 1i] %initial points
disp(['Initial point = ', num2str(x2)]);

disp('Laguerre''s method');
rlaguerre = laguerre(p,dP,dP2, x2);

disp('Muller''s MM2 method');
rmuller2 = mm2(p,dP,dP2, x2);
if not(isreal(rlaguerre))
continue;
end

% Plotting the graph
X = -2:0.01:6;
figure;
plot(X,polyval(p,X),'b-',rlaguerre,0,'co',rmuller2,0,'rx',x2,0,'g*');
legend('f(x)','Root (Laguerre)','Root (MM2)','initial point');
xlabel('x');
ylabel('y');
title('Laguerre - MM2');
grid on;
end
```

```
function x = laguerre(p,dP,dP2,x)
% Function for finding roots using laguerre's method
approx = [];
max_it = 100;
n = length(p) - 1; % order of the poly

for i = 0:max_it
px = polyval(p,x);

if abs(px) < 1e-12
disp(['Number of iterations: ',num2str(i)]);
disp('Iterations');
disp(approx);
disp('Root = ');
disp(x);
return;
end

% Laguerre's Method
G = polyval(dP,x)/px;
H = G*G - polyval(dP2,x)/px;
c = sqrt((n-1)*(n*H - G*G));
if abs(G-c) > abs(G+c)
x = x - (n/(G-c));
else
x = x - (n/(G+c));
end
approx = [approx; x];
end
error('Number of iterations exceeded');
end
```