Project B13

ENUME

Mazen Ibrahim

295924

Table of Contents

Find Zeros Of Function

Theory

Solutions of nonlinear equations using iterative methods are called **zeros**, or roots of the function. To find the root, an interval must be identified where the root exists. The process of finding this interval is called **root bracketing**, and there are two methods of doing this:

1. **Interactive user-computer process** where the approximate graph of the function is drawn by the computer and the user identifies the interval(s) where a root is located

2. **Algorithmic** approach (without user input), then we check if a function f(x) is continuous on a closed interval [a, b] and f(a).f(b) < 0. If both cases are satisfied, then at least one root of f(x) is located within [a, b]. If it does not include the root, (for example if f(a).f(b) > 0), the reasonable approach is to expand this interval. The way this works is that f(a).f(b) is only less than 0 if there is a sign change between the interval, meaning it crosses the x-axis, meaning root exists in the interval.

Once the root interval is found, we can use an iterative method to find the roots values.

I was tasked to find the roots of this following function $f(x) = 3.1 - 3x - e^{-x}$ over intervals of [-5,10]

f(-5) = -130.3131591, f(10)= −26.9000454,f(0)= 2.1 since f(0) is a different sign from f(-5) and f(10) then f(0)*f(-5) < 0 and f(0)*f(10) < 0 , f(x) is continuous so we will have at least 2 roots within those intervals. We will use Bisection Method and Newton Method to find the

Bisection Method

The **bisection method** in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods. The method is also called the **interval halving** method, the **binary search method**.

Algorithm

Let [a,b] = **[a_0,b_0]** be initial interval containing a root α of function f in bisection method , at every (n-th) iteration :

1. The current interval containing the root **[a_n,b_n]**, is divided into two equal subintervals with the division point **c_n** located in the middle of the interval $c_n = \frac{a_n+b_n}{2}$ then f(c_n) is evaluated.
2. Products f(a_n)f(c_n) and f(c_n) f(b_n) are evaluated then the next inetravl is chosen as the subinterval corresponding to the product with the negative value. The endpoints of this interval is denoted a_{n+1},b_{n+1}
3. The procedure is repeated until f(x_{n+1}) ≤ δ where δ is assumed solution accuracy and solution accuracy in bisection method depends on the number of iterations performed not the accuracy of calculations of the function values. This test sometimes is imprecise when the derivative of a function has small absolute values in a neighbourhood of the root and to resolve this issue, checking the length of interval [b_n-a_n] is recommended assuming that it should be sufficiently small.

Now based on the following formula $\varepsilon_{n+1} = \frac{1}{2}\varepsilon_n$ we can see that bisection method is linearly convergent (p=1) with convergence factor (k=1/2).

Newton Method

This method is called tangent method as well. It operates by approximation of function f(x) by the first order of its expansion into Taylor series at current point X_n(current approx. of the root)

$f(x)=f(x_n)+f'(x_n)(x-x_n)$

The next point is a root of obtained linear function :

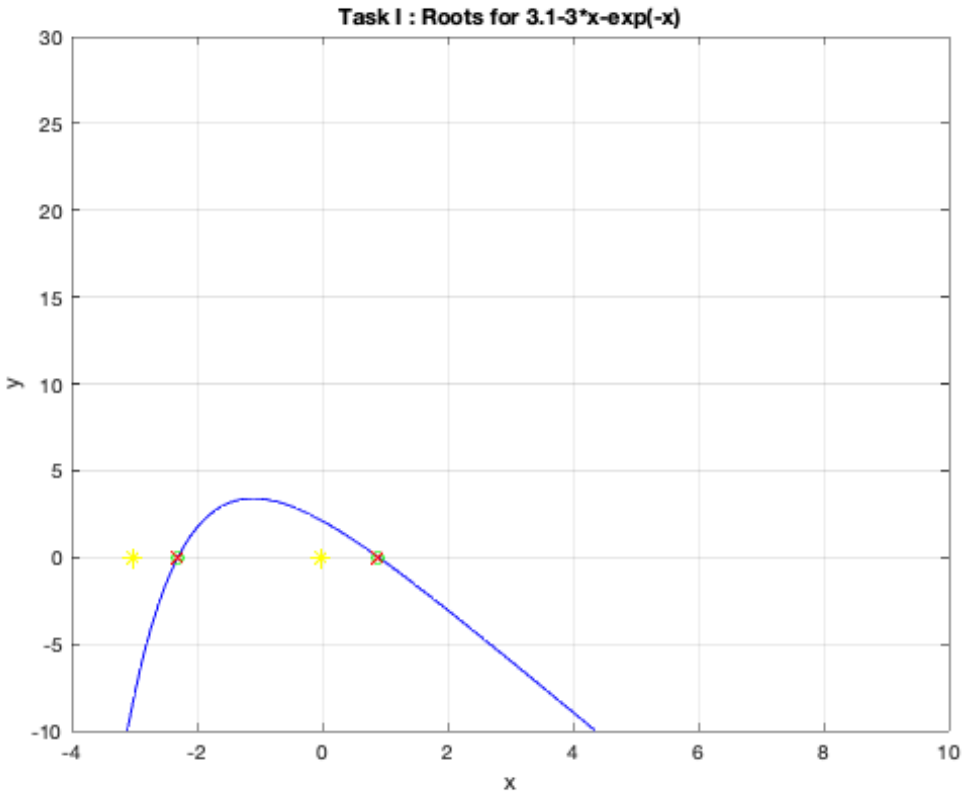
$f(x_n)+f'(x_n)(x_n+1-x_n)$

This leads to:

$X_{n+1}=x_n-f(x_n)/f'(x_n)$

Newton's method is locally convergent. When it is convergent , it converges really fast (quadratic convergence). It is very effective when the function derivative is far from 0 but when it gets close to 0 then numerical errors arises. Order of convergence p =2

Results



Where my initial points[-3,0] are in yellow and bisection method is in green circle and red cross is the newton method.

Now using my Estimation function I was able to estimate my interval points where there going to be roots and I got [-3,0].

I took those initial points and plugged them into **fzero** matlab function to get the values of the roots then compare them to my bisection and newton results. So running **fzero** at -3 = -2.303692114313962 and **fzero** at 0 = 0.897466303315499

Bisection Method :

1	-2.500000
2	-2.250000
3	-2.375000
4	-2.312500
5	-2.281250
6	-2.296875
7	-2.304688
8	-2.300781
9	-2.302734
10	-2.303711
11	-2.303223
12	-2.303467
13	-2.303589
14	-2.303650
15	-2.303680
16	-2.303696
17	-2.303688
18	-2.303692
19	-2.303694
20	-2.303693
21	-2.303692
22	-2.303692
23	-2.303692
24	-2.303692
25	-2.303692
26	-2.303692
27	-2.303692
28	-2.303692
29	-2.303692
30	-2.303692
31	-2.303692
32	-2.303692
33	-2.303692
34	-2.303692
35	-2.303692
36	-2.303692
37	-2.303692

Root:

-2.3037

Iterations:

37

1	0.500000
2	0.750000
3	0.875000
4	0.937500
5	0.906250
6	0.890625
7	0.898438
8	0.894531
9	0.896484
10	0.897461
11	0.897949
12	0.897705
13	0.897583
14	0.897522
15	0.897491
16	0.897476
17	0.897469
18	0.897465
19	0.897467
20	0.897466
21	0.897466
22	0.897466
23	0.897466
24	0.897466
25	0.897466
26	0.897466
27	0.897466
28	0.897466
29	0.897466
30	0.897466
31	0.897466
32	0.897466
33	0.897466
34	0.897466
35	0.897466
36	0.897466
37	0.897466

Root:

0.8975

Iterations:

37

Newton's Method			
1	-2.532614	1	1.050000
2	-2.335613	2	0.899084
3	-2.304402	3	0.897467
4	-2.303692	4	0.897466
5	-2.303692		
Root:		Root:	
-2.3037		0.8975	
Iterations:		Iterations:	
5		4	

Conclusion

Now comparing both results from Newton method and bisection method we can see got results close to what I got from **fzero** function but as expected bisection took way more iteration **37** while Newton found the results in way less iteration **4** and **5.** this is because bisection method checks all intervals until it reaches satisfying error

Finding Roots Of Polynomial

Theory

Muller
Approximate the root of the polynomial by taking the root of an approximate quadratic function. There are two versions, MM1 and MM2. MM1 takes three points from the polynomial f(x). MM2 takes one point, and calculates the first and second derivative to get the second and third point respectively.

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:
$z = x - x_2$
$z_0 = x_0 - x_2$
$z_1 = x_1 - x_2$

$f(x_0) = y(z_0) = az_0 + bz_0 + c$
$f(x_1) = y(z_1) = az_1 + bz_1 + c$
$f(x_2) = y(0) = c$

$x_3 = x_2 + z_{min}$

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.

$y(0) = c = f(x_k)$
$y'(0) = b = f'(x_k)$
$y''(0) = 2a = f''(x_k)$
z = -2c / (b +- sqrt(b2 - 2 c 2a))
$x_{k+1} = x_k + z_{min}$

Locally convergent. Order of convergence is 1.84. Newton's method is more effective, but only for real roots.

Results

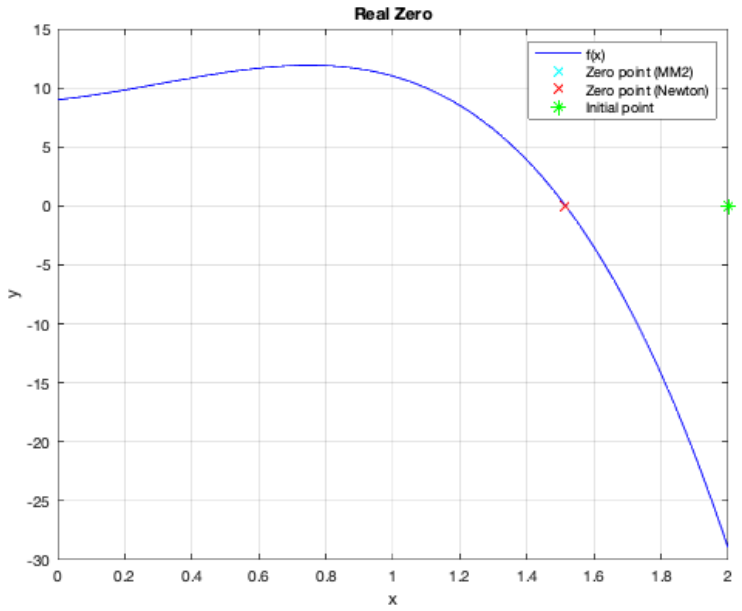


Figure 1. Root : 1.515

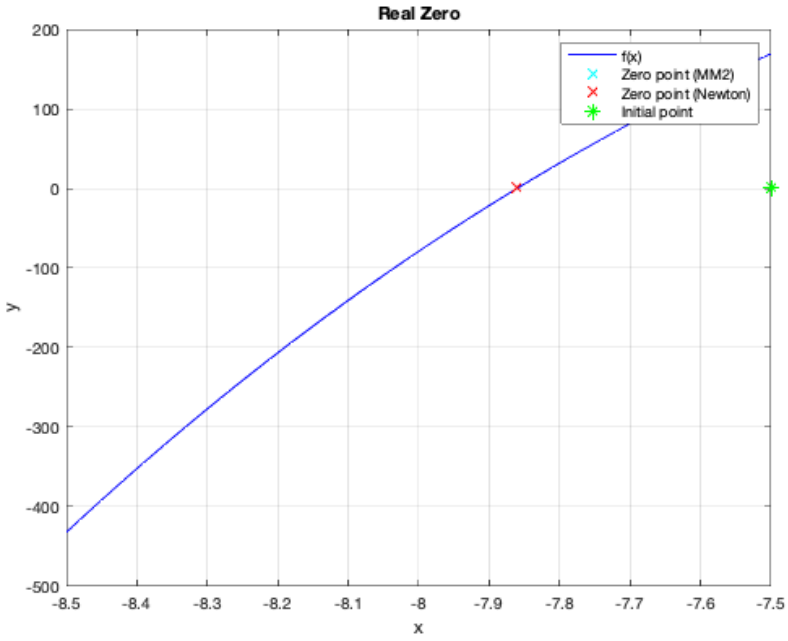


Figure 2. Root = -7.861

```
Error tolerance is 10^-12

Searching roots of polynomial p(x)

Coefficients of f(x) =

   -1   -7    7    3    9



Coefficients of f'(x) =

   -4  -21   14    3



Coefficients of f''(x) =

  -12  -42   14



Muller's MM1 Method

initial point = [-8.5,-8,-7.5]

Zero point, x =

  -7.860505196932658



f(x) = -8.8818e-15

Number of iterations: 7

Iterations

  -8.175475549103977

  -7.911885020243333

  -7.862556741021428

  -7.860506912260441

  -7.860505196926615

  -7.860505196932658

  -7.860505196932658
```

```
Muller's MM2 Method

initial point = -7.5

Zero point, x =

  -7.860505196932658



f(x) = -8.8818e-15

Number of iterations: 3

Iterations

  -7.862572861342487

  -7.860505196531060

  -7.860505196932658



Newton's Method

Initial point = -7.5

1                    -7.918831

2                    -7.861726

3                    -7.860506

4                    -7.860505

Root:

  -7.860505196932771



Iterations:

     4
```

Muller's MM1 Method

initial point = [0,1,2]

Zero point, x =

1.515070078358147

f(x) = -3.5527e-15

Number of iterations: 6

Iterations

1.584129681327890

1.526723641599846

1.515253485274370

1.515070097552550

1.515070078358134

1.515070078358147

Muller's MM2 Method

initial point = 2

Zero point, x =

1.515070078358147

f(x) = -3.5527e-15

Number of iterations: 4

Iterations

1.445078146575789

1.515186892390409

1.515070078357598

1.515070078358147

Newton's Method

Initial point = 2

1	1.658824
2	1.532660
3	1.515378
4	1.515070
5	1.515070

Root:

1.515070078358157

Iterations:

5

Muller's MM1 Method

initial point = [0,0+1i,1]

Zero point, x =

-0.327282440712744 + 0.805359201021283i

f(x) = 3.5527e-15+2.2204e-15i

Number of iterations: 41

Iterations

0.563897673344207 - 0.676194462019296i

0.744562149145638 + 0.2236921489546l7i

0.866744660496409 - 0.505012852692550i

0.604523741344415 + 0.100447907202829i

0.516832630153413 - 0.820002465793278i

0.284053514885992 - 0.315296366845843i

-0.340504644122822 + 0.138227095579296i

0.263899895336749 - 0.2996399418478001

-0.522242063510961 - 0.0891923089562671

0.128236483556812 - 0.1459544130770191

-0.677661618185150 + 0.008194730909710i

-0.123484138981585 - 0.0146417512087091

-0.995665786726069 + 0.0341310798590461

-0.467813543212239 + 0.0295864478546331

0.086464490097695 + 0.0449050129944361

-0.487980008747952 + 0.0458131597723431

0.129539818756217 + 0.0774444111369261

-0.672481822538826 + 0.0547647335589271

-0.112579716737723 + 0.077414930391994i

-0.976569197652724 + 0.083575347697307i

-0.452401635198207 + 0.081617404824109i

0.103788331911262 + 0.1256835437045111

-0.462543600145374 + 0.126258864884609i

0.149832412724963 + 0.219982813285532i

-0.607329379677357 + 0.150054190657167i

-0.055759583495565 + 0.226283453404683i

-0.841201861478171 + 0.220506881194682i

-0.341768172536979 + 0.236265641805382i

0.222017078564360 + 0.415986479613998i

-0.281738455303615 + 0.353157634214363i

0.226778425413173 + 0.754079925871257i

-0.178408534680634 + 0.494334352584038i

-0.778596362036296 + 0.597204685355327i

-0.415064079281547 + 0.621167420024586i

-0.235162563069064 + 0.771002556751313i

-0.335947547446353 + 0.791817425831711i

-0.326780375592608 + 0.805450386324306i

-0.327282669934056 + 0.805358731662547i

-0.327282440712251 + 0.805359201022564i

-0.327282440712744 + 0.805359201021283i

-0.327282440712744 + 0.805359201021283i

Muller's MM2 Method

initial point = 1

Zero point, x =

1.515070078358147

f(x) = -3.5527e-15

Number of iterations: 4

Iterations

1.568114574786861

1.515017868922086

1.515070078358197 ,1.515070078358147

Muller's MM1 Method	Iterations
initial point = [0,0-1i,0-7i]	1.504511805834134 - 4.175005845349707i
Zero point, x =	1.929467950645287 - 2.062636256897718i
-0.327282440712744 - 0.805359201021283i	1.824343482499823 - 0.694622590734530i
f(x) = 0+4.4409e-16i	1.599166784755549 - 0.061962302781401i
Number of iterations: 12	1.515211391292258 + 0.000375883923548i
Iterations	1.515070078377813 + 0.000000000010542i
-0.009889347604432 - 4.290941364679873i	1.515070078358147 - 0.000000000000000i
-0.152083926369799 - 3.414504914656272i	
-0.199660932756738 - 2.564207573988289i	
-0.201162497972679 - 1.897189387546498i	
-0.192194872366421 - 1.397467438470327i	
-0.204073689589347 - 1.050373734018006i	
-0.255877141358951 - 0.850502893844591i	
-0.319037359378851 - 0.797911328139113i	
-0.327439014763019 - 0.805456888282797i	
-0.327282505901633 - 0.805359244071004i	
-0.327282440712793 - 0.805359201021261i	
-0.327282440712744 - 0.805359201021283i	
Muller's MM2 Method	
initial point = 0-7i	
Zero point, x =	
1.515070078358147 - 0.000000000000000i	
f(x) = 8.8818e-15+1.2248e-25i	
Number of iterations: 7	

Conclusion

As we can see MM1 is very inefficient method to find the roots of a polynomial compared to MM2 when we were trying to find real and complex roots MM2 took less number of iteration than MM1 and also MM2 seems to have better accuracy.

MM2 took less iteration than Newton to find the real roots of the polynomial. Regarding the graph the reason why we can only see one method pinned on the graph is because both MM2 and Newton rely on the same point so only one of them is displayed.

fzero function was used to check if the roots we got were correct**. fzero** at [-8 -7.5] = -7.860505196932658
and **fzero** at [0 2] = 1.515070078358147. so the real roots we got using our methods in this task matches **fzero**

Laguerre's Method

Theory

Laguerre's method was designed for the same purpose as Muller's method, which is to find the roots of polynomial and its defined by the following formula :

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

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.

The Laguerre's formula is slightly more complex, it takes into account the order of the polynomial, therefore the Laguerre's method is better, in general. 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. Despite a lack of formal analysis for a complex roots case, the numerical practice has shown good numerical properties also in this case (although situations of divergence may happen). The Laguerre's method is regarded as one of the best methods for polynomial root finding.

Results

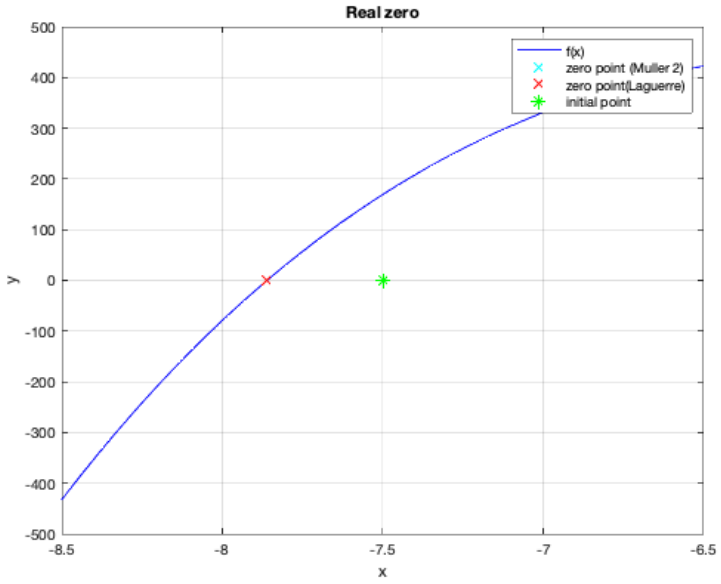


Figure 1. Root = -7.86

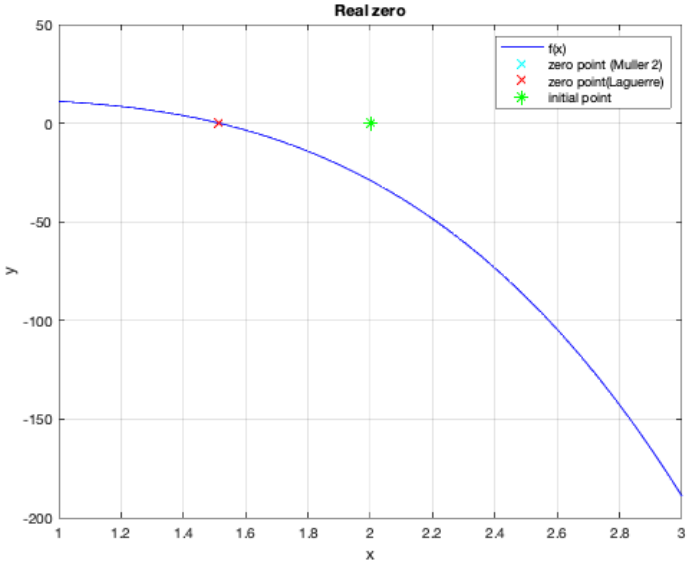


Figure 2. Root : 1.515

```
Error tolerance is 10^-12
Searching roots of polynomial p(x)
Coefficients of f(x) =
   -1   -7    7    3    9

Coefficients of f'(x) =
   -4  -21   14    3

Coefficients of f''(x) =
  -12  -42   14

Initial point = -7.5
Laguerre's method
Zero point, x =
  -7.860505196932658

f(x) = 190480052.1654
Number of iterations: 2
Iterations
  -7.860505212682339
  -7.860505196932658

Muller's MM2 method
Zero point, x =
  -7.860505196932658

f(x) = -8.8818e-15
Number of iterations: 3
Iterations
  -7.862572861342487
  -7.860505196531060
  -7.860505196932658

Initial point = 2
Laguerre's method
Zero point, x =
   1.515070078358147

f(x) = 1880125044564.659
Number of iterations: 3
Iterations
   1.516250260358106
   1.515070078359743
   1.515070078358147

Muller's MM2 method
Zero point, x =
   1.515070078358147

f(x) = -3.5527e-15
Number of iterations: 4
Iterations
   1.445078146575789
   1.515186892390409
   1.515070078357598
   1.515070078358147
```

```
Initial point = 0-1i
Laguerre's method
Zero point, x =
 -0.327282440712744 -
0.805359201021283i

f(x) = 96834690455.319-
8781207202.00406i
Number of iterations: 3


Iterations
 -0.328083705842789 - 0.807741591877772i
 -0.327282440743472 - 0.805359201024069i
 -0.327282440712744 - 0.805359201021283i

Muller's MM2 method
Zero point, x =
 -0.327282440712744 - 0.805359201021283i

f(x) = 0+1.3323e-15i
Number of iterations: 3
Iterations
 -0.317044822038044 - 0.816740165465035i
 -0.327282628007149 - 0.805360144928637i
 -0.327282440712744 - 0.805359201021283i

Initial point = 0-7i
Laguerre's method
Zero point, x =
  1.515070078358170 + 0.000000000000000i

f(x) = 10438.7851-35.8106578i
Number of iterations: 4
Iterations
  1.267034548650265 - 1.700086283177163i
  1.479369280514528 + 0.292134167300387i
  1.515357436720577 + 0.000000985697915i
  1.515070078358170 + 0.000000000000000i

Muller's MM2 method
Zero point, x =
  1.515070078358147 - 0.000000000000000i

f(x) = 8.8818e-15+1.2248e-25i
Number of iterations: 7
Iterations
  1.504511805834134 - 4.175005845349707i
  1.929467950645287 - 2.062636256897718i
  1.824343482499823 - 0.694622590734530i
  1.599166784755549 - 0.061962302781401i
  1.515211391292258 + 0.000375883923548i
  1.515070078377813 + 0.000000000010542i
  1.515070078358147 - 0.000000000000000i
```

Conclusion

As we can see from our results that Laguerre's method is on par with MM2 Method both of them have small number of iteration which makes them both efficient.

Appendix

Main Task 1:

```
a = -5;

b = 10;

syms x real;
f = 3.1-3*x-exp(-x);
R = esti(f,a,b);
n = length(R);
%using fzero function to compare results with bisection and newton
fun = @(x) 3.1-3*x-exp(-x);
int = [-3]
int2 = [0]
fzero(fun,int)
fzero(fun,int2)
%estimation of points where we have zeros
disp('Estimated roots : ');
disp(R);
bisectionr = zeros(n,1);
newtonr = zeros(n,1);
disp('Bisection Method :');
for k=1:n
    bisectionr(k)=BisectionMethod(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=-4:0.1:10
    y(j)=feval(f,i);
    j=j+1;
end
figure
i=-4:0.1:10;
plot(i,y,'b');
axis ([-4 10 -10 30])
hold on
grid on
plot(R,0,'*y')
plot(bisectionr,0,'og')
plot(newtonr,0,'xr');
title('Task I : Roots for 3.1-3*x-exp(-x)');
xlabel('x');
ylabel('y');
```

Bisection Method function :

```
function [ c ] = BisectionMethod(f, a, b)
    %Defining function
   % f = inline(f);
```

```
    %Checking f(a)
    ya = feval(f,a);
    %Checking f(b)
    yb = feval(f,b);

    iter = 0;
    while (iter < 1000) && (b-a)>10e-12
        %Bisection method
        c = (a+b)/2;
        %find f(c)
        yc = feval(f,c);
        %if f(c) = 0,root has been found.
        if(yc==0)
            a=c;
            b=c;
            break
        elseif(yb*yc>0)
            b=c;
            yb=yc;
        else
            a=c;
            ya=yc;
        end

        %Number of iterations is incremented
        iter = iter + 1;
        %Iterations are printed along with current zero
        fprintf('%d\t%f \n',iter,c);

    end
    c = (a*yb - b *ya)/(yb - ya);
    disp('Root:');
    disp(c);
    disp('Iterations:');
    disp(iter);
end
```

Estimation function :

```
%Estimation of function roots
function [ R ] = esti (f ,a ,b )
    %Defining functions
    f = inline(f);
    n = 1;

    for i = a : b
        %Evaluate y
        y = feval(f,i);
        %Evaluate next value of y
        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
```

Newton Method :

```
%Newton's Method
function [ x ] = newton(f,x,df)
    %Defining functions
    df = diff(f);
    f = inline(f);
    df = inline(df);
    %Error
    err = 1;
    %Iterations
    iter = 0;
    %Newton's loop that stops when error is negligible or number of
    %iterations become prohibitive.
    while(iter<1000)&&(err>10e-7)
        %Newton's algorithm
        x1 = x - feval(f, x)/feval(df, x);
        %Calculation of absolute error
        err = abs(x1 - x);
        %Root/zero is updated to new value
        x = x1;
        iter = iter + 1;
        %Iterations are printed along with current zero
        fprintf('%d\t%f \n',iter,x);
    %While loop finishes when error goes below 10e-12 or iterations go
    %beyond 1000
    end
    %Printing of the results
    disp('Root:');
    disp(x);
    disp('Iterations:');
    disp(iter);
end
```

Main Task 2 :

```
format long;

p = [-1, -7, 7, 3, 9];

dP = polyder(p);
dP2 = polyder(dP);

 f = @(x) polyval(p,x);
df = @(x) polyval(dP,x);

syms x real;
fn = -1*x*x*x*x - 7*x*x*x + 7*x*x + 3*x + 9;
%finding roots using matlab fzero function
fun = @(x) -x.^4-7*x.^3+7*x.^2+3*x+9
int = [-8 -7.5]
int2 = [0 2]
fzero(fun,int)
fzero(fun,int2)
disp('Error tolerance is 10^-12');
disp('Searching roots of polynomial p(x)');
disp('Coefficients of f(x) = ');
disp(p);
disp('Coefficients of f''(x) = ');
disp(dP);
disp('Coefficients of f''''(x) = ');
disp(dP2);

X = (-8):0.01:3;
```

```
figure;
plot(X,polyval(p,X));
legend('f(x)');
xlabel('x');
ylabel('y');
grid on;

start = zeros(3,4);
start(:,1) = [-8.5;-8;-7.5];
start(:,2) = [0;1;2];
start(:,3) = [0;1i;-1i*1i];
start(:,4) = [0;-1i;-7*1i];

for interval = start
    x0 = interval(1);
    x1 = interval(2);
    x2 = interval(3);

    disp('Muller''s MM1 Method');
    disp(['initial point = [' num2str(x0), ',', num2str(x1),',',num2str(x2),']']);
    x_muller1 = mm1(p,x0,x1,x2);

    disp('Muller''s MM2 Method');
    disp(['initial point = ' num2str(x2)]);
    x_muller2 = mm2(p,dP,dP2,x2);

    if not(isreal(x_muller1))
        continue;
    end

    disp('Newton''s Method');
    disp(['Initial point = ', num2str(x2)]);
    x_newton = newton(fn,x2);

    X = x0:0.01:x2;
    figure;
    plot(X,polyval(p,X),'-b',x_muller2,0,'cx',x_newton,0,'rx',x2,0,'g*');
    legend('f(x)','Zero point (MM2)','Zero point (Newton)','Initial point');
    xlabel('x');
    ylabel('y');
    title('Real Zero');
    grid on;
end
```

MM1 Method :

```
function x2 = mm1(p,x0,x1,x2)

approx = [];
px0 = polyval(p,x0);
px1 = polyval(p,x1);

for i = 1:100 %Maximum iterations = 100
    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;
    %discriminant
    D = sqrt(b*b + 4*px2*d);
```

```
    %checking which denominator is bigger
    if abs(b - D) < abs(b + D)
        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('Zero point, x = ');
        disp(x2);
        disp(['f(x) = ',num2str(polyval(p,x2))]);
        disp(['Number of iterations: ',num2str(i)]);
        disp('Iterations ');
        disp(approx);
        return;
    end
    px0 = px1;
    px1 = px2;
end
error('Number of iterations exceeded');
```

MM2 Method :

```
function x = mm2(p,dP,dP2,x)

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

for i = 1:100 %Maximum iterations = 100
    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);

    if abs(c) < 1e-12
        disp('Zero point, x = ');
        disp(x);
        disp(['f(x) = ',num2str(c)]);
        disp(['Number of iterations: ',num2str(i)]);
        disp('Iterations ');
        disp(approx);
        return;
    end
end
```

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

Main Task 3 :

```
format long;
p = [-1, -7, 7, 3, 9];
dP = polyder(p);
dP2 = polyder(dP);

disp('Error tolerance is 10^-12');
disp('Searching roots of polynomial p(x)');
disp('Coefficients of f(x) = ');
disp(p);
disp('Coefficients of f''(x) = ');
disp(dP);
disp('Coefficients of f''''(x) = ');
disp(dP2);

for x2 = [-7.5, 2, -1i, -7i]
    disp(['Initial point = ', num2str(x2)]);

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

    disp('Muller''s MM2 method');
    x_muller2 = mm2(p,dP,dP2, x2);

    if not(isreal(x_laguerre))
        continue;
    end

    X = (x2 - 1):0.01:(x2+1);
    figure;
    plot(X,polyval(p,X),'b-',x_muller2,0,'cx',x_laguerre,0,'rx',x2,0,'g*');
    legend('f(x)','zero point (Muller 2)','zero point(Laguerre)','initial point');
    xlabel('x');
    ylabel('y');
    title('Real zero');
    grid on;
end
```

Laguerre's Method :

```
function x = laguerre(p,dP,dP2,x)
    approx = [];
    n = length(p) - 1;
    for i = 0:100 %Maximum number of iterations = 100
        px = polyval(p,x);
        if abs(px) < 1e-12
            disp('Zero point, x = ');
            disp(x);
            disp(['f(x) = ', num2str(c)]);
            disp(['Number of iterations: ',num2str(i)]);
            disp('Iterations');
            disp(approx);
            return;
        end
```

```
        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
```