ENUME ASSIGNMENT B no.22

TAIMUR RAHMAN

Contents

Find Roots 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 we find the root interval, we can use an **iterative** method to find the root.

In the question, the interval is already given, [-5, 10] over the function:

$$f(x) = -1.5 + 0.3x - xe^{-x}$$

f(-5) = 739.0657, f(10) = 1.499546, f(0) = -1.5. f(-5).f(0) < 0, f(0).f(10) < 0 and f(x) is continuous, so at least two roots exist in the interval. As for the iterative methods, we will use **false position method** (*regula falsi* method) and **Newton's method**.

False Position Method

The false position method is very similar to bisection method. The bisection method uses a division point c from interval [a, b] where c = (a + b)/2, false position uses c = (af(b) – bf(a))/(f(b)-f(a)). [a, c] and [c, b] is then checked to find if it has the root and it continues until a root is found. For false position method, the point of intersection is the secant line connecting the ends of the interval [a, b] and the x-axis.

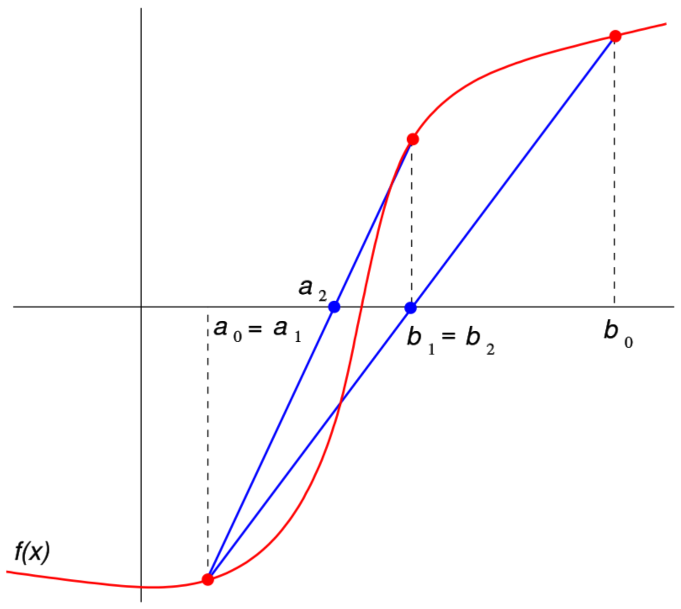


Fig 1: Illustration of the false posititon method. Created by Jitse Niesen using Xfig.

The false position method is always convergent because it always chooses and shortens the interval containing the root. It is linearly convergent with order of convergence, p=1 if the function is continuous and differentiable.

"Order of convergence of an iterative method is defined as the largest number p ≥ 1 such that $\lim\limits_{n\to\infty} \frac{|x_{n+1}-\alpha|}{|x_n-\alpha|^p} = k < \infty$ *" – [1] Numerical Methods, Piotr Tatjewski*

Newton's Method

"This method is also called the Tangent Method. It operates by approximation of the function f(x) by the first order of its expansion into Taylor series at the current point x_n *(current approximation of the root)*

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

The next point is a root of obtained linear function:

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

This leads to:

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

Newton's Method is locally convergent. When it is convergent, then it is very fast (quadratic convergence, p = 2). It is very effective when function derivative at the root is far from 0. However, when it is close to 0, then it is very sensitive to numerical errors."– [2] Numerical Methods,Piotr Tatjewski

The Newton's Method is divergent when the function derivative near the root is close to 0, because the tangent line becomes near horizontal.

The graph of the function looks like this:

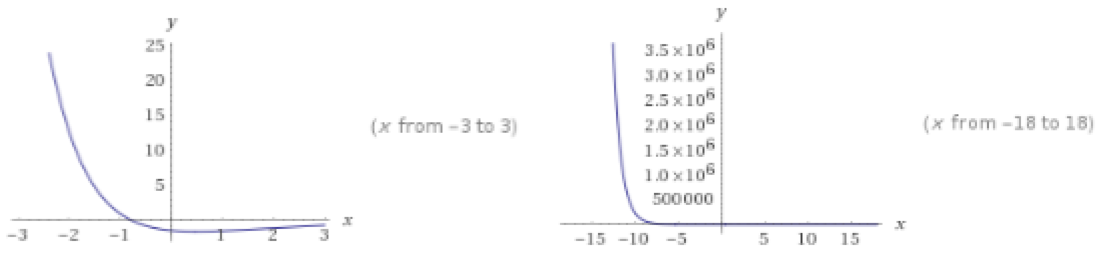


Fig 2: Plots of $f(x) = -1.5 + 0.3x - xe^{-x}$ from WolframAlpha and the roots are x = -0.788988 and x = 5.10336 so I expect the values to be similar.

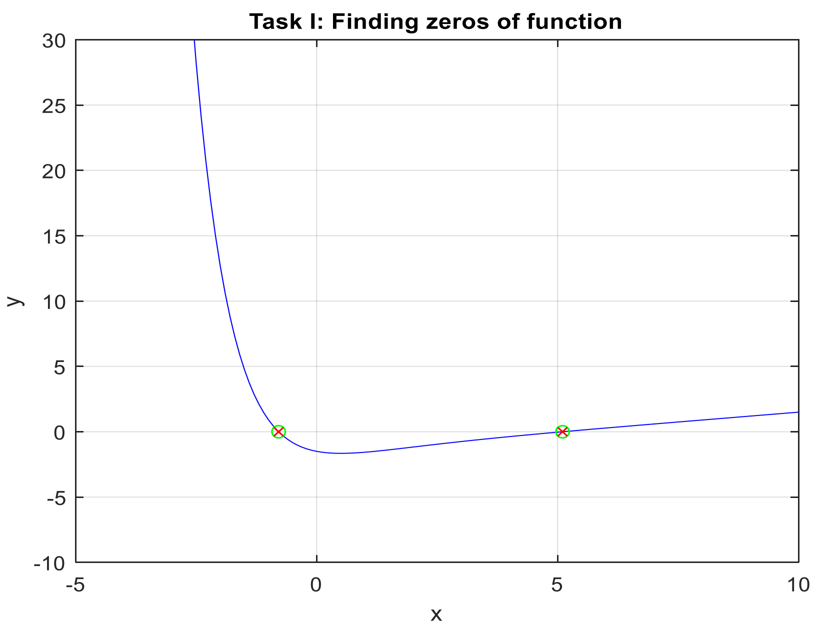
However, there is possibility that Newton's method may diverge because function derivative is close to zero at the roots.

Result

```
Estimated roots:
    -1      5
```

```
 Regula Falsi Method:

 1      -0.620275     1      5.105671
 2      -0.759687     2      5.103372
 3      -0.784133     3      5.103365
 4      -0.788191     4      5.103365
 5      -0.788857     5      5.103365
 6      -0.788967     6      5.103365
 7      -0.788985     7      5.103365
 8      -0.788988
 9      -0.788988
 10     -0.788988
 11     -0.788988
 12     -0.788988
 13     -0.788988
 14     -0.788988
 15     -0.788988
 16     -0.788988
 17     -0.788988
 18     -0.788988
 19     -0.788988
 20     -0.788988


 Root:                Root:
    -0.7890               5.1034

 Iterations:          Iterations:
     20                    7
```

```
 Newton's method:

 1      -0.821226     1      5.103042
 2      -0.789844     2      5.103365
 3      -0.788989     3      5.103365
 4      -0.788988     4      5.103365
 5      -0.788988


 Root:                Root:
    -0.7890               5.1034

 Iterations:          Iterations:
      5                    4
```



Conclusion

The false position method is much slower than Newton's method due to linear convergence compared to quadratic convergence, but Newton's method is more prone to diverge if function derivative is close to zero. For the methods I got the values I expected, and as seen by the iterations, false position method takes more iterations to arrive at a proper value while Newton method can come up with the value in less iterations.

Muller’s Method

Theory

Any polynomial is a continuous and differentiable nonlinear function with exactly n roots which can either be real or complex, single or multiple. While Newton’s method can also be used to find roots of polynomials, there are methods developed to find roots of polynomials and complex roots as well. One of such methods is called the Muller’s method.

“The idea of the Muller’s method is to approximate the polynomial locally in a neighbourhood of a root by a quadratic function. The method can be developed using a quadratic interpolation based on three different points.” [2] Numerical Methods, Piotr Tatjewski

There are two versions of the Muller’s method, referenced as MM1 and MM2.

MM1: Let us consider three points x0, x1, x2 alongside their polynomial values f(x0), f(x1), f(x2). A parabola is formed that goes through these points, and the roots of said parabola are selected. One of the roots are used for the next approximation of the solution.

Let us assume that x2 is an actual approximation of the root of the polynomial. So we introduce a new incremental variable z = x – x2, and use the differences to make z0 = x0 – x2, z1 = x1 – x2. The interpolating parabola is then defined as follows:

$$y(z) = az^2 + bz + c \rightarrow z_{\pm} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

MM2: Another version of the Muller’s method that uses the values of a polynomial and its first and second order derivatives at one point only. It is slightly more effective numerically because it calculates values at only a single point rather than three points, therefore is it often recommended.

$$y(z) = az^2 + bz + c \rightarrow z_{\pm} = \frac{-2f(x_k)}{f'() \pm \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}$$

The Muller’s method is locally convergent, and the order of convergence is 1.84. Therefore, this method is locally more effective than the secant method and almost as fast as the Newton’s method. It is also able to find complex roots as well.

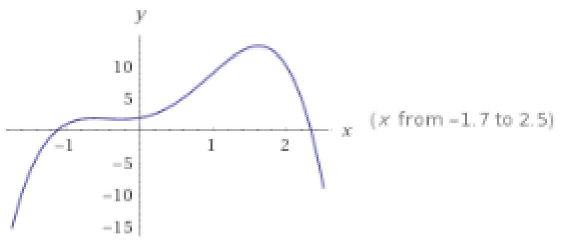


Fig 3: Plot of the polynomial given to me in the question from Wolfram Alpha, with real roots = -1.1277, 2.3336.

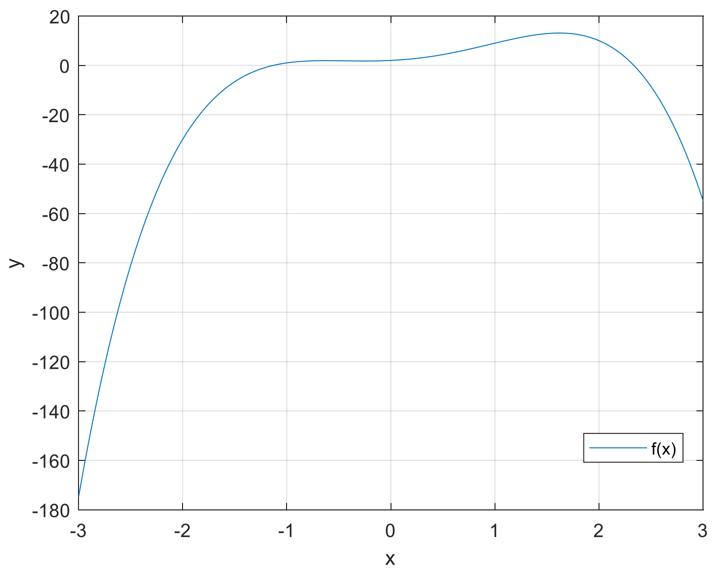
I expect to get a similar result.

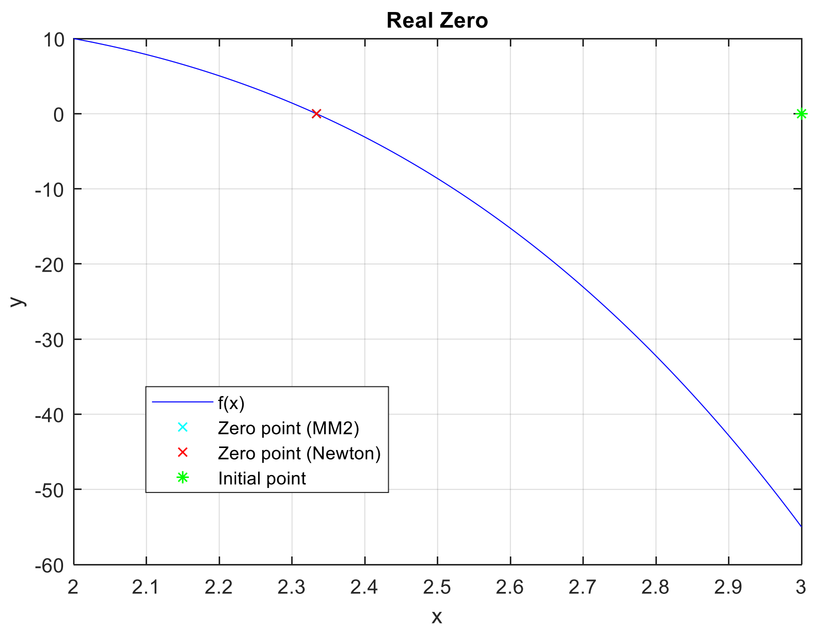
Result

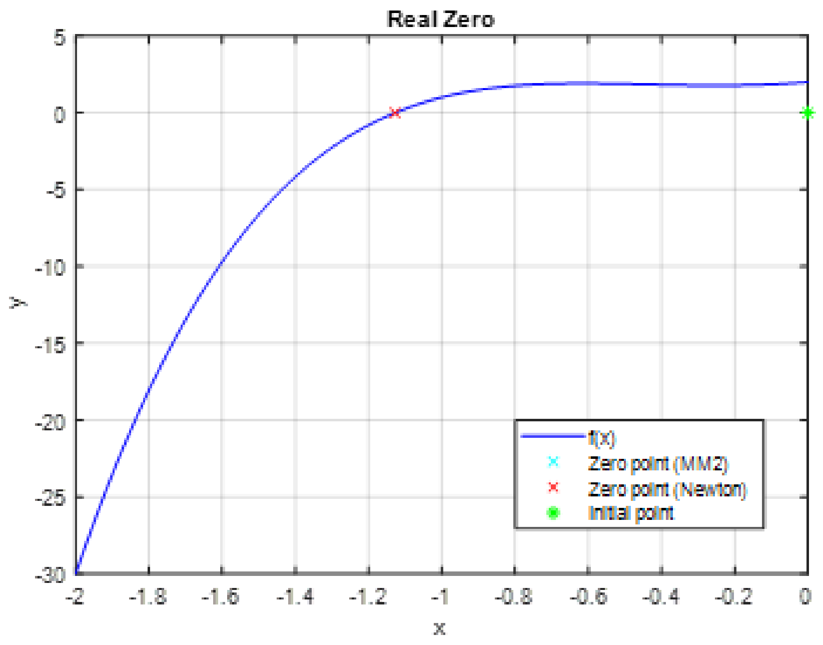
```
Error tolerance is 10^-12
Searching roots of polynomial p(x)
Coefficients of f(x) =
    -2     2     5     2     2

Coefficients of f'(x) =
    -8     6    10     2

Coefficients of f''(x) =
   -24    12    10

Muller's MM1 Method
initial point = [-2,-1.5,-1]
Zero point, x =
 -1.127745687577845 + 0.000000000000000i

f(x) = -8.8818e-16+4.7552e-23i
Number of iterations: 10
Iterations
 -0.992063492063492 - 0.177997313428617i
 -0.979278053704772 - 0.004699568980687i
 -1.198370135716937 - 0.182962611790083i
 -1.101739375991089 - 0.067279486682989i
 -1.108679053314788 + 0.005818353003583i
 -1.128505648760080 - 0.001012614176840i
 -1.127741999745120 - 0.000005061283393i
 -1.127745687366549 + 0.000000000176721i
 -1.127745687577845 + 0.000000000000000i
 -1.127745687577845 + 0.000000000000000i

Muller's MM2 Method
initial point = -1
Zero point, x =
  -1.127745687577844

f(x) = 4.4409e-16
Number of iterations: 3
Iterations
  -1.130031981524879
  -1.127745674157795
  -1.127745687577844

Muller's MM1 Method
initial point = [2,2.5,3]
Zero point, x =
   2.333616213822647

f(x) = -7.1054e-15
Number of iterations: 7
Iterations
   2.612667714615792
   2.426531043070173
   2.346574290761838
   2.333876589295529
   2.333616242088024
   2.333616213822612
   2.333616213822647
```

```
Muller's MM2 Method
initial point = 3
Zero point, x =
  2.333616213822647 - 0.000000000000000i

f(x) = -7.1054e-15+1.6178e-20i
Number of iterations: 4
Iterations
  2.235294117647059 + 0.249567099242311i
  2.327373056512880 + 0.002708690903371i
  2.333616254282897 - 0.000000113058920i
  2.333616213822647 - 0.000000000000000i

Newton's Method
Initial point = 3
1     2.576923
2     2.380450
3     2.335801
4     2.333621
5     2.333616
6     2.333616
7     2.333616
Root:
   2.333616213822647

Iterations:
     7

Warning: Imaginary parts of complex X and/or Y arguments ignored
Muller's MM1 Method
initial point = [0,0+1i,0+2i]
Zero point, x =
 -0.102935263122401 + 0.607768996434717i

f(x) = 0-5.5511e-17i
Number of iterations: 9
Iterations
 -0.024384134007528 + 1.352758184850745i
 -0.042593168196341 + 1.054468095970355i
 -0.069889694858652 + 0.819232464338252i
 -0.092953310944892 + 0.675813326793377i
 -0.102432903381606 + 0.617089648809407i
 -0.102981803291846 + 0.607930528534961i
 -0.102935296628568 + 0.607768972155384i
 -0.102935263122373 + 0.607768996434761i
 -0.102935263122401 + 0.607768996434717i

Muller's MM2 Method
initial point = 0+2i
Zero point, x =
 -0.102935263122401 + 0.607768996434717i
```

```
f(x) = 0-5.5511e-17i
Number of iterations: 5
Iterations
 -0.530029156853617 + 1.228909390110159i
 -0.050523355617676 + 0.844248949016467i
 -0.112624397291283 + 0.594969287163621i
 -0.102932108829564 + 0.607768608389567i
 -0.102935263122401 + 0.607768996434717i

Muller's MM1 Method
initial point = [0,0-1i,0-2i]
Zero point, x =
 -0.102935263122401 - 0.607768996434717i

f(x) = 0+5.5511e-17i
Number of iterations: 9
Iterations
 -0.024384134007528 - 1.352758184850745i
 -0.042593168196341 - 1.054468095970355i
 -0.069889694858652 - 0.819232464338252i
 -0.092953310944892 - 0.675813326793377i
 -0.102432903381606 - 0.617089648809407i
 -0.102981803291846 - 0.607930528534961i
 -0.102935296628568 - 0.607768972155384i
 -0.102935263122373 - 0.607768996434761i
 -0.102935263122401 - 0.607768996434717i

Muller's MM2 Method
initial point = 0-2i
Zero point, x =
 -0.102935263122401 - 0.607768996434717i

f(x) = 0+5.5511e-17i
Number of iterations: 5
Iterations
 -0.530029156853617 - 1.228909390110159i
 -0.050523355617676 - 0.844248949016467i
 -0.112624397291283 - 0.594969287163621i
 -0.102932108829564 - 0.607768608389567i
 -0.102935263122401 - 0.607768996434717i
```







Conclusion

We can observe that MM1 is not as efficient when compared to MM2. MM1 has 10, 7, 9, 9 iterations compared to MM2's 3, 4, 5, 5 iterations. Newton has 7 iterations. So MM2 is the most efficient algorithm compared to both MM1 and Newton's method. In the graphs the zero points found by MM2 and Newton are indistinguishable as they found the same zeros. I also get the real roots I expected from all three methods.

Laguerre’s Method

Theory

Laguerre’s method was designed for the same purpose as Muller’s method, to find roots of polynomials. It is defined by the following formula:

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

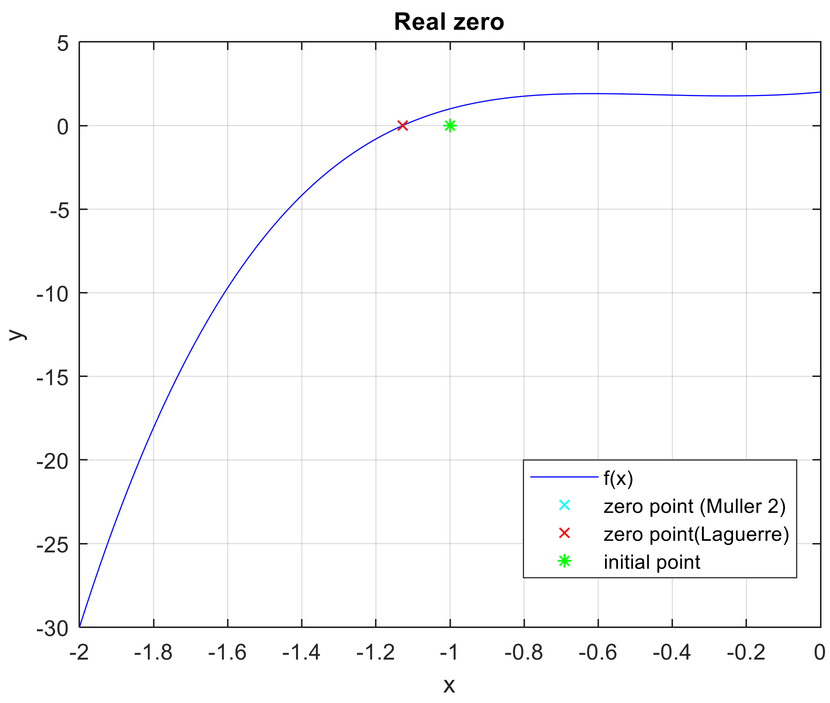
Where n denotes the order of the polynomial, and the sign of the denominator is chosen so that there is a larger absolute value of the denominator,

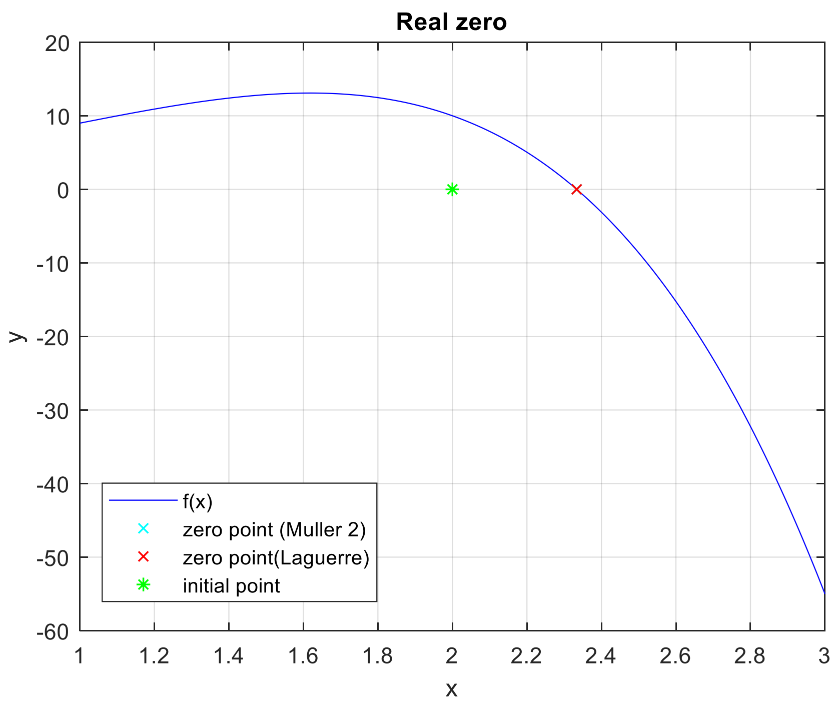
“The Laguerre’s formula is slightly more complex, it takes into account the order of the polynomial, therefore the Laguerre’s method is better, in general. In the case of polynomials with real roots only, the Laguerre’s method is convergent starting from any real initial point, thus it is globally convergent. 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.” - [3] Numerical Methods, Piotr Tatjewski

Results

```
Error tolerance is 10^-12
Searching roots of polynomial p(x)
Coefficients of f(x) =
    -2     2     5     2     2

Coefficients of f'(x) =
    -8     6    10     2

Coefficients of f''(x) =
   -24    12    10

Initial point = -1
Laguerre's method
Zero point, x =
  -1.127745687577844

f(x) = 448732968734.7349
Number of iterations: 3
Iterations
  -1.128126936172247
  -1.127745687571159
  -1.127745687577844

Muller's MM2 method
Zero point, x =
  -1.127745687577844

f(x) = 4.4409e-16
Number of iterations: 3
Iterations
  -1.130031981524879
  -1.127745674157795
  -1.127745687577844

Initial point = 2
Laguerre's method
Zero point, x =
   2.333616213822647

f(x) = 9464615476122.949
Number of iterations: 3
Iterations
   2.333988970756254
   2.333616213822330
   2.333616213822647

Muller's MM2 method
Zero point, x =
   2.333616213822647

f(x) = -7.1054e-15
Number of iterations: 3
Iterations
   2.347539352686619
   2.333615177857328
   2.333616213822647
```

```
Initial point = 0+1i
Laguerre's method
Zero point, x =
 -0.102935263122401 + 0.607768996434717i

f(x) = 3886226.1283+2751949.2885i
Number of iterations: 3
Iterations
 -0.117060939808751 + 0.607965531538050i
 -0.102935777264490 + 0.607769360513832i
 -0.102935263122401 + 0.607768996434717i

Muller's MM2 method
Zero point, x =
 -0.102935263122401 + 0.607768996434717i

f(x) = 0+1.6653e-16i
Number of iterations: 4
Iterations
 -0.200000000000000 + 0.600000000000000i
 -0.103559906273339 + 0.607383014566370i
 -0.102935263024478 + 0.607768996144801i
 -0.102935263122401 + 0.607768996434717i

Initial point = 0-1i
Laguerre's method
Zero point, x =
 -0.102935263122401 - 0.607768996434717i

f(x) = 3886226.1283-2751949.2885i
Number of iterations: 3
Iterations
 -0.117060939808751 - 0.607965531538050i
 -0.102935777264490 - 0.607769360513832i
 -0.102935263122401 - 0.607768996434717i

Muller's MM2 method
Zero point, x =
 -0.102935263122401 - 0.607768996434717i

f(x) = 0-1.6653e-16i
Number of iterations: 4
Iterations
 -0.200000000000000 - 0.600000000000000i
 -0.103559906273339 - 0.607383014566370i
 -0.102935263024478 - 0.607768996144801i
 -0.102935263122401 - 0.607768996434717i
```





Conclusion

The Laguerre's method is seen to be very efficient at finding the roots of the polynomial like MM2 method, both having small number of iterations. However Laguerre's method takes 3 iterations to find complex roots while MM2 method takes 4 iterations.

Appendix

All the MATLAB codes shown on the report were used after `clear` and `clc` commands were used.

Task I: Find Roots of Function

MATLAB code

```
%interval
a = -5;
b = 10;
%function
syms x real;
f = -1.5 + 0.3*x - x*exp(-x);
%estimation
R = estimation(f, a, b);
n = length(R);
%displaying estimated roots
disp('Estimated roots: ');
disp(R);
%solutions using bisection method and newton method
%initialisation
regulafalsiR = zeros(n, 1);
newtonR = zeros(n, 1);

%Regula Falsi method
disp('Regula Falsi Method:');
for k=1:n
    regulafalsiR(k) = false_position(f,R(k),(R(k)+1));
end
%Newton's method
disp('Newton''s method:');
for j = 1:n
    newtonR(j)=newton(f,R(j));
end

%Graphing the results
f = inline(f);
j = 1;
for i = -5:0.1:10
    y(j)=feval(f,i);
    j = j + 1;
end
i = -5:0.1:10;
plot(i,y,'b');
axis([-5 10 -10 30]);
grid on
hold on
plot(regulafalsiR,0,'o g');
plot(newtonR,0,'x r');
title('Task I: Finding zeros of function');
xlabel('x');
ylabel('y');
```

Estimating the roots – `estimation.m`

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

    for i = a : b-1
        %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
```

False Position Method – `false_position.m`

```
function [ c ] = false_position(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
        %false position
        c = (a*yb - b *ya)/(yb - ya);
        %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+b)/2;
    disp('Root:');
    disp(c);
    disp('Iterations:');
    disp(iter);
end
```

Newton's Method – `newton.m`

```
%Newton's Method
function [ x ] = newton(f, x )
    %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-12)
        %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;
        %Number of iterations is incremented
        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
```

Task II: Muller's Method

MATLAB code

```
format long;
p = [-2, 2, 5, 2, 2];
dP = polyder(p);
dP2 = polyder(dP);
syms x real;
fn = -2*x*x*x*x + 2*x*x*x + 5*x*x + 2*x + 2;
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 = (-3):0.01:3;
figure;
plot(X,polyval(p,X));
legend('f(x)');
xlabel('x');
ylabel('y');
grid on;
start = zeros(3,4);
start(:,1) = [-2;-1.5;-1];
start(:,2) = [2;2.5;3];
start(:,3) = [0;1i;2*1i];
start(:,4) = [0;-1i;-2*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 = muller1(p,x0,x1,x2);

    disp('Muller''s MM2 Method');
    disp(['initial point = ' num2str(x2)]);
    x_muller2 = muller2(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 Muller Function – `muller1.m`

```
function x2 = muller1(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 Muller Function – `muller2.m`

```
function x = muller2(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');
```

Task III: Laguerre’s Method

MATLAB Code

```
format long;
p = [-2, 2, 5, 2, 2];
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 = [-1, 2, 1i, -1i]
    disp(['Initial point = ', num2str(x2)]);

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

    disp('Muller''s MM2 method');
    x_muller2 = muller2(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 – `laguerre.m`

```
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
```