
Warsaw University of Technology
Numerical Methods
ENUME

**Project B No.10**

Report

Submitted by:

# Gaurav Chauhan

# 309602

# 1. FINDING ALL ZEROS OF A FUNCTION

The problem is to find all zeros of the function

$$f(x) = 0.4x * \cos(5x) - 5\ln(x + 3)$$

In the interval [5, 14] using:

a. The bisection method,

b. The Newton's method.

The root of a function is point at which the function value is equal to a zero. The zeros of a function refer to the roots of a function. Any function contains real roots contains both real negative and positive values. The key characteristics of detecting the presence of a zero in an interval [a, b] is that the product of the function values at the interval ends is negative, that is, f(a)*f(b)<0.

A quadratic function have an analytic formula to determine its roots. Higher order polynomials and non-polynomial function have no analytic formulas to determine their roots. Different iterative methods have been developed to estimate the root of a function from an initial estimate. An iterative methods is assured to give a root if successive iterates become closer to each other, a property called convergence. The order of convergence is the rate at which iterative approach a certain finite constant. The convergence of an iterative methods is described using errors at nth and nth+1 iterations as follows.

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

Where s is the root of the function, $x_n$ is the nth iterative value, $x_{n+1}$ is the (n+1)th iterative value, A is a constant and p is the order convergence.

In the above stated problem, bisection method and Newton's methods are used to find all the zeros of the function.

## a. THE BISECTION METHOD

Bisection method is based on intermediate theorem. Bisection method has linear convergence

Given a continuous function f(x) in the interval [a, b] and f(a)*f(b)≤0, then a value $x^* \in$ [a0, b0] such that $f(x^*) = 0$. To find the point $x^*$ we follow the following algorithm:

*Step 1. Set the interval [$a_0$, $b_0$] such f(a)*f(b)≤0 to search for the root and also set N maximum number of iterations.*

*Step 2. Set* $c = \frac{a_n + b_n}{2}$

*Step 3. If f(x) =0 or n>N, then x*=c and stop the iterations.*

*Step 4. If f(a)*f(b)<0, set* $a_{n+1} = a_n, b_{n+1} = c$ *Go to step 2*

*Step 5. If f(a)*f(b)>0, set* $a_{n+1} = c, b_{n+1} = b_n$ *Go to step 2*

A graphical visualization of bisection method is shown in figure 1

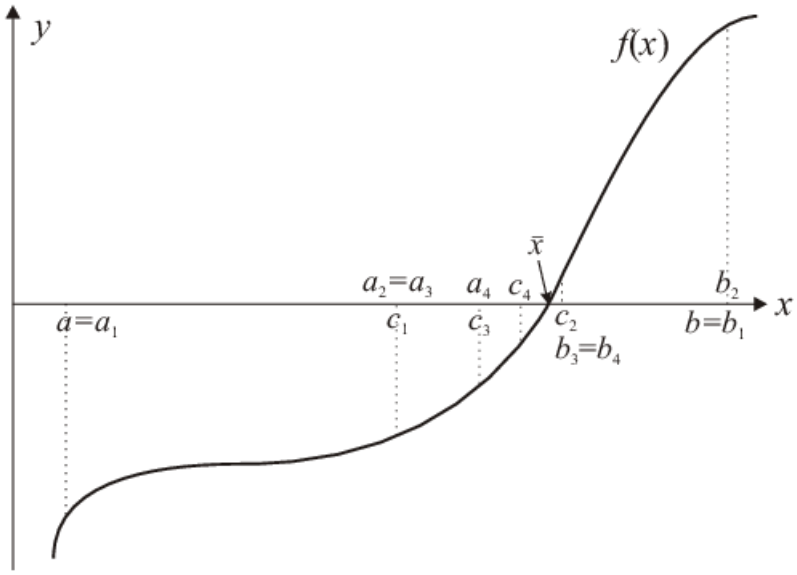



***Figure 1: Bisection method***

The MATLAB code implemented is placed in the appendix section.

**b. THE NEWTON'S METHOD.**

The Newton's method finds the roots of a function using linear approximations. Let us assume a root r exist for the function f(x). Let $x_0$ be an initial estimate of the root r. Let he be the difference between the roots and its estimate, $h = r - x_0$. Applying linear approximation, we define the following equation.

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0)$$

Solving for h we get

$$h \approx -\frac{f(x_0)}{f'(x_0)}$$

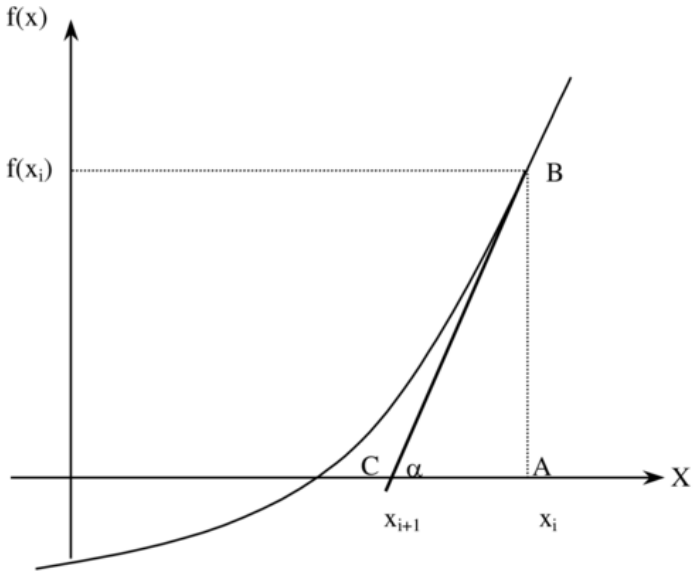
Then the root is defined by

$$r = x_0 + h \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

This is the first estimate of the root. If $x_n$ is the current estimate, then the iterative formula is given by the following.

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

This iterative formula fails if the first derivative is zero or does not exists. The Newton's method may diverge if the initial estimate is not carefully chosen. Newton's method has second order convergence when it converges. The Newton's method is visualized graphically in the figure 2 shown below.




$$\tan(\alpha) = \frac{AB}{AC}$$

$$f'(x_i) = \frac{f(x_i)}{x_i - x_{i+1}}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

***Figure 2: Newton's method***

Newton's method has been applied to solve for all the zeros of the function and the MATLAB script is placed in the appendix section.

**RESULTS: Bisection Method**

There a total of 13 roots that are identified in the interval [5, 14]. A plot of the function in the given interval is done to help determine the valid intervals for root searching. This is shown in figure 3.

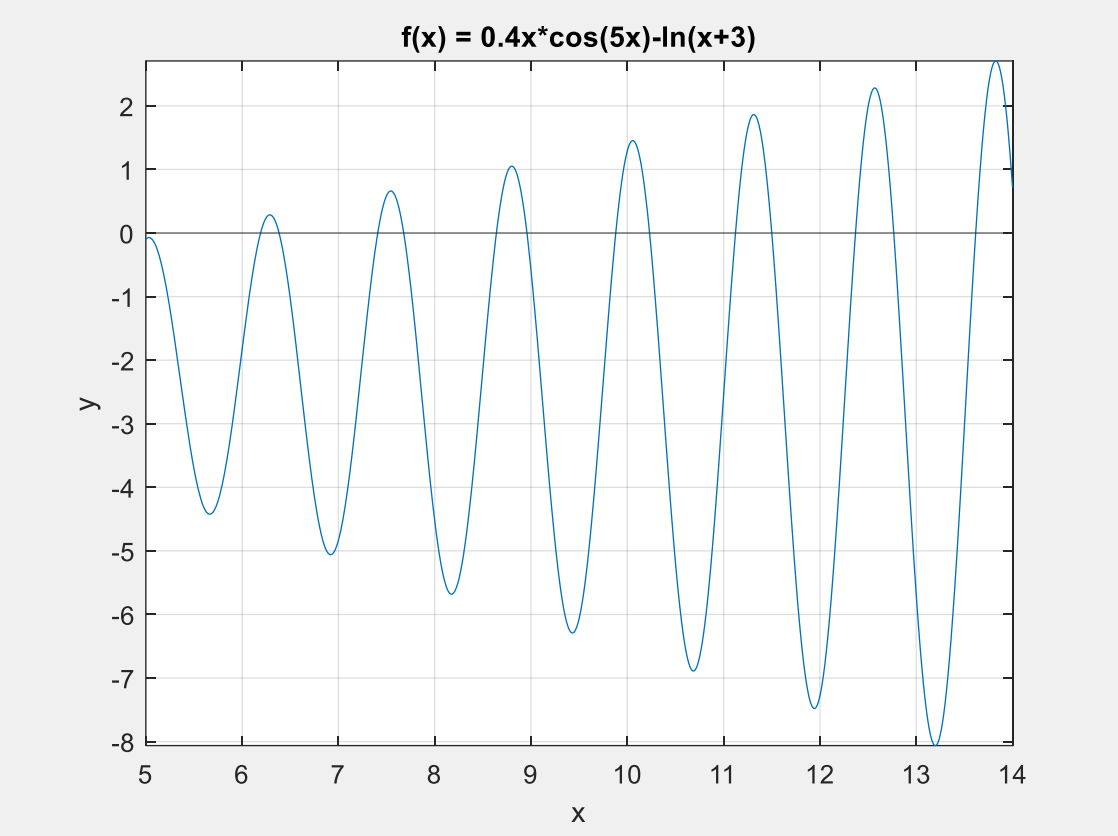



*Figure 3: A plot of the function fx = 0.4x*cos(5x)-ln(x+3)*

Since the function is oscillating, the intervals over which to search for a roots are set to be between a tough and the next peak. The peaks function in Matlab is used to determine all peaks and troughs existing in the interval. Then a for loop is used to get one interval at a time and call the bisection function to return the determined root.

The following figure 4 shows the obtained roots, the corresponding function value and the iterations required to achieve the set tolerance.

```
Zeros using the Bisection Method
   root            f(root)      iteration
 6.1910       6.36946e-10          32
 6.3835       6.43787e-10          32
 7.4080      -1.08176e-09          32
 7.6779      -1.83157e-10          32
 8.6403       2.73523e-10          32
 8.9576      -3.62862e-10          32
 9.8796       7.42728e-10          32
10.2307       3.77008e-10          32
11.1231      -2.31859e-10          32
11.4998      -7.04871e-10          32
12.3692       1.73546e-09          32
12.7664       1.21268e-09          32
13.6173      -1.97810e-09          32
```

*Figure 4: Results of the bisection method*

The roots are plotted together with the original function on the same axis. This confirms that all the roots in the interval [5, 14] have been found. This is shown in figure 5.

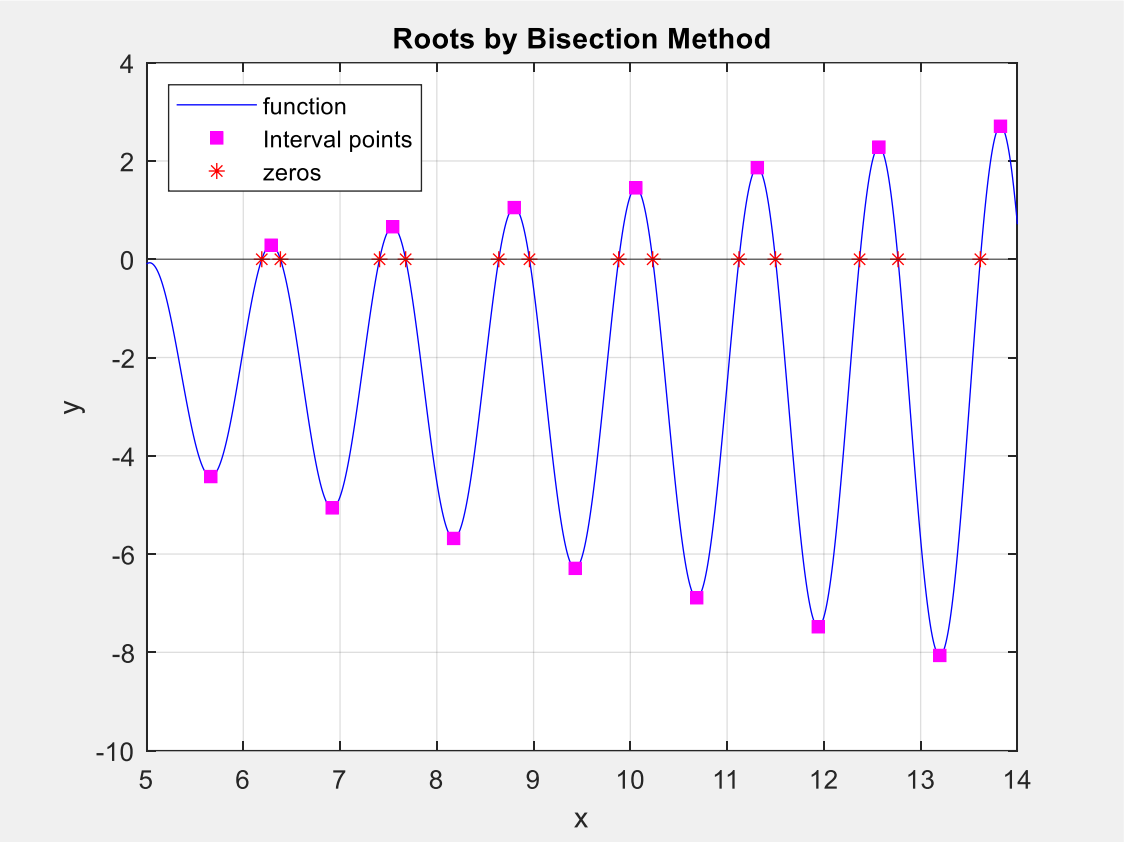



*Figure 5: Results of the bisection method*

The bisection can only determine one roots in a given interval. Therefore each interval must be selected to contain only one root in order to identify all the roots. If the function values are all positive or all negative in the defined interval, the method of bisection fails.

**RESULTS: Newton's method**

The Newton's method is also used to get all the 13 roots in the interval [5, 14]. The initial guess used in the Newton method are defined as the mid-interval of the individual intervals used for the bisection method. The convergence of Newton Method is faster compared to the bisection method as illustrated by the smaller number of iterations required to achieved the set tolerance. Figure 6 and 7 shows the display of all the roots obtained using the Newton's method.

```
Zeros using the Newton's Method
  root           f(root)      iteration
 6.1910      -3.37508e-14         5
 6.3835      -7.14095e-13         5
 7.4080       1.02141e-14         5
 7.6779      -6.66134e-15         5
 8.6403      -3.52385e-12         4
 8.9576      -3.32783e-11         4
 9.8796      -1.10578e-13         4
10.2307      -1.03162e-12         4
11.1231       1.15463e-14         4
11.4998      -6.92779e-14         4
12.3692       2.17604e-14         4
12.7664      -9.32587e-15         4
13.6173       6.12843e-14         4
```

*Figure 6: Results of the Newton method*

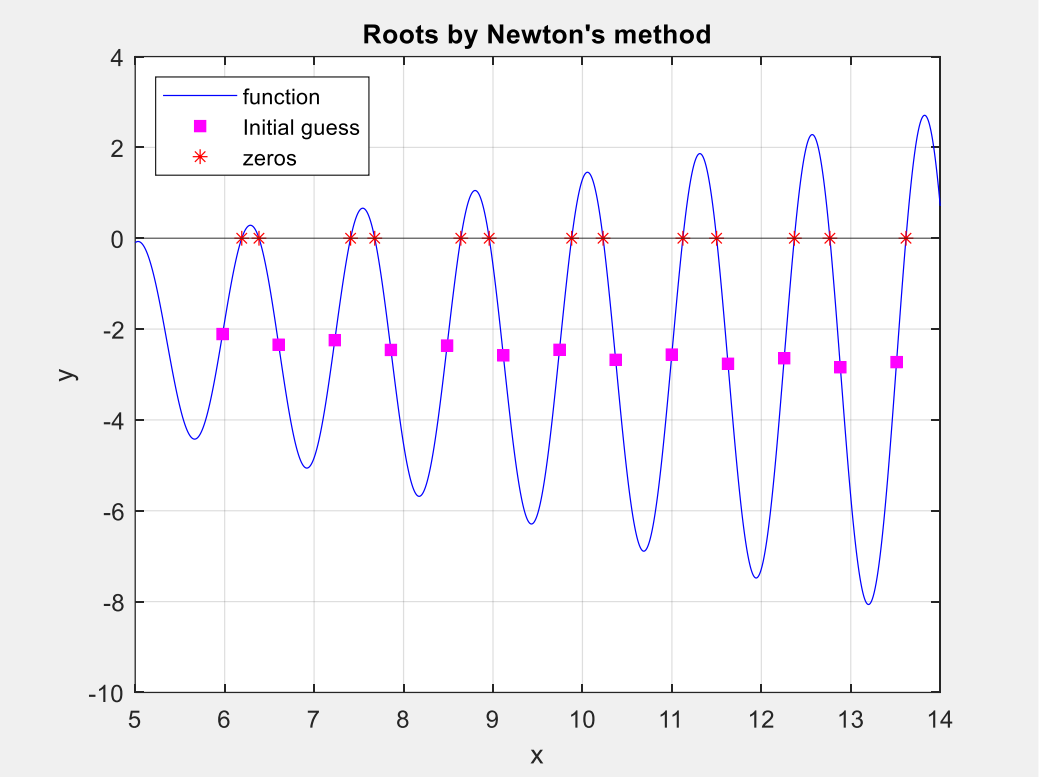



*Figure 7: Results of the Newton method*

**A comparison of the Bisection and Newton's method is shown in Table 1.**

*Table 1: Comparison of Bisection and Newton's methods*

| | | Bisection Method | | Newton's Method | |
|---|---|---|---|---|---|
| | Root | f(root) | iterations | f(root) | iterations |
| Root 1 | 6.1910 | 6.36946E-10 | 32 | -3.37508E-14 | 5 |
| Root 2 | 6.3835 | 6.43787E-10 | 32 | -7.14095E-13 | 5 |
| Root 3 | 7.4080 | -1.08176E-09 | 32 | 1.02141E-14 | 5 |
| Root 4 | 7.6779 | -1.83157E-10 | 32 | -6.66134E-15 | 5 |
| Root 5 | 8.6403 | 2.73523E-10 | 32 | -3.52385E-12 | 4 |
| Root 6 | 8.9576 | -3.62862E-10 | 32 | -3.32783E-11 | 4 |
| Root 7 | 9.8796 | 7.42728E-10 | 32 | -1.10578E-13 | 4 |
| Root 8 | 10.2307 | 3.77008E-10 | 32 | -1.03162E-12 | 4 |
| Root 9 | 11.1231 | -2.31859E-10 | 32 | 1.15463E-14 | 4 |
| Root 10 | 11.4998 | -7.04871E-10 | 32 | -6.92779E-14 | 4 |
| Root 11 | 12.3692 | 1.73546E-09 | 32 | 2.17604E-14 | 4 |
| Root 12 | 12.7664 | 1.21268E-09 | 32 | -9.32587E-15 | 4 |
| Root 13 | 13.6173 | -1.97810E-09 | 32 | 6.12843E-14 | 4 |

Comparison of the bisection method and Newton in evolution of the first root

***Table 2: Comparison of the bisection method and Newton in evolution of the first root***

| Root 1 | Bisection | | Newton | |
|---|---|---|---|---|
| Iteration | Argument | Function value | Argument | Function value |
| 1 | 5.97614761476148 | -2.1095E+00 | 6.15419870457539 | -2.46990E-01 |
| 2 | 6.13208820882088 | -4.2628E-01 | 6.18665609221155 | -2.57700E-02 |
| 3 | 6.21005850585058 | 9.9526E-02 | 6.19095962403734 | -4.92390E-04 |
| 4 | 6.17107335733573 | -1.2540E-01 | 6.19104512613522 | -1.96130E-07 |
| 5 | 6.19056593159316 | -2.7638E-03 | 6.19104516021941 | -3.37510E-14 |
| 6 | 6.20031221872187 | 5.0999E-02 | | |
| 7 | 6.19543907515752 | 2.4763E-02 | | |
| 8 | 6.19300250337534 | 1.1160E-02 | | |
| 9 | 6.19178421748425 | 4.2381E-03 | | |
| 10 | 6.19117507453870 | 7.4711E-04 | | |
| 11 | 6.19087050306593 | -1.0058E-03 | | |
| 12 | 6.19102278880232 | -1.2874E-04 | | |
| 13 | 6.19109893167051 | 3.0934E-04 | | |
| 14 | 6.19106086023641 | 9.0336E-05 | | |
| 15 | 6.19104182451937 | -1.9195E-05 | | |
| 16 | 6.19105134237789 | 3.5573E-05 | | |
| 17 | 6.19104658344863 | 8.1896E-06 | | |
| 18 | 6.19104420398400 | -5.5025E-06 | | |
| 19 | 6.19104539371631 | 1.3436E-06 | | |
| 20 | 6.19104479885016 | -2.0794E-06 | | |
| 21 | 6.19104509628323 | -3.6791E-07 | | |
| 22 | 6.19104524499977 | 4.8785E-07 | | |
| 23 | 6.19104517064150 | 5.9972E-08 | | |
| 24 | 6.19104513346237 | -1.5397E-07 | | |
| 25 | 6.19104515205194 | -4.6998E-08 | | |
| 26 | 6.19104516134672 | 6.4868E-09 | | |
| 27 | 6.19104515669933 | -2.0256E-08 | | |
| 28 | 6.19104515902302 | -6.8844E-09 | | |
| 29 | 6.19104516018487 | -1.9876E-10 | | |
| 30 | 6.19104516076580 | 3.1440E-09 | | |
| 31 | 6.19104516047533 | 1.4726E-09 | | |

| | | | | |
|---|---|---|---|---|
| 32 | 6.19104516033010 | 6.3695E-10 | | |

**Conclusion**

Newton's method emergences the best both in accuracy and speed. A perfect zero, the function values should be zero. Therefore, the accuracy is measured by closeness of the function value to 0. Newton's methods have the smallest magnitude of function values at the estimated roots compared to the bisection method. Also, Newton's method converges to the roots using much fewer iterations compared to the bisection method.

# 2. FINDING ALL ROOTS OF A POLYNOMIAL USING MULLER'S METHODS

The task it find all (real and complex) roots of the polynomial of the given function using the Müller's method implementing both the MM1 and MM2 versions. The results are compared. Real roots of the polynomial are also determined using Newton's methods and compared to result of Müller's methods

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0, \qquad [a_4, a_3, a_2, a_1, a_0] = [1\ 7 - 5\ 8\ 9]$$

**MULLER METHOD**

Müller's methods are root finding algorithms that can determined both the real and complex roots of a polynomial function. The technique is developed from a secant method where three points are used and a quadratic interpolation is done around an initial guess. It also has a higher order convergence than the secant method. Theoretically, the Müller method have a convergence order of about 1.84.

There are two version of Müller methods, namely Müller Method 1 (MM1) and Müller Method 2 (MM2).

**Müller Method 1 (MM1)**

The method starts with three points near the root x0, x1, x2, where x2 is the initial root guess. Quadratic interpolation is performed using the three points their corresponding function values, f(x0), f(x1), and f(x2). Three equations are constructed as follows:

$$f(x_0) = az_0^2 + bz_0 + c = y(z_0)$$

$$f(x_1) = az_1^2 + bz_1 + c = y(z_1)$$

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

Where

$$z = x - x_2$$

$$z_0 = x_0 - x_2$$

$$z_1 = x_1 - x_2$$

The general interpolating quadratic function has the following form:

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

The corresponding roots are given by

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

The smaller root is selected for the next approximation of the root. This leads to a recurrence formula of the following form.

$$x_{n+1} = x_n - \frac{2c}{\max\left(b \pm \sqrt{b^2 - 4ac}\right)}$$

After each iteration, a new set of three point is selected with x2 being the new roots and two other points selected from the previous set which are closest to the new approximation.

Once the first root is obtained, other roots are determined from a polynomial of one order less than the original polynomial. The polynomial is reduced by dividing the original polynomial with the linear term (x-x*) where x* is the previous root determined. The process continues until all roots are found.

**Müller Method 2 (MM2)**

This methods seeks to improve order of convergence by using first and second order derivatives at only a single point. A similar quadratic interpolation equation used in MM1 is used.

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

Where

$$z = x - x_k$$

The constants of the interpolating equation are the defined as follows.

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

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

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

Substituting for constants in the recurrence formula of MM1 we get the recurrence formula for MM2 as follows

$$x_{n+1} = x_n - \frac{2f(x_k)}{\max\left(f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f''(x_k)f(x_k)}\right)}$$

Once the first root is obtained, other roots are determined from a polynomial of one order less than the original polynomial. The polynomial is reduced by dividing the original polynomial with the linear term (x-x*) where x* is the previous root determined. The process continues until all roots are found.

The MATLAB scripts for MM1 and MM2 are located in the appendix section.

**RESULTS:**

Both MM1 and MM2 are applied to solve the roots of the given polynomial function. MM2 has four sets of initial conditions with each set contains three different points. Each set is applied when the calling the function. MM2 uses single point as initial conditions. In both MM2, initial conditions for searching real roots are set to be real while initial conditions for searching the complex roots are also complex.

To come up with reasonable initial guess, the function was plotted to visualize the x-intercepts. This is shown in figure 8.

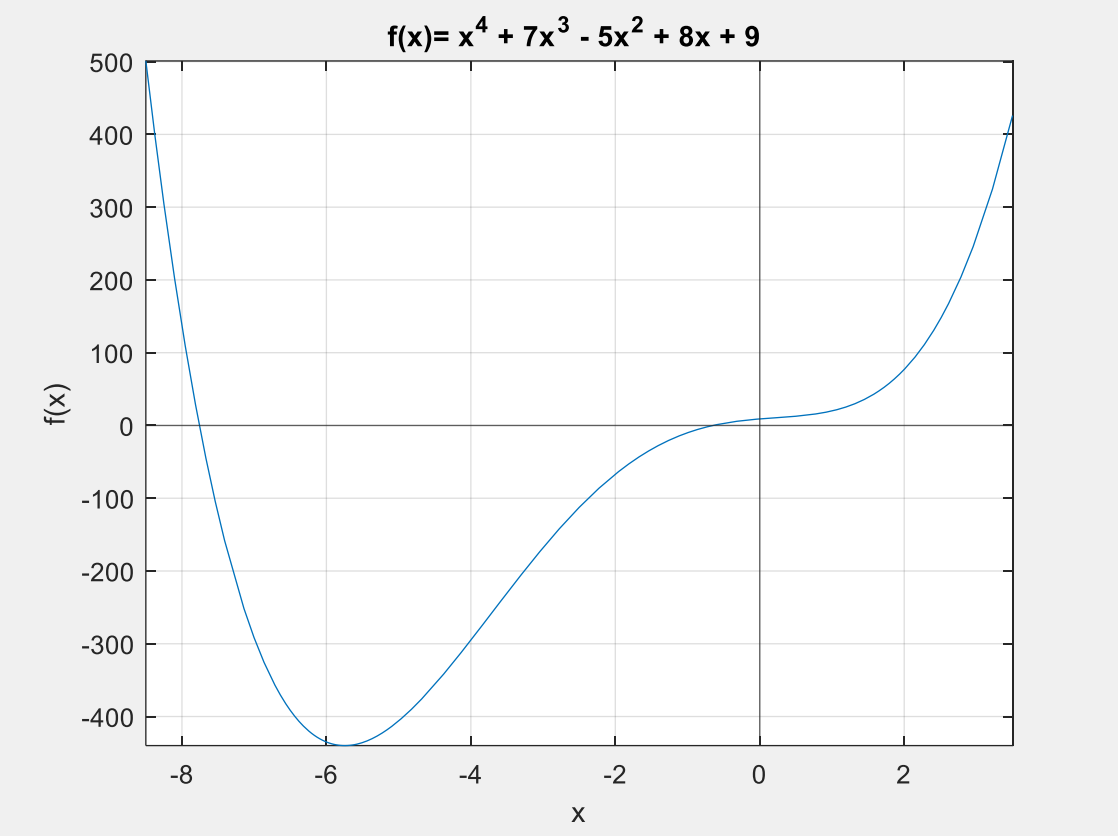



*Figure 8: A plot of the function for problem II*

After application of MM1, MM2 and Newton methods, the two real roots and two complex roots are obtained as shown in the following results in figures 9, 10, 11&12.

```
Roots by MM1
r1 = -7.7581+0.0000i      f(r1) = -1.30917e-12    Iter = 4
r2 = -0.6475+0.0000i      f(r2) = -1.77636e-15    Iter = 11
r3 =  0.7028+1.1392i      f(r3) = 0.00000e+00    Iter = 6
r4 =  0.7028-1.1392i      f(r4) = 0.00000e+00    Iter = 6


Roots by MM2
r1 = -7.7581+0.0000i      f(r1) = 9.41469e-14    Iter = 1
r2 = -0.6475+0.0000i      f(r2) = 1.20792e-13    Iter = 1
r3 =  0.7028+1.1392i      f(r3) = 0.00000e+00    Iter = 1
r4 =  0.7028-1.1392i      f(r4) = 0.00000e+00    Iter = 1


Roots by Newton Method
r1 = -7.7581     f(r1) = 5.71720e-11    Iter = 9
r2 = -0.6475     f(r2) = -1.77636e-15    Iter = 6
```

*Figure 9: Results of the MM1, MM2, and Newton's method*

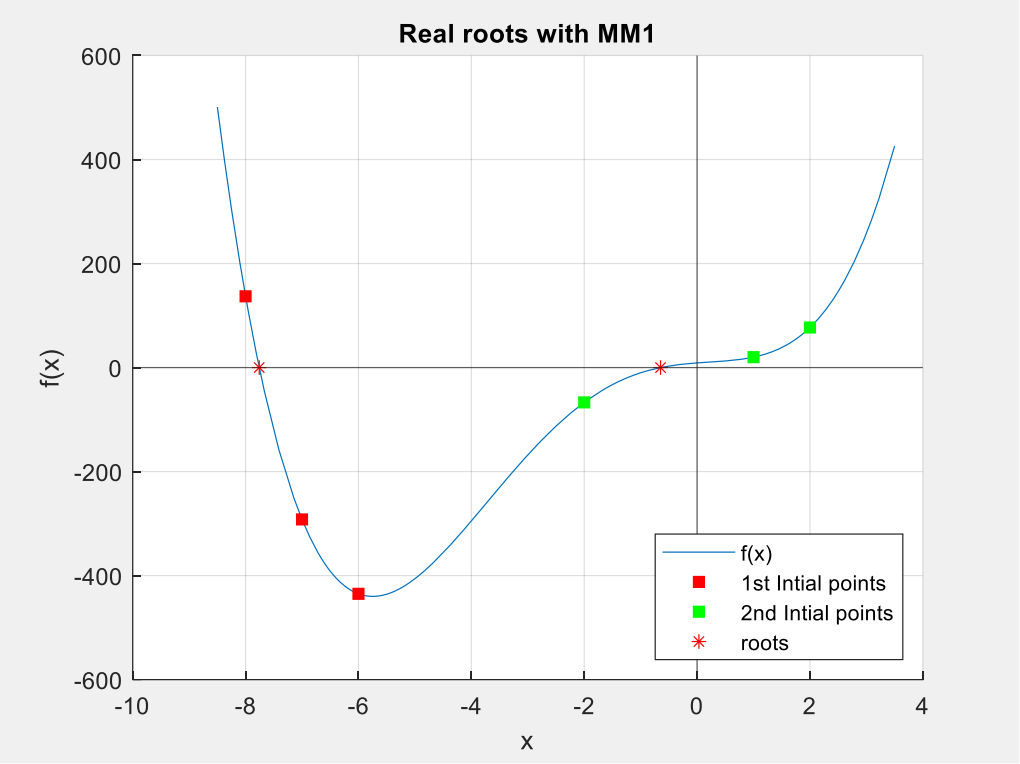



*Figure 10: Plot of real roots solved by MM1*

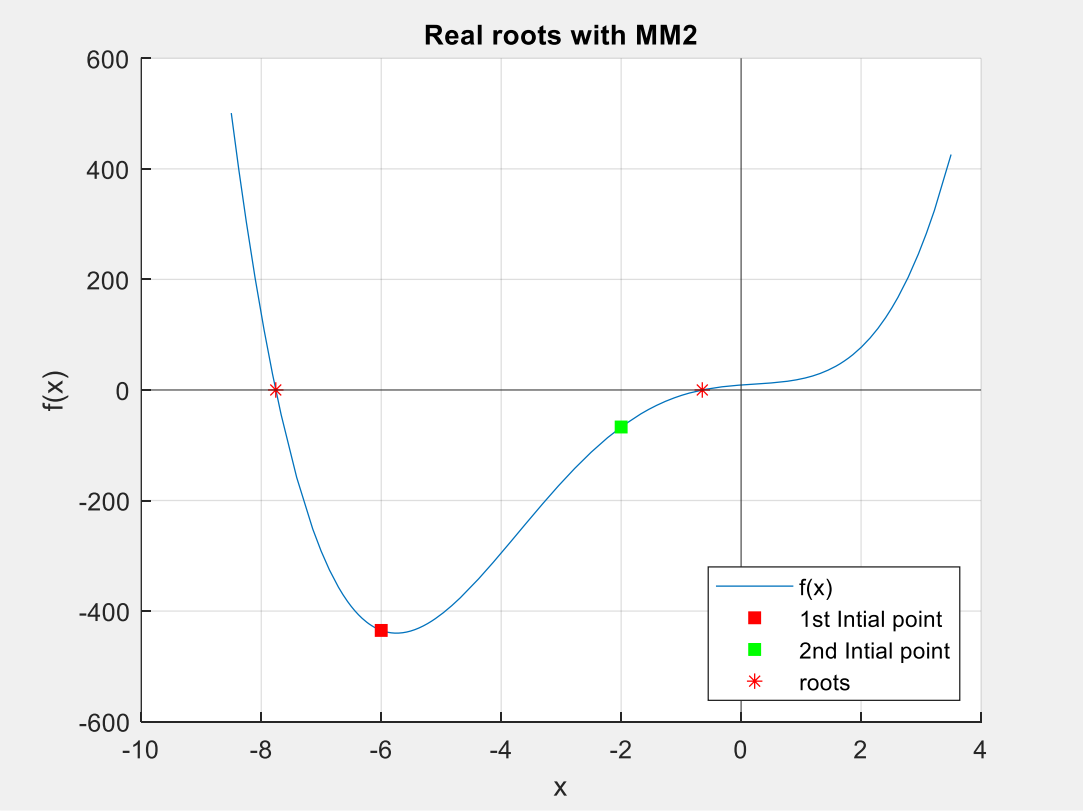



*Figure 11: Plot of real roots solved by MM2*

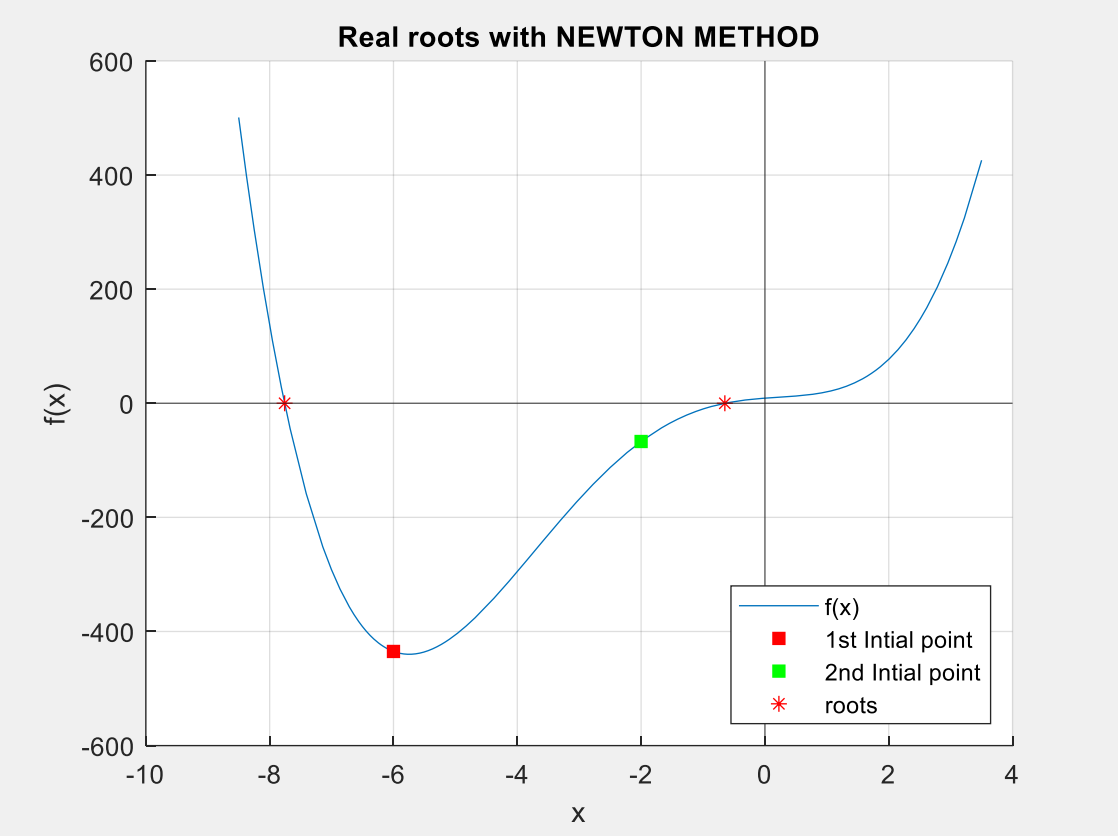



*Figure 12: Plot of real roots solved by Newton's Method*

Comparing the results of the three methods we find out that, complex roots can only be obtained using Muller method and not with Newton methods.

*Table 3: Comparison of finding Root 1 by different methods*

| Root 1 | MM1 | | MM2 | | Newton Method | |
|---|---|---|---|---|---|---|
| Iterations | Argument | Function value | Argument | Function value | Argument | Function value |
| 1 | -7.7733 | 7.9258 | -7.7581 | 9.4147E-14 | -16.875 | 45904 |
| 2 | -7.7584 | 0.14528 | | | -13.361 | 14184 |
| 3 | -7.7581 | 6.2981E-06 | | | -10.851 | 4254.7 |
| 4 | -7.7581 | -1.3092E-12 | | | -9.1643 | 1181.5 |
| 5 | | | | | -8.1921 | 263.28 |
| 6 | | | | | -7.8159 | 30.579 |
| 7 | | | | | -7.7593 | 0.62671 |
| 8 | | | | | -7.7581 | 2.8211E-04 |
| 9 | | | | | -7.7581 | 5.7172E-11 |

**Conclusion**

The MM2 method requires the shortest number of iterations to achieved tolerances, followed MM1 and then by Newton's method. MM2 derives its advantage of converging very first from its use of only a single point together with first and second derivative.

The complex roots of are in both cases obtain in complex conjugate pairs. This is an absolute characteristics of polynomials with complex roots that they only exist as complex conjugate pairs.

# 3. FINDING ALL ROOTS OF A POLYNOMIAL USING LAGUERRE'S METHODS

Laguerre's method is a numerical method used to determine roots of polynomials. The recurrence formula for Laguerre's method is stated as shown in the following equation.

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

When n is the order of the polynomial. This formula takes the sign in the denominator that results to the maximum absolute value of the denominator. The method is very power in obtain simple roots and has third order convergence. The convergence reduces to linear if there are multiple roots for the sought root.

All roots of a function, both real and complex, are obtained by a careful selection of the starting values. The best initial guesses are determined from the graphical representation of the polynomial in order to have an estimate of the x-intercepts.

**RESULTS**

The initial guesses applied in MM2 in problem 2 are the same applied for the Laguerre's method. This is supported by the plot of the function as shown in figure 13.

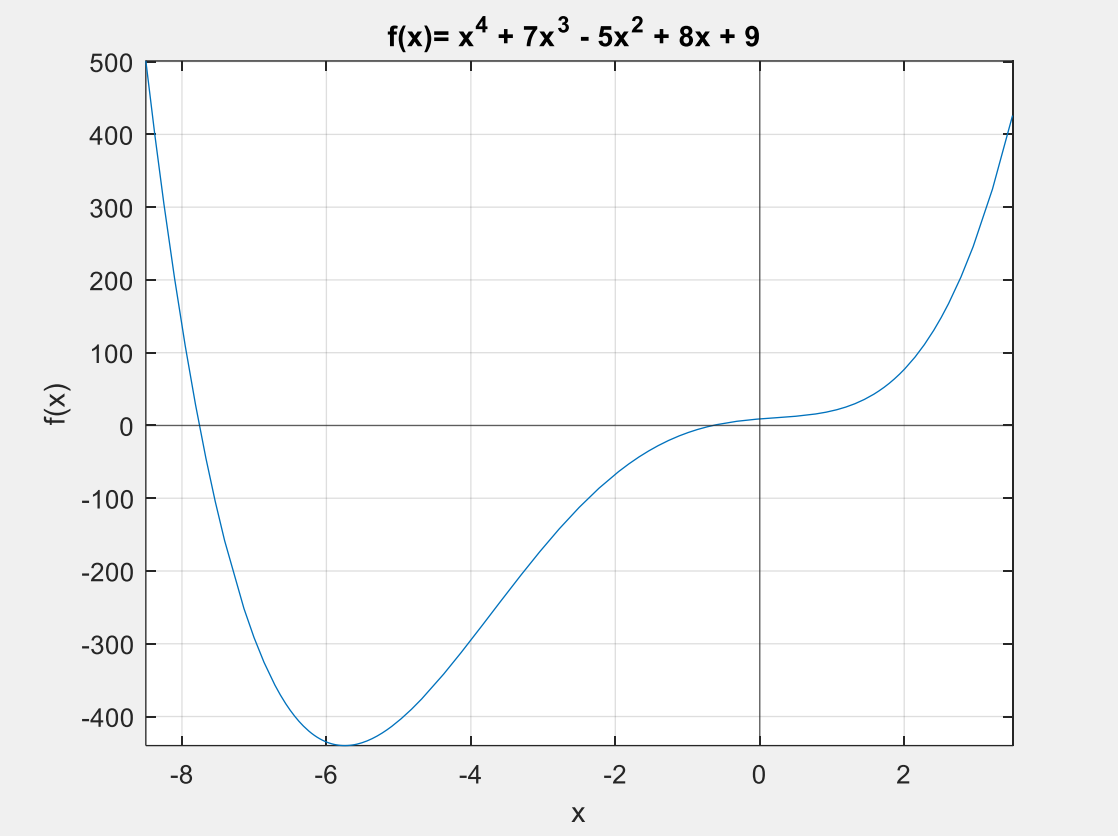



*Figure 13: A plot of the polynomial function*

After the application of Laguerre's method, all roots are obtained within one iteration. This is similar to what was observed with MM2. The calculated roots are obtained as shown in figure 14.

```
Roots by Laguerre's method
r1 = -7.7581+0.0000i      f(r1) = 9.41469e-14     Iter = 1
r2 = -0.6475+0.0000i      f(r2) = 3.55271e-15     Iter = 1
r3 =  0.7028+1.1392i      f(r3) = 3.55271e-15     Iter = 1
r4 =  0.7028-1.1392i      f(r4) = 3.55271e-15     Iter = 1


Roots by MM2
r1 = -7.7581+0.0000i      f(r1) = 9.41469e-14     Iter = 1
r2 = -0.6475+0.0000i      f(r2) = 1.20792e-13     Iter = 1
r3 =  0.7028+1.1392i      f(r3) = 0.00000e+00     Iter = 1
r4 =  0.7028-1.1392i      f(r4) = 0.00000e+00     Iter = 1
```

*Figure 14: Results of Laguerre's Method and MM2*

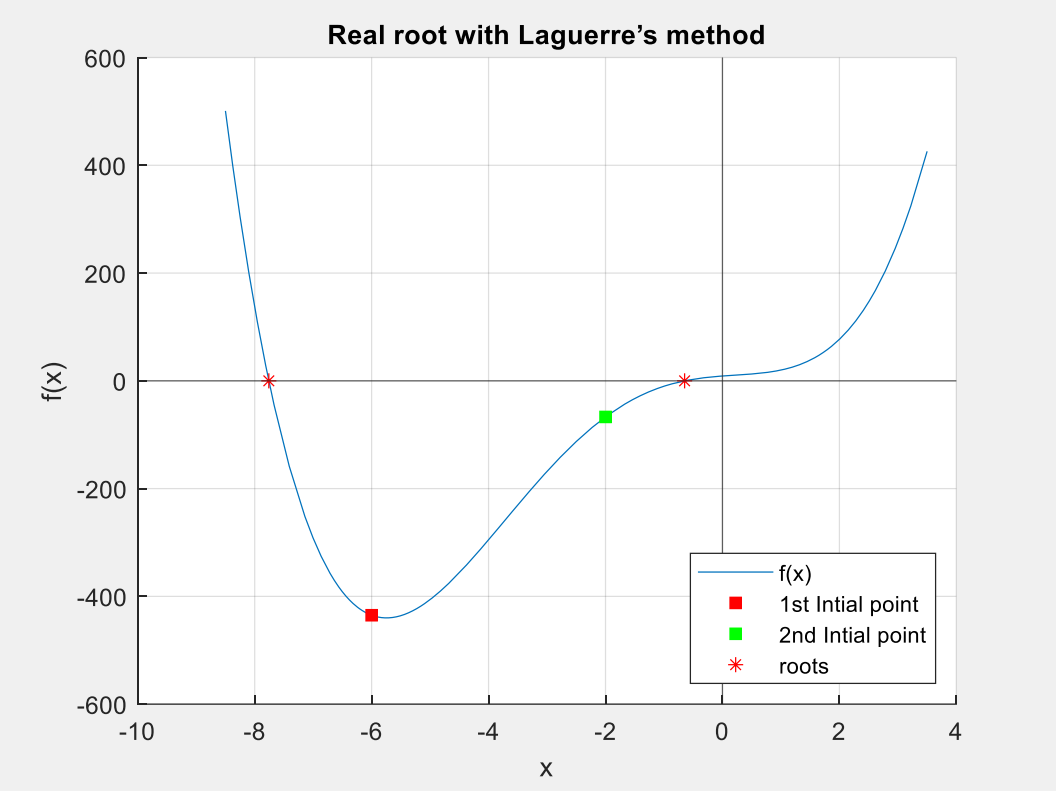



*Figure 15: Plot of real roots solved by Laguerre's Method*

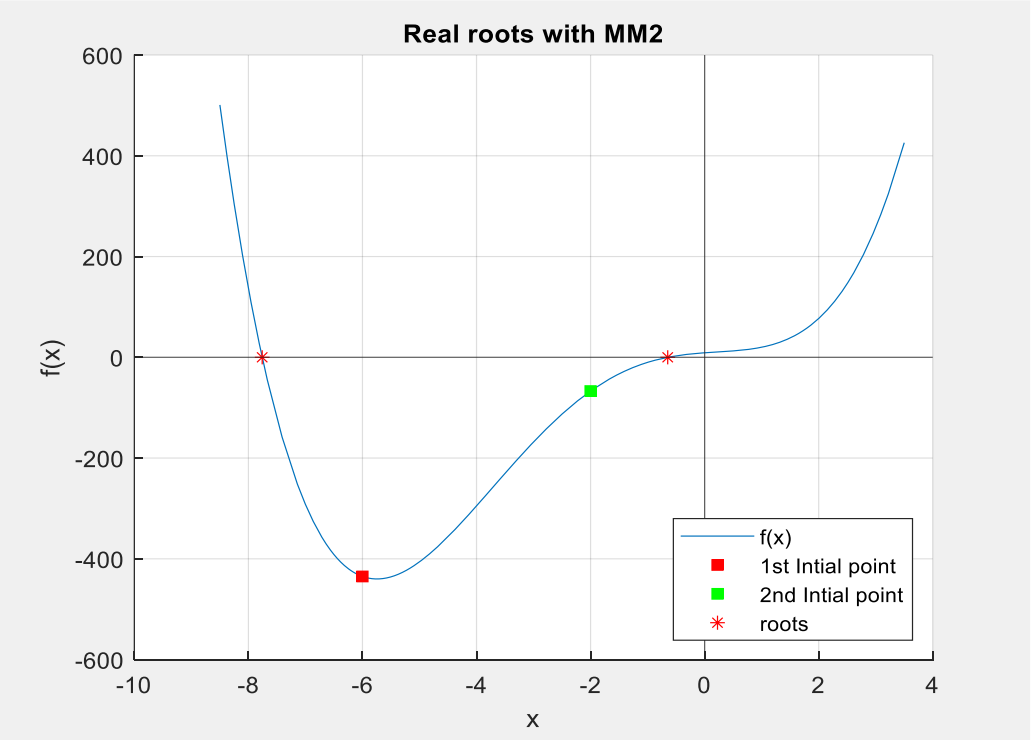



*Figure 16: Plot of real roots solved by MM2 Method*

*Table 4: Comparison of finding Root 1 by Laguerre's method and MM2*

| Root 1 | Laguerre's method | | MM2 | |
|---|---|---|---|---|
| Iterations | Argument | Function value | Argument | Function value |
| 1 | -7.7581 | 9.4147e-14 | -7.7581 | 9.4147e-14 |

**Conclusion**

We find that for the given polynomial, the Laguerre method and MM2 method achieves results of each root with a single iteration. In theory, the Laguerre method is faster than MM2 method. This would be better demonstrated for a polynomials where roots would require several iterations of both methods.

# Appendix

### Problem 1 Matlab script

```
% Find all zeros of the function
%         f(x0 = 0.4x*cos(5x)-4ln(x+3)
% in the interval [5, 14] using:
% a) the Bisection method,
% b) the Newton's method
%
% Clear the work space
clc;clear;close all;
format long;

% Define the function as anonymous function and plot in the search
interval
f = @(x) 0.4.*x.*cos(5.*x)-log(x+3);
figure()
fplot(f,[5,14]);yline(0) % Plot the function in the given interval
grid on;xlabel('x');ylabel('y') % Add axis labels
title('f(x) = 0.4x*cos(5x)-ln(x+3)')

% We notice the function oscillates and therefore, we can use the x-
values
% at peaks and troughs to define serch intervals.
x = linspace(5,14,1e4);
[pks, pklocs] = findpeaks(f(x));          % Original Peaks
[troughs,trlocs] = findpeaks(-f(x));      % Original Troughs
pktr = [pklocs(:);trlocs(:)];             % Locations
pktrs = sortrows(pktr);                   % Sorted LocationS
pktrs=pktrs(2:end);                       % Remove first index
s_intervals = x(pktrs)';                  % x_values of peaks and troughs
f_intervals = f(s_intervals);             % f_values at the peaks and
troughs



%% Roots by the Bisection Method
fprintf('Zeros using the Bisection Method\n')
tol = 1e-10;      % Tolerance
Num_r = length(s_intervals);
Sol_b = zeros(Num_r-1,3);
fprintf('  root       \tf(root) \titeration\n')
for i=1:Num_r-1
    a = s_intervals(i); b = s_intervals(i+1); % Search Interval
    [X,it] = bisectionMethod(f, a, b, tol); % Call Bisection method
    Sol_b(i,1) = X(end); % Root
    Sol_b(i,2) = f(X(end)); % function value at the root
    Sol_b(i,3) = it; % Number of iteration used to obtain the root
    fprintf('%7.4f \t%12.5e\t\t%g\n',Sol_b(i,1),Sol_b(i,2),Sol_b(i,3))
    if i==1
        X_b = X;
    end
end
figure()
plot(x,f(x),'b-');% Plot the function in the given interval
hold on
plot(s_intervals,f_intervals,'ms','MarkerFaceColor','m')
plot(Sol_b(:,1),Sol_b(:,2),'r*'); % Plot the rooots in the given interval
yline(0)
grid on;xlabel('x');ylabel('y') % Add axis labels
```

```
title('Roots by Bisection Method')
legend('function','Interval points','zeros','location','northwest')
hold off


%% Roots by the Newton's Method
fprintf('\rZeros using the Newton''s Method\n')
% Determine the delivative of the function
Fsys = sym(f);                  % Convert to symbolic expression
dFsys = diff(Fsys);             % Get the symbolic derivative
df = matlabFunction(dFsys);     % Convert derivative to anonymous function

tol = 1e-10;      % Tolerance
Num_r = length(s_intervals);
Sol_n = zeros(Num_r-1,3);
initial_guess = (s_intervals(1:end-1)+s_intervals(2:end))./2;
fprintf('  root       \tf(root) \titeration\n')
for i=1:Num_r-1
    r = initial_guess(i); % SInitail guess
    [X,it] = Newton_Method(f, df, r, tol); % Call Newtons method
    Sol_n(i,1) = X(end); % Root
    Sol_n(i,2) = f(X(end)); % function value at the root
    Sol_n(i,3) = it; % Number of iteration used to obtain the root
    fprintf('%7.4f \t%12.5e\t\t%g\n',Sol_n(i,1),Sol_n(i,2),Sol_n(i,3))
    if i==1
        X_n = X;
    end
end
figure()
plot(x,f(x),'b-');% Plot the function in the given interval
hold on
plot(initial_guess,f(initial_guess),'ms','MarkerFaceColor','m')
plot(Sol_n(:,1),Sol_n(:,2),'r*'); % Plot the rooots in the given interval
yline(0)
grid on;xlabel('x');ylabel('y') % Add axis labels
title('Roots by Newton''s method')
legend('function','Initial guess','zeros','location','northwest')
hold off

%% Comparison
% A comparison of the results containing a table with all successive
% iteration points for the first root
%*****************************************************************
fprintf('\r\r')
iter_bisection = [1:length(X_b)]';
roots_bisection = X_b(:);
funValue_bisection= f(X_b(:));
B_Table = table(iter_bisection,roots_bisection,funValue_bisection);
disp(B_Table)
%*****************************************************************************
iter_Newton = [1:length(X_n)]';
roots_Newton = X_n(:);
funValue_Newton = f(X_n(:));
N_Table = table(iter_Newton,roots_Newton,funValue_Newton);
disp(N_Table)
```

**Problem 2 Matlab script**

```
% Find all (real and complex) roots of the polynomial
% f(x) = a4x4+ a3x3+ a2x2+a1x+a0, [a4 a3 a2 a1 a0] = [1 7 -5 8 9]
% a) the Muller method 1,
% b) the Muller method 2,
% c) the Newton's method.
%
% Clear the work space
clc;clear;close all;
format short;

% Define the function and its derivatives.
P = [1 7 -5 8 9];           % Coefficients of the polynomial
dP = polyder(P);            % Coefficients of the 1st derivative
dP2 = polyder(dP);          % Coefficients of the 2nd derivative
f = @(x) polyval(P,x);      % Anonymous function for the polynomial
df = @(x) polyval(dP,x);    % Anonymous function for the 1st derivative
df2 = @(x) polyval(dP2,x);  % Anonymous function for the 2nd derivative

% Define other constants to use in the program
tol = 1e-10; % Set the toleance
mm1_roots = zeros(4,1);
mm2_roots = zeros(4,1);
newt_roots = zeros(2,1);

% Visualize the function revealing all x-intercepts
figure() % Figure 1
fplot(f,[-8.5 ,3.5])
yline(0);xline(0);grid on;
xlabel('x');ylabel('f(x)');grid on
title('f(x)= x^4 + 7x^3 - 5x^2 + 8x + 9 ')


%% Solve for all Roots using MM1
% Finding first root
IP1 = [-6,-7,-8];% First set of INITIAL POINTS
mm1_X1 = MM1(P, IP1(1), IP1(2), IP1(3), tol);
mm1_roots(1) = mm1_X1(end); % Store the first Root

% Finding second root
IP2 = [-2,1,2];% Second set of INITIAL POINTS
mm1_X2 = MM1(P, IP2(1), IP2(2), IP2(3), tol);
mm1_roots(2) = mm1_X2(end); % Store the second Root

% Finding third root
IP3 = [1+0.5i,1+1i,1+1.5i];% Third set of INITIAL POINTS
mm1_X3 = MM1(P, IP3(1), IP3(2), IP3(3), tol);
mm1_roots(3) = mm1_X3(end); % Store the third Root

% Finding fourth root
IP4 = [1-0.5i,1-1i,1-1.5i];% Fourth set of INITIAL POINTS
mm1_X4 = MM1(P, IP4(1), IP4(2), IP4(3), tol);
mm1_roots(4) = mm1_X4(end); % Store the fourth Root

% Display Results
% display(mm1_roots, 'Roots by MM1')

% Plot the results
```

```
figure;hold on;
fplot(f,[-8.5 ,3.5]);
realRoots = mm1_roots(abs(imag(mm1_roots)) <1e-10);
rr = real(realRoots);frr = f(rr);
plot(IP1,f(IP1),'sr','MarkerFaceColor','r')
plot(IP2,f(IP2),'sg','MarkerFaceColor','g')
plot(rr,frr,'*r')
yline(0);xline(0)
grid on;xlabel('x');ylabel('f(x)');
title('Real roots with MM1');
legend('f(x)','1st Intial points','2nd Intial points','roots',...
    'location','southeast');

%% Solve for all Roots using MM2)
IG = [-6, -2,1+0.5i,1-0.5i];% mm2 Initial Points

% Finding first root
mm2_X1 = MM2(P, dP, dP2, IG(1), tol);
mm2_roots(1) = mm2_X1(end); % Store the first Root

% Finding second root
mm2_X2 = MM2(P, dP, dP2, IG(2), tol);
mm2_roots(2) = mm2_X2(end); % Store the second Root

% Finding third root
mm2_X3 = MM2(P, dP, dP2, IG(3), tol);
mm2_roots(3) = mm2_X3(end); % Store the third Root

% Finding fourth root
mm2_X4 = MM2(P, dP, dP2, IG(4), tol);
mm2_roots(4) = mm2_X4(end); % Store the fourth Root

% Plot the results
figure;hold on;
fplot(f,[-8.5 ,3.5]);
realRoots2 = mm2_roots(abs(imag(mm2_roots)) <1e-10);
rr2 = real(realRoots2);frr2 = f(rr2);
plot(IG(1),f(IG(1)),'sr','MarkerFaceColor','r')
plot(IG(2),f(IG(2)),'sg','MarkerFaceColor','g')
plot(rr2,frr2,'*r')
yline(0);xline(0)
grid on;xlabel('x');ylabel('f(x)');
title('Real roots with MM2');
legend('f(x)','1st Intial point','2nd Intial point','roots',...
    'location','southeast');

%% APPLY NEWTON METHOD
% Finding first root
newt_X1 = Newton_Method(f, df, IG(1), tol);
newt_roots(1) = newt_X1(end); % Store the first Root

% Finding second root
newt_X2 = Newton_Method(f, df, IG(2), tol);
newt_roots(2) = newt_X2(end); % Store the first Root

% Plot the results
figure;hold on;
fplot(f,[-8.5 ,3.5]);
plot(IG(1),f(IG(1)),'sr','MarkerFaceColor','r')
```

```
plot(IG(2),f(IG(2)),'sg','MarkerFaceColor','g')
plot(newt_roots,f(newt_roots),'*r')
yline(0);xline(0)
grid on;xlabel('x');ylabel('f(x)');
title('Real roots with NEWTON METHOD');
legend('f(x)','1st Intial point','2nd Intial point','roots',...
    'location','southeast');

%% Display and Compare the methods
fprintf('Roots by MM1\n')
nMM1 = [length(mm1_X1),length(mm1_X2),length(mm1_X3),length(mm1_X4)];
MM1sol = [mm1_roots(:),f(mm1_roots(:)),nMM1(:)];
fprintf('r1 = %7.4f%+7.4fi      f(r1) = %7.5e    Iter = %g\n',...
    real(MM1sol(1,1)),imag(MM1sol(1,1)),MM1sol(1,2),MM1sol(1,3))
fprintf('r2 = %7.4f%+7.4fi      f(r2) = %7.5e    Iter = %g\n',...
    real(MM1sol(2,1)),imag(MM1sol(2,1)),MM1sol(2,2),MM1sol(2,3))
fprintf('r3 = %7.4f%+7.4fi      f(r3) = %7.5e    Iter = %g\n',...
    real(MM1sol(3,1)),imag(MM1sol(3,1)),MM1sol(3,2),MM1sol(3,3))
fprintf('r4 = %7.4f%+7.4fi      f(r4) = %7.5e    Iter = %g\n',...
    real(MM1sol(4,1)),imag(MM1sol(4,1)),MM1sol(4,2),MM1sol(4,3))

fprintf('\r\rRoots by MM2\n')
nMM2 = [length(mm2_X1),length(mm2_X2),length(mm2_X3),length(mm2_X4)];
MM2sol = [mm2_roots(:),f(mm2_roots(:)),nMM2(:)];
fprintf('r1 = %7.4f%+7.4fi      f(r1) = %7.5e    Iter = %g\n',...
    real(MM2sol(1,1)),imag(MM2sol(1,1)),MM2sol(1,2),MM2sol(1,3))
fprintf('r2 = %7.4f%+7.4fi      f(r2) = %7.5e    Iter = %g\n',...
    real(MM2sol(2,1)),imag(MM2sol(2,1)),MM2sol(2,2),MM2sol(2,3))
fprintf('r3 = %7.4f%+7.4fi      f(r3) = %7.5e    Iter = %g\n',...
    real(MM2sol(3,1)),imag(MM2sol(3,1)),MM2sol(3,2),MM2sol(3,3))
fprintf('r4 = %7.4f%+7.4fi      f(r4) = %7.5e    Iter = %g\n',...
    real(MM2sol(4,1)),imag(MM2sol(4,1)),MM2sol(4,2),MM2sol(4,3))

fprintf('\r\rRoots by Newton Method\n')
nNewt = [length(newt_X1),length(newt_X2)];
newtsol = [newt_roots(:),f(newt_roots(:)),nNewt(:)];
fprintf('r1 = %7.4f     f(r1) = %7.5e    Iter = %g\n',...
    newtsol(1,1),newtsol(1,2),newtsol(1,3))
fprintf('r2 = %7.4f     f(r2) = %7.5e    Iter = %g\n',...
    newtsol(2,1),newtsol(2,2),newtsol(2,3))

% A comparison of the results containing a table with all successive
% iteration points for the first root
%**************************************************************************
fprintf('\r\r')
iter_mm1 = [1:length(mm1_X1)]';
roots_mm1 = mm1_X1(:);
funValue_mm1 = f(mm1_X1(:));
MM1_Table = table(iter_mm1,roots_mm1,funValue_mm1);
disp(MM1_Table)
%**************************************************************************
iter_mm2 = [1:length(mm2_X1)]';
roots_mm2 = mm2_X1(:);
funValue_mm2 = f(mm2_X1(:));
MM2_Table = table(iter_mm2,roots_mm2,funValue_mm2);
disp(MM2_Table)

%**************************************************************************
iter_newt = [1:length(newt_X1)]';
roots_newt = newt_X1(:);
```

```
funValue_newt = f(newt_X1(:));
newt_Table = table(iter_newt,roots_newt,funValue_newt);
disp(newt_Table)
```

**Problem 3 Matlab script**

```
% Find all (real and complex) roots of the polynomial
% f(x) = a4x4+ a3x3+ a2x2+a1x+a0, [a4 a3 a2 a1 a0] = [1 7 -5 8 9]
% a) Laguerre's method ,
% b) the Muller method 2,
%
% Clear the work space
clc;clear;close all;
format short;

% Define the function and its derivatives.
P = [1 7 -5 8 9];           % Coefficients of the polynomial
dP = polyder(P);            % Coefficients of the 1st derivative
dP2 = polyder(dP);          % Coefficients of the 2nd derivative
f = @(x) polyval(P,x);      % Anonymous function for the polynomial
df = @(x) polyval(dP,x);    % Anonymous function for the 1st derivative
df2 = @(x) polyval(dP2,x);  % Anonymous function for the 2nd derivative

% Define other constants to use in the program
tol = 1e-10; % Set the toleance
L_roots = zeros(4,1);
mm2_roots = zeros(4,1);
IG = [-6, -2,1+0.5i,1-0.5i];% Initial guess Points

% Visualize the function revealing all x-intercepts
figure() % Figure 1
fplot(f,[-8.5 ,3.5])
yline(0);xline(0);grid on;
xlabel('x');ylabel('f(x)');grid on
title('f(x)= x^4 + 7x^3 - 5x^2 + 8x + 9 ')


%% Solve for all Roots using Laguerre Method)
% Finding first root
L_X1 = laguerre_Method(P, dP, dP2, IG(1), tol);
L_roots(1) = L_X1(end); % Store the first Root

% Finding Second root
L_X2 = laguerre_Method(P, dP, dP2, IG(2), tol);
L_roots(2) = L_X2(end); % Store the second Root

% Finding Third root
L_X3 = laguerre_Method(P, dP, dP2, IG(3), tol);
L_roots(3) = L_X3(end); % Store the Third Root

% Finding Third root
L_X4 = laguerre_Method(P, dP, dP2, IG(4), tol);
L_roots(4) = L_X4(end); % Store the Third Root

%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
% Plot the results
% Plot the results
```

```
figure;hold on;
fplot(f,[-8.5 ,3.5]);
realRoots_L = L_roots(abs(imag(L_roots)) <1e-10);
Lr = real(realRoots_L);fLr = f(Lr);
plot(IG(1),f(IG(1)),'sr','MarkerFaceColor','r')
plot(IG(2),f(IG(2)),'sg','MarkerFaceColor','g')
plot(Lr,fLr,'*r')
yline(0);xline(0)
grid on;xlabel('x');ylabel('f(x)');
title('Real root with Laguerre's method');
legend('f(x)','1st Intial point','2nd Intial point','roots',...
    'location','southeast');

%% Solve for all Roots using MM2)
% Finding first root
mm2_X1 = MM2(P, dP, dP2, IG(1), tol);
mm2_roots(1) = mm2_X1(end); % Store the first Root

% Finding second root
mm2_X2 = MM2(P, dP, dP2, IG(2), tol);
mm2_roots(2) = mm2_X2(end); % Store the second Root

% Finding third root
mm2_X3 = MM2(P, dP, dP2, IG(3), tol);
mm2_roots(3) = mm2_X3(end); % Store the third Root

% Finding fourth root
mm2_X4 = MM2(P, dP, dP2, IG(4), tol);
mm2_roots(4) = mm2_X4(end); % Store the fourth Root


% Plot the results
figure;hold on;
fplot(f,[-8.5 ,3.5]);
realRoots_L = mm2_roots(abs(imag(mm2_roots)) <1e-10);
rr2 = real(realRoots_L);frr2 = f(rr2);
plot(IG(1),f(IG(1)),'sr','MarkerFaceColor','r')
plot(IG(2),f(IG(2)),'sg','MarkerFaceColor','g')
plot(rr2,frr2,'*r')
yline(0);xline(0)
grid on;xlabel('x');ylabel('f(x)');
title('Real root with MM2');
legend('f(x)','1st Intial point','2nd Intial point','roots',...
    'location','southeast');

%% Display and Compare the methods
fprintf('Roots by Laguerre's method\n')
nL = [length(L_X1),length(L_X2),length(L_X3),length(L_X4)];
L_sol = [L_roots(:),f(L_roots(:)),nL(:)];
fprintf('r1 = %7.4f%+7.4fi      f(r1) = %7.5e    Iter = %g\n',...
    real(L_sol(1,1)),imag(L_sol(1,1)),L_sol(1,2),L_sol(1,3))
fprintf('r2 = %7.4f%+7.4fi      f(r2) = %7.5e    Iter = %g\n',...
    real(L_sol(2,1)),imag(L_sol(2,1)),L_sol(2,2),L_sol(2,3))
fprintf('r3 = %7.4f%+7.4fi      f(r3) = %7.5e    Iter = %g\n',...
    real(L_sol(3,1)),imag(L_sol(3,1)),L_sol(3,2),L_sol(3,3))
fprintf('r4 = %7.4f%+7.4fi      f(r4) = %7.5e    Iter = %g\n',...
    real(L_sol(4,1)),imag(L_sol(4,1)),L_sol(4,2),L_sol(4,3))

fprintf('\r\rRoots by MM2\n')
```

```
nMM2 = [length(mm2_X1),length(mm2_X2),length(mm2_X3),length(mm2_X4)];
MM2sol = [mm2_roots(:),f(mm2_roots(:)),nMM2(:)];
fprintf('r1 = %7.4f%+7.4fi      f(r1) = %7.5e    Iter = %g\n',...
    real(MM2sol(1,1)),imag(MM2sol(1,1)),MM2sol(1,2),MM2sol(1,3))
fprintf('r2 = %7.4f%+7.4fi      f(r2) = %7.5e    Iter = %g\n',...
    real(MM2sol(2,1)),imag(MM2sol(2,1)),MM2sol(2,2),MM2sol(2,3))
fprintf('r3 = %7.4f%+7.4fi      f(r3) = %7.5e    Iter = %g\n',...
    real(MM2sol(3,1)),imag(MM2sol(3,1)),MM2sol(3,2),MM2sol(3,3))
fprintf('r4 = %7.4f%+7.4fi      f(r4) = %7.5e    Iter = %g\n',...
    real(MM2sol(4,1)),imag(MM2sol(4,1)),MM2sol(4,2),MM2sol(4,3))

% A comparison of the results containing a table with all successive
% iteration points for the first root
%**************************************************************************
fprintf('\r\r')
iter_Laguerre = [1:length(L_X1)]';
roots_Laguerre = L_X1(:);
funValue_Laguerre = f(L_X1(:));
Laguerre_Table = table(iter_Laguerre,roots_Laguerre,funValue_Laguerre);
disp(Laguerre_Table)
%**************************************************************************
iter_mm2 = [1:length(mm2_X1)]';
roots_mm2 = mm2_X1(:);
funValue_mm2 = f(mm2_X1(:));
MM2_Table = table(iter_mm2,roots_mm2,funValue_mm2);
disp(MM2_Table)
```

# MATLAB Functions

## Bisection Method

```
%% Bisection function
function [X,it]= bisectionMethod(f, a, b, tol)
N = ((log10(b-a)-log10(tol))/log10(2))-1;
N = ceil(N);
% check validity
if f(a)*f(b)>0
    disp('f(a)*f(b)>0!'),
    disp('No root available in the interval')
    X = NaN;
    return
end
it=0;
X = zeros(1,N);
while it < N
    c = (a+b)/2;     % Callculate new root estimate
    X(it+1) = c;       % Store the new root estimate
    if f(a)*f(c)> 0 % If f(a)*f(b)>0, set a = c
        a = c;
    else             % If f(a)*f(b)<0, set b = c
        b = c;
    end
    it=it+1;         % Increament the iteration
end
end
```

## Newton's Method

```
%% Netwon Function
function [X,it] = Newton_Method(f, df, x, tol)
maxCount = 50; % Limit iterations to a maximum of 100
counter = 1 ; % Initalize the iterations counter
while abs(f(x))>tol ||maxCount<maxCount
    if df(x)==0
        disp('Division by zero encountered')
        break
    end
    x = x - f(x)/ df(x);     % Newton Raphson Formula
    X(counter) = x;          % Store approximate roots in all iterations
    counter = counter+1;     % Increament the iteration counter
end
it = counter-1; % Total Number of Iterations
end
```

## Muller's Method 1 (MM1)

```
%% Muller's Method 1
function X = MM1(P, x0, x1, x2,tol)
% Evaluate the Intial points
fx0 = polyval(P,x0);
fx1 = polyval(P,x1);
fx2 = polyval(P,x2);

iter = 1; % Start Iterations at 1
I_max = 100; % Maximum number of iterations allowed
while abs(fx2) > tol && iter <100 I_max;
    % Implement MM1 formulas
    H1 = x1 - x0;    D1 = (fx1 - fx0) / H1;
    H2 = x2 - x1;    D2 = (fx2 - fx1) / H2;

    % Calculate interpolation polynomial constants
    C = (D2 - D1)/(H1 + H2);
    B = D2 + H2*C;
    A = fx2;
    Den = sqrt(B*B + 4*A*C);
    x0 = x1; x1 = x2;

    % Get the minimum of the roots
    if abs(B + Den)> abs(B - Den)
        x2 = x2 - 2*A/(B + Den);
    else
        x2 = x2 - 2*A/(B - Den);
    end

    % Update the variables
    X(iter) = x2;
    fx0 = fx1;fx1 = fx2;
    fx2 = polyval(P,x2);
    iter =iter +1;
end
end
```

## Muller's Method 2 (MM2)

```
%% Function for Muller's Method 2
function X= MM2(P, dP, dP2, x, tol)
% This method requires a single intial point
C = polyval(P,x);
count = 1;
% Implement muller's method 2
while abs(C) > tol && count<100
    A = 0.5 * polyval(dP2,x);
    B = polyval(dP,x);
    D = sqrt(B * B - 4 * A * C);

    % Check and use maximum term of denominator
    if abs(B + D) > abs(B - D)
        x = x - 2 * C / (B + D);
    else
        x = x - 2 * C / (B - D);
    end

    X(count) = x;
    C = polyval(P,x);
end
end
```

## Laguerre's method

```
%% Laguerre's method
function X = laguerre_Method(P, dP, dP2, x, tol)
% This method requires a single initial guess and the first and second
% derivatives of the polynomial
n = length(P) - 1;
Pxi = polyval(P,x);
count = 1; % Initialize Iterations Counter
% Implement the laguerre Method
while  abs(Pxi) > tol&& count<100
    G = polyval(dP,x)/Pxi;
    G2 = G*G;
    H = G2 - polyval(dP2,x)/Pxi;
    C = sqrt((n - 1)*(n*H - G2));
    % Evaluate for the smallest root
    if abs(G - C)> abs(G + C)
        x = x - (n/(G - C));
    else
        x = x - (n/(G + C));
    end
    % Update the variables
    X(count) = x;
    Pxi = polyval(P,x);
end
end
```