Numerical Methods

Project Number: B9

Name Surname: Beste Baydur

Index Number: 295597

Winter Semester 2020

Finding Zeros of Function

Finding zeros is finding solutions of nonlinear equations with iterative methods. In order to find the root, first, an interval should be identified where the root exists. This process is called root bracketing. We can do root bracketing with two methods: Interactive user-computer process and algorithmic approach. Interactive user-computer process is where computer draws the approximate graph of the function and user selects the interval or intervals that roots are located. When we apply algorithmic approach (without user interaction), then we check if a function f(x) is continuous on a closed interval [a,b] and f(a).f(b)<0. In case that both conditions are met, then at least one root of f(x) is located within [a,b]. If the root is not included due to such a condition like f(a).f(b)>0, interval should be extended.

When we find the root interval, by using iterative methods we can find the root. I am going to find the roots of the function $f(x) = 1.4\sin(x)-e^x+6x-0.5$ over intervals [-5,5], which is given in the task. I will use Secant and Newton's methods to find the roots.

Secant Method

In this method, a secant line joins the two last obtained points which are denoted x_{n-1} and x_n. It may happen that a next interval does not contain the root. A new point in the method is defined by the formula:

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

The order of convergence is $p = (1+\sqrt{5})/2 \approx 1.618$ so it makes the method better convergent compared to some of other methods such as bisection and regula falsi. But since it is locally convergent, a lack of convergence may happen in case that initial isolation interval of root is not small enough.

Newton's Method

Newton's method (Tangent Method) operated by approximation of the function f(x) by the first order part of its expansion into a Taylor series at a current point x_n which is the current approximation of the root
The approximation:

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

The next point x_{n+1} results as a root of the function above:

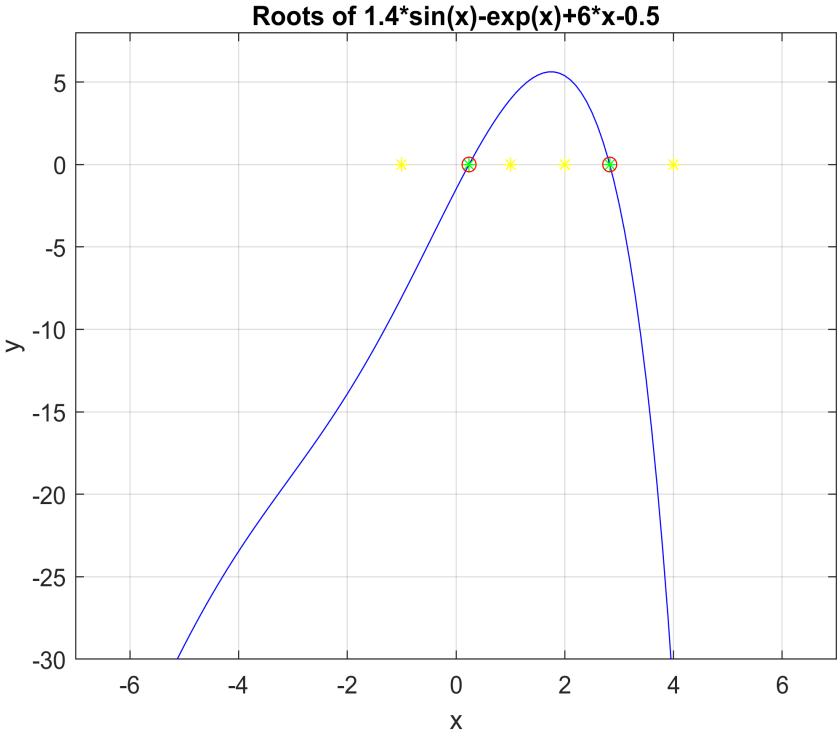
$$f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0,$$

And the iteration formula is:

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

When an initial point is not in the root's set of attraction it can diverge but in case that it doesn't, it converges, and it is fast (quadratic convergence). It can have numerical errors when the derivative is close to zero due to its sensitivity. This method is much faster than the Secant method, but it also requires more information, i.e., it needs the function's derivative when the Secant method doesn't.

Results



Result of running Newton's method(green) Secant method(red circle) with initial points(yellow)

```
Secant method:
[-1, 1]
Iteration 1: 0.340351
Iteration 2: 0.22162
Iteration 3: 0.240002
Iteration 4: 0.239749
Iteration 5: 0.239748
Iteration 6: 0.239748
Number of iterations: 6

root =

   0.239747976597135
[0,2]:
Iteration 1: 0.435796
Iteration 2: 0.00643291
Iteration 3: 0.245652
Iteration 4: 0.239903
Iteration 5: 0.239748
Iteration 6: 0.239748
Iteration 7: 0.239748
Number of iterations: 7

root=

   0.239747976597135

Newton's Method
1     0.234375
2     0.239744
3     0.239748
4     0.239748
Root:
   0.239747976597135

Iterations:
     4
1     4.730671
2     3.921258
3     3.296073
4     2.941207
5     2.835497
6     2.827091
7     2.827041
8     2.827041
Root:
   2.827040909883657
Iterations:
     8
```

Conclusions

As i mentioned before, Newton's method is faster than Secant method, and now it is proved. The required number of iterations is less for obtaining the same accuracy from both methods.

We choose the initial intervals after checking the graph of the function f(x) which was given in the task. Roots are roughly around the middle of the subintervals. Divergence can occur if the initial point is too far from the actual root therefore the proper root wouldn't be found. Both methods have their pros and cons. For secant method, since it doesn't need the function's derivative, it is useful. But it also fails to converge in case that isolation interval is too small since it is locally convergent as mentioned before. For Newton's method, it converges really fast, but it requires more information like function's derivative, and it is sensitive to numerical errors since it is also locally convergent as the Secant Method. As a conclusion, it can be said that both methods are good, and the roots establish the roots well.

Finding Roots of Polynomial

Müller's Method

It is a method that is created specially to find real and complex roots of polynomials. Method can be done using a quadratic interpolation based on three different points. Secant Method is similar to Müller's method but in Secant method there are two points where in Müller's there are three. There are two versions of Müller's Method: MM1 and MM2.

MM1

In MM1, we construct a parabola by taking three points x_0,x_1,x_2 together with corresponding polynomial values $f(x_0)$, $f(x_1)$ and $f(x_2)$. Roots of the constructed parabola is used for next approximation.
We introduce a new variable:

$$z = x - x_2$$

And use the differences:

$$z_0 = x_0 - x_2,$$
$$z_1 = x_1 - x_2.$$

The interpolating parabola defined in the variable z is considered:

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

Using three given points, we can get:

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

Roots are given by

$$\begin{aligned} z_+ &= \frac{-2c}{b + \sqrt{b^2 - 4ac}}, \\ z_- &= \frac{-2c}{b - \sqrt{b^2 - 4ac}}. \end{aligned}$$

The root with smaller absolute value should be chosen for the next step:

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

where

$$\begin{aligned} z_{min} &= z_+, \quad \text{if } |b + \sqrt{b^2 - 4ac}| \geq |b - \sqrt{b^2 - 4ac}|, \\ z_{min} &= z_-, \quad \text{in the opposite case.} \end{aligned}$$

MM2

It is almost the same as MM1 but it is slightly more numerically effective.

We have that:

$$z \overset{\text{df}}{=} x - x_k,$$

At point z=0

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

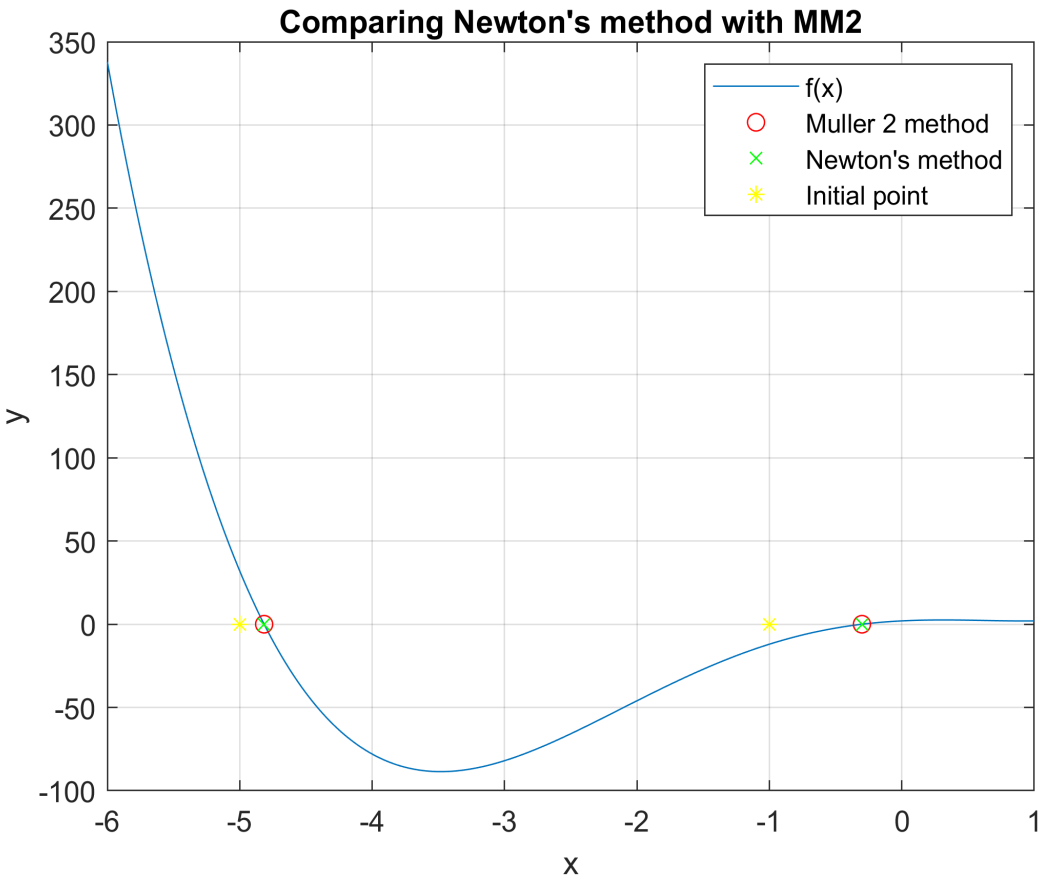
Root is calculated from formula:

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

To approximate the solution α, a root with smaller absolute value is chosen:

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

Result



```
fzero1 =

  -4.815775287862217


fzero2 =

  -0.300746614999176

Error tolerance 10^-12
Finding roots of polynomial p(x)
Coefficients of f(x) =
     1     3    -8     4     2

Coefficients of f'(x) =
     4     9   -16     4

Coefficients of f''(x) =
    12    18   -16

MM1 Method
Initial points = [-4,-4.5,-5]
Zero point, x =
  -4.815775287862217

f(x) = -4.885e-15
Number of iterations: 5

MM2 Method
Initial points = -5
Zero point, x =
  -4.815775287862217

f(x) = -4.885e-15
Number of iterations: 3

Newton's Method
Initial points = -5
1    -4.832461
2    -4.815929
3    -4.815775
4    -4.815775

Root is:
  -4.815775287862217

Number of iterations:
     4
```

```
MM1 Method
Initial points = [0,-0.5,-1]
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 7
MM2 Method
Initial points = -1
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
Newton's Method
Initial points = -1
1    -0.520000
2    -0.337358
3    -0.302090
4    -0.300749
5    -0.300747
6    -0.300747

Root is:
  -0.300746614999176

Number of iterations:
     6

MM1 Method
Initial points = [1,0.5+0.5i,0]
Zero point, x =
 -0.300746614999176 - 0.000000000000000i

f(x) = 2.2204e-16-2.3141e-21i
Number of iterations: 11
MM2 Method
Initial points = 0
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
MM1 Method
Initial points = [0,0.5-0.5i,1-1i]
Zero point, x =
  1.058260951430696 - 0.510868198395578i

f(x) = 8.8818e-16-7.7716e-16i
Number of iterations: 8
```

```
MM2 Method
Initial points = 1-1i
Zero point, x =
  1.058260951430696 - 0.510868198395578i

f(x) = -4.4409e-16-6.6613e-16i
Number of iterations: 4

Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
Newton's Method
Initial points = -1
1    -0.520000
2    -0.337358
3    -0.302090
4    -0.300749
5    -0.300747
6    -0.300747
Root:
  -0.300746614999176

Number of iterations:
     6

     1

MM1 Method
Initial points = [1,0.5+0.5i,0]
Zero point, x =
 -0.300746614999176 - 0.000000000000000i

f(x) = 2.2204e-16-2.3141e-21i
Number of iterations: 11

MM2 Method
Initial points = 0
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
     0

MM1 Method
Initial points = [0,0.5-0.5i,1-1i]
Zero point, x =
  1.058260951430696 - 0.510868198395578i
```

```
f(x) = 8.8818e-16-7.7716e-16i
Number of iterations: 8
MM2 Method
Initial points = 1-1i
Zero point, x =
  1.058260951430696 - 0.510868198395578i

f(x) = -4.4409e-16-6.6613e-16i
Number of iterations: 4
```

Conclusions

With Müller's method, both real and complex roots of polynomials can be found in a very effective way. Since the initial points were well chosen and they were all in the roots' sets of attraction, the estimation of the real roots of the polynomial with the identical initial points from MM2 method using Newton's method has given accurate results.

MM1 and MM2 methods can both find complex roots while Newton's Method cannot. But they also have disadvantages. MM1 method is less convenient to use since it requires three initial points. MM2 is better than MM1 since only one initial point should be chosen and therefore it is faster. But compared to Newton's method, there is less progress per iteration and the reason is that order of convergence is smaller. As for Newton's method, it is very fast, and the progress is considerably more because of quadratic order of convergence. But it is not able to find the complex roots.

Laguerre's Method

Laguerre's Method was also created specially for finding roots of the polynomial like Müller's method and it is one of the best methods to find them. We start applying the method by choosing an initial point x_k

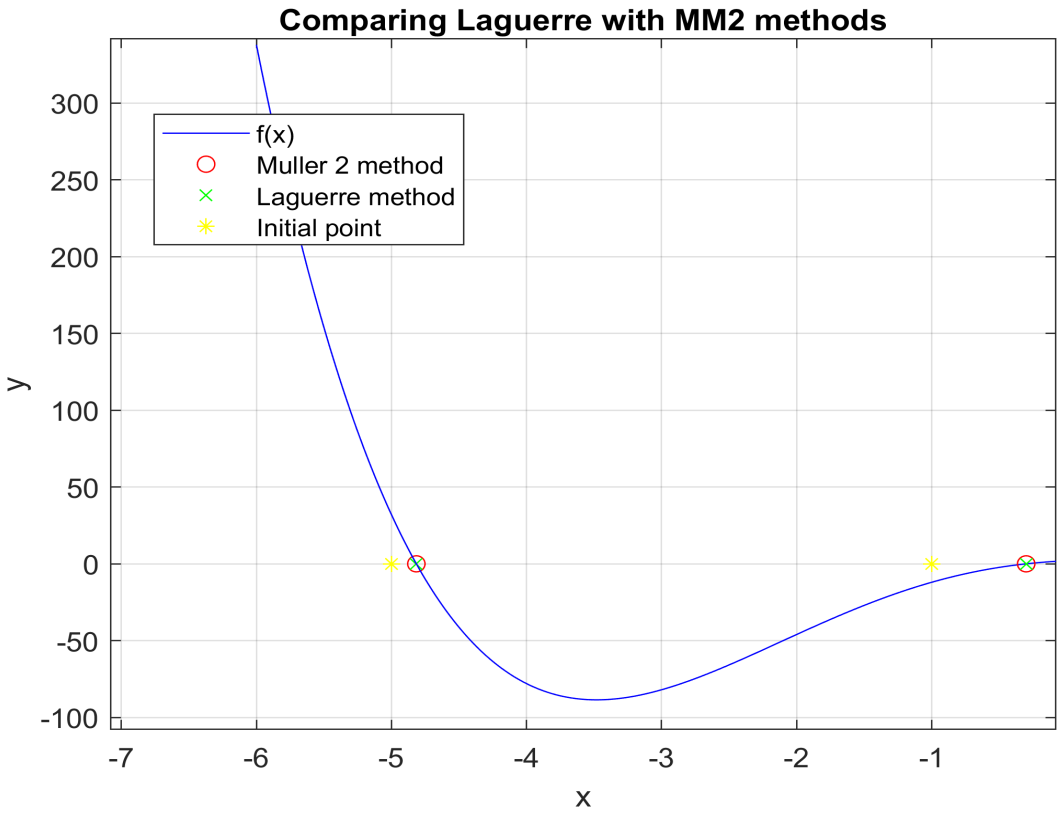
$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)(f'(x_k))^2 - nf(x_k)f''(x_k)]}}$$

Where n is the order of polynomials and the sign in the denominator is chosen in a way that the denominator's absolute value is larger.

It is comparably more complex since it takes into account the order of the polynomial. So, we can assume that it will be the best method to be used in this project, although it lacks formal analysis for a complex root case.

Laguerre's method is a globally convergent method, therefore it will converge from any initial point. When the initial point is sufficiently close to the root and it is a single root, it converges cubically and when it is a multiple root, it converges linearly

Results



```
Error tolerance is 10^-12
Searching roots of polynomial p(x)
Coefficients of f(x) =
     1     3    -8     4     2

Coefficients of f'(x) =
     4     9   -16     4

Coefficients of f''(x) =
    12    18   -16

Initial points = -5

Laguerre's method
Zero point, x =
  -4.815775287862217

f(x) = 5.385e-16
Number of iterations: 2
Iterations
  -4.815779207378283
  -4.815775287862217

MM2 method
Zero point, x =
  -4.815775287862217

f(x) = -4.885e-15
Number of iterations: 3
Initial points = -1

Laguerre's method
Zero point, x =
  -0.300746614999183

f(x) = 6.554e-16
Number of iterations: 3
Iterations
  -0.356293408063539
  -0.300779706891933
  -0.300746614999183

MM2 method
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
Initial points = 0
```

```
Laguerre's method
Zero point, x =
  -0.300746614999176

f(x) = 2.2931e-16
Number of iterations: 3
Iterations
  -0.296535165408627
  -0.300746600718369
  -0.300746614999176

MM2 method
Zero point, x =
  -0.300746614999176

f(x) = -4.4409e-16
Number of iterations: 3
Initial points = 1-1i

Laguerre's method
Zero point, x =
  1.058260951430696 - 0.510868198395578i

f(x) = 1.4690e-16-1.4212e-16i
Number of iterations: 3
Iterations
  1.062323601629286 - 0.518572484425081i
  1.058261056870996 - 0.510868096342462i
  1.058260951430696 - 0.510868198395578i

MM2 method
Zero point, x =
  1.058260951430696 - 0.510868198395578i

f(x) = -4.4409e-16-6.6613e-16i
Number of iterations: 4
```

Conclusion

When we compare MM2 and Laguerre's Method we can see that the results are same (with the same initial points), and Laguerre's Method is quicker. Values of complex roots are also found faster in Laguerre's method. Compared to all the methods that were used in this project, Laguerre was the most reliable and fast one. It also has a disadvantage that it needs a lot of information. But still it is the best method out of all I have used in this project.

Appendix

Task1

```
clear all
clc
f=@ 1.4*sin(x)-exp(x)+6*x-0.5;
x0=-5;
x1=5;
epsilon= 0.000000001;
err=abs(x1-x0);

[roots1, iterations1, deltas1, actuals1, criterion1] =
secant(f);
[roots2, iterations2, deltas2, actuals2, criterion2] =
newtons(f, f_deriv);

disp('solutions secant');
disp(roots1);

disp('solutions newtons');
disp(roots2);

figure(1)
semilogx(deltas1, iterations1, 'r', deltas2, iterations2,
'b');
legend('secant', 'newtons');
xlabel('deltas');
ylabel('iterations');

figure(2)
loglog(deltas1, actuals1, 'b', deltas1, criterion1, 'r');
title('secant method');
legend('actual absolute error vs tolerance', 'criterion value
vs tolerance');
xlabel('tolerance');
ylabel('abs error & criterion');

figure(3)
loglog(deltas2, actuals2, 'b', deltas2, criterion2, 'r');
title('newton method');
legend('actual absolute error vs tolerance', 'criterion value
vs tolerance');
xlabel('tolerance');
ylabel('abs error & criterion');

figure(4)
```

```
x=linspace(2,12);
root = fzero(f,7);
y = 0.3.*x.*sin(x)-log(x+1);
plot(x,y)
hold on
plot(root,0,'r*','displayName','Root');
title('given function');
xlabel('x');
ylabel('y');
```

Secant Method

```
function [roots, iterations, deltas, actuals, criterion] =
secant(f)
delta = 10^(-3);
x0 = -5;
x1 = 5;
x_abs = fzero(f, 7);

deltas = [];
iterations = [];
roots = [];
actuals = [];
criterion = [];



while delta > 10^(-17)

    for n=1:1000
        f0 = f(x0);
        f1 = f(x1);
        x2 = x1 - ((x1-x0)/(f1-f0))*f1;

        if abs(x2-x1) < delta
            break;
        end

        x0 = x1;
        x1 = x2;



    end
```

```
    deltas = [deltas, delta];
    iterations = [iterations, n];
    roots = [roots, x2];
    if abs(x2-x_abs) == 0
        actuals = [actuals, 10^(-16)];
    else
        actuals = [actuals, abs(x2-x_abs)];
    end
    criterion = [criterion, abs(x2-x1)];

    x0 = 2;
    x1 = 12;

    delta = delta*0.1;
end



end
```

Newton's Method

```
function [roots, iterations, deltas, actuals, criterion] =
newtons(f, f_deriv)
delta = 10^(-3);
x0 = 2;
x1 = 12;
x_abs = fzero(f,7);

deltas = [];
iterations = [];
roots = [];
actuals = [];
criterion = [];

while delta > 10^(-17)
    for n = 1:1000

        f0 = f(x0);
        f0_deriv = f_deriv(x0);

        x2 = x0-f0/f0_deriv;



        if abs(x2-x0) < delta
            break;
        end
```

```
        x0 = x2;

    end

    deltas = [deltas, delta];
    iterations = [iterations, n];
    roots = [roots, x2];
    if x2-x_abs == 0
        actuals = [actuals, 10^(-16)];
    else
        actuals = [actuals, abs(x2-x_abs)];
    end
    criterion = [criterion, abs(x2-x0)];



    delta = delta * 0.1;
    x0 = 2;


end
end
```

MM1

```
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);
23
 %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

```
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
24
error('Number of iterations exceeded');
```

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
25
 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
```