ENUME Project B

No. 33

Fengxi Zhao

317342

December 2023

Task 1:

I Find all zeros of the function

$$f(x) = 7.2 - 2x - 5e^{-x}$$

in the interval [–5, 10] using:
a) the secant method,
b) the Newton's method.

Algorithm:

Secant method begins with an interval. We substitute the old interval with smaller one for each iteration until the error is neglectable. Just need to use this formula from the textbook:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} = \frac{x_{n-1}f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}. \qquad (6.6)$$

Notice it's local convergent. So, picking the interval is important.

Newton's method needs to find the derivative first, then use this formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \qquad (6.8)$$

Notice that the error will be huge if the derivative is close to 0.

Result:

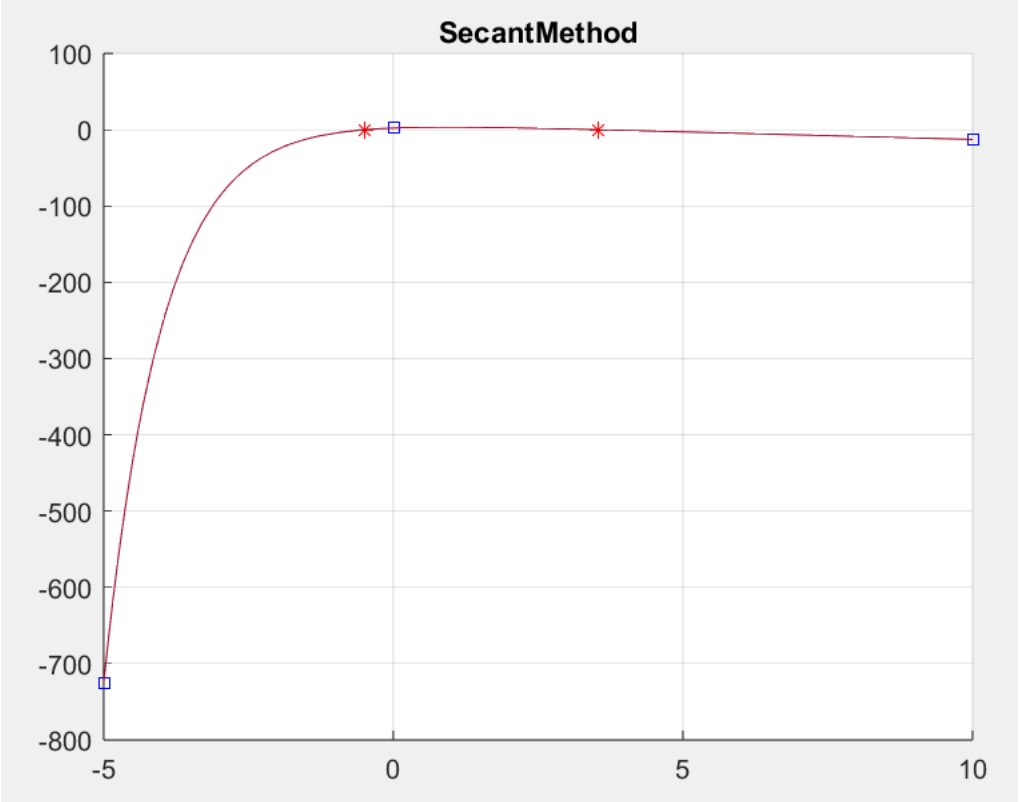
(a) Secant method

For the first root:

Iteration	Answer	Error
1	-0.0151	0.0151
2	-0.7242	0.709
3	-0.4149	0.3093
4	-0.4816	0.0668
5	-0.4936	0.0119
6	-0.4930	0.00060849
7	-0.4930	4.5467e-06
8	-0.4930	1.8172e-09
9	-0.4930	5.6066e-15
10	-0.4930	2.7756e-16
11	-0.4930	5.5511e-17

For the second root:

Iteration	Answer	Error
1	1.4666	8.5334
2	3.1361	1.6694
3	3.6298	0.4938
4	3.5247	0.1051
5	3.5265	0.0018
6	3.5265	7.0912e-06
7	3.5265	5.0544e-10
8	3.5265	0



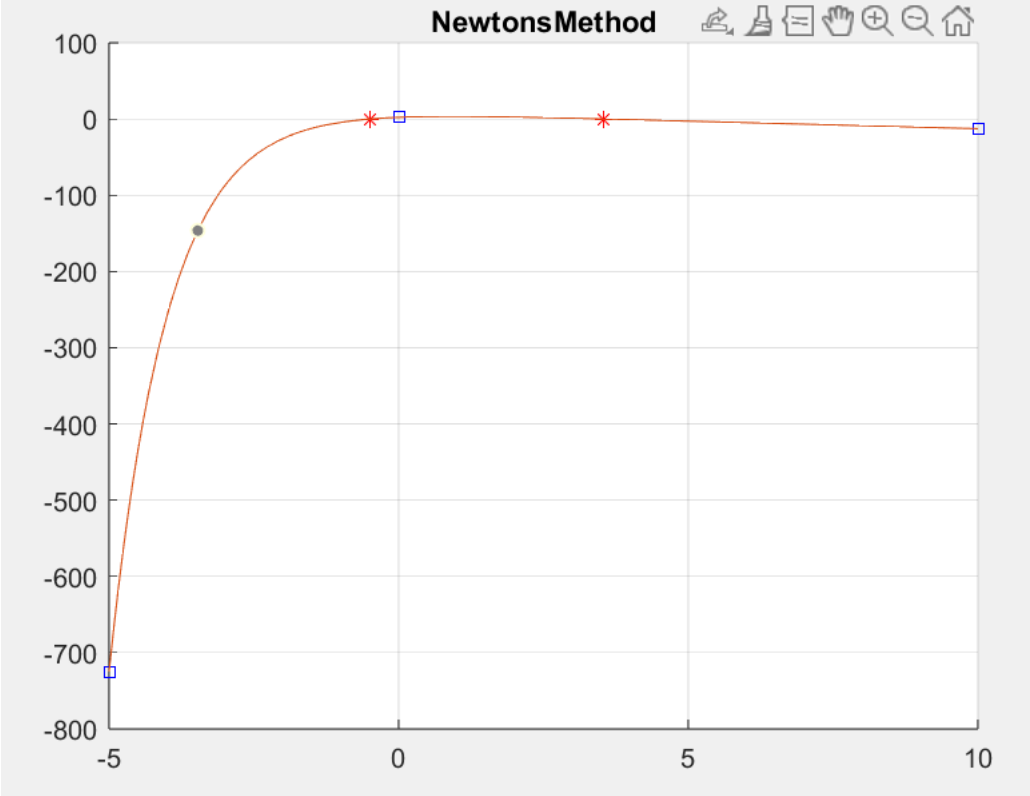
(b) Newton's method

For the first root:

Iteration	Answer	Error
1	-1.6731	0.8269
2	-1.0199	0.6532
3	-0.6301	0.3898
4	-0.5044	0.1257
5	-0.4931	0.0113
6	-0.4930	8.5633e-05
7	-0.4930	4.8522e-09
8	-0.4930	0

For the second root:

Iteration	Answer	Error
1	3.5589	1.4411
2	3.5265	0.0324
3	3.5265	4.0661e-05
4	3.5265	6.5597e-11
5	3.5265	0



Discussion and comparison:

The result shows that Newton's method is faster than secant method.

Fewer iterations are required to get the acceptable results.

Task 2:

II Find all (real and complex) roots of the polynomial

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

using the Müller's method implementing both the MM1 and MM2 versions. Compare the results. Find also real roots using the Newton's method and compare the results with the MM2 version of the Müller's method (using the same initial points).

Algorithm:

For MM1, choose three points close to each root, then find

$$z = x - x_2$$

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

The interpolating parabola defined in the variable z is considered as

$$y(z) = az^2 + bz + c. \qquad (6.32)$$

Now we need to find a, b and c

$$az_0^2 + bz_0 + c = y(z_0) = f(x_0),$$
$$az_1^2 + bz_1 + c = y(z_1) = f(x_1),$$
$$c = y(0) = f(x_2).$$

Find two roots for 6.32

$$z_+ = \frac{-2c}{b + \sqrt{b^2 - 4ac}}, \qquad (6.35)$$

$$z_- = \frac{-2c}{b - \sqrt{b^2 - 4ac}}. \qquad (6.36)$$

Choose the one with smaller absolute value to be zmin, then update the answer

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

The iteration stops when c, which is y(0) close to 0 enough.

For MM2, it is easier to find a, b and c by using

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

The rest is more or less the same. Calculate roots with

$$z_{+,-} = \frac{-2f(x_k)}{f'(x_k) \pm \sqrt{(f'(x_k))^2 - 2f(x_k)f''(x_k)}}. \qquad (6.37)$$

Choose the one with smaller absolute value to be zmin, then update the answer

$$x_{k+1} = x_k + z_{min}, \qquad (6.38)$$

The iteration stops when c, which is y(0) close to 0 enough.

Result:

(a) MM1

For the first root (real):

Iteration	Argument	Function value
1	-3.3512	-0.4019
2	-3.3434	-0.0047
3	-3.3433	1.2938e-06
4	-3.3433	2.0006e-13
5	-3.3433	2.3315e-15
6	-3.3433	2.3315e-15

For the second root (real):

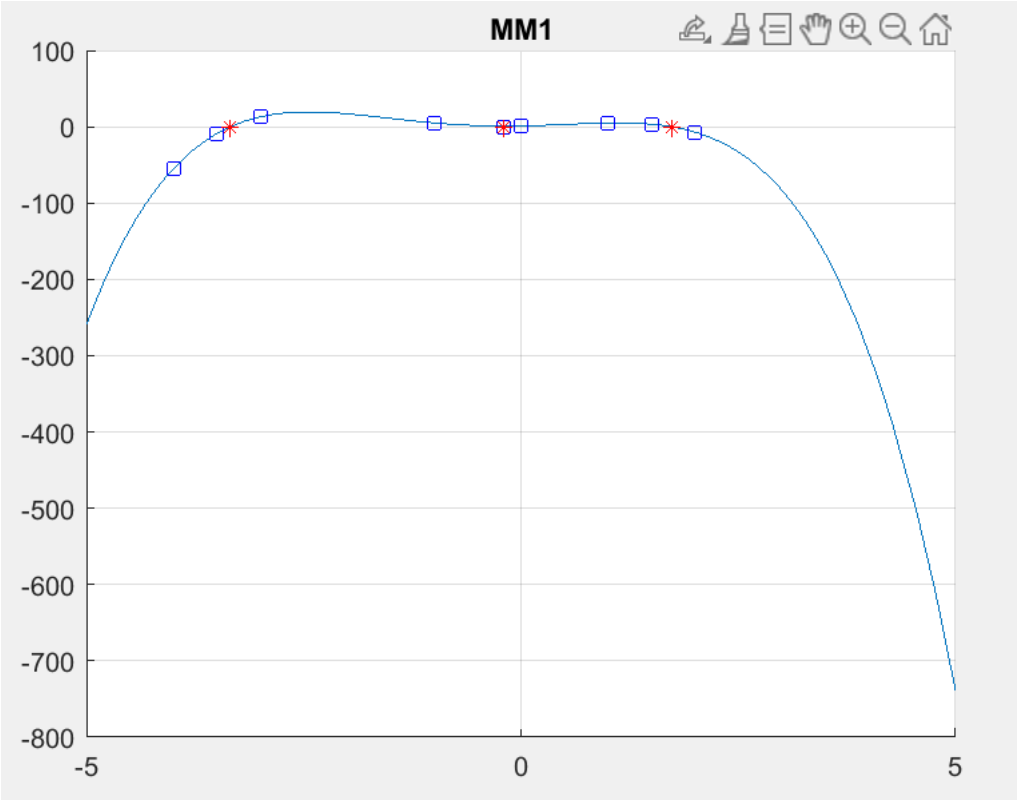
Iteration	Argument	Function value
1	1.7169	0.3617
2	1.7352	0.0102
3	1.7357	-2.0126e-05
4	1.7357	8.9698e-11
5	1.7357	0

For the third root (complex):

Iteration	Argument	Function value
1	-0.1703 + 0.3327i	0.1538 + 0.0962i
2	-0.1946 + 0.3708i	-0.0221 + 0.0089i
3	-0.1962 + 0.3658i	1.6313e-04 - 1.3292e-05i
4	-0.1962 + 0.3658i	1.3907e-08 - 5.3616e-09i
5	-0.1962 + 0.3658i	4.4409e-16 - 9.9920e-16i
6	-0.1962 + 0.3658i	0

For the fourth root (complex):

Iteration	Argument	Function value
1	-0.1703 - 0.3327i	0.1538 - 0.0962i
2	-0.1946 - 0.3708i	-0.0221 - 0.0089i
3	-0.1962 - 0.3658i	1.6313e-04 + 1.3292e-05i
4	-0.1962 - 0.3658i	1.3907e-08 + 5.3616e-09i
5	-0.1962 - 0.3658i	4.4409e-16 + 9.9920e-16i
6	-0.1962 - 0.3658i	0



(b) MM2

For the first root (real):

Iteration	Argument	Function value
1	-3.3424	0.0463
2	-3.3433	-8.5487e-09
3	-3.3433	2.3315e-15

For the second root (real):

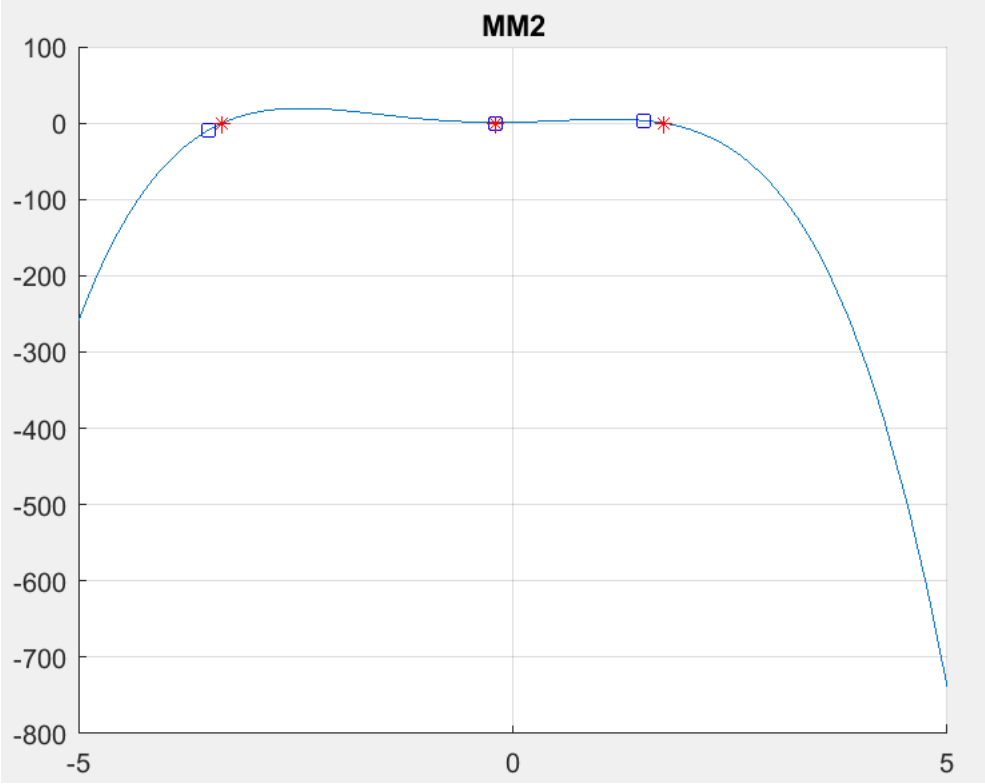
Iteration	Argument	Function value
1	1.7416	-0.1162
2	1.7357	1.8195e-06
3	1.7357	0

For the third root (complex):

Iteration	Argument	Function value
1	-0.1967 - 0.3647i	0.0048 + 0.0026i
2	-0.1962 - 0.3658i	1.3361e-09 - 3.0694e-09i
3	-0.1962 - 0.3658i	1.1102e-16 - 5.5511e-17i
4	-0.1962 - 0.3658i	0

For the fourth root (complex):

Iteration	Argument	Function value
1	-0.1967 + 0.3647i	0.0048 - 0.0026i
2	-0.1962 + 0.3658i	1.3361e-09 + 3.0694e-09i
3	-0.1962 + 0.3658i	1.1102e-16 + 5.5511e-17i
4	-0.1962 + 0.3658i	0



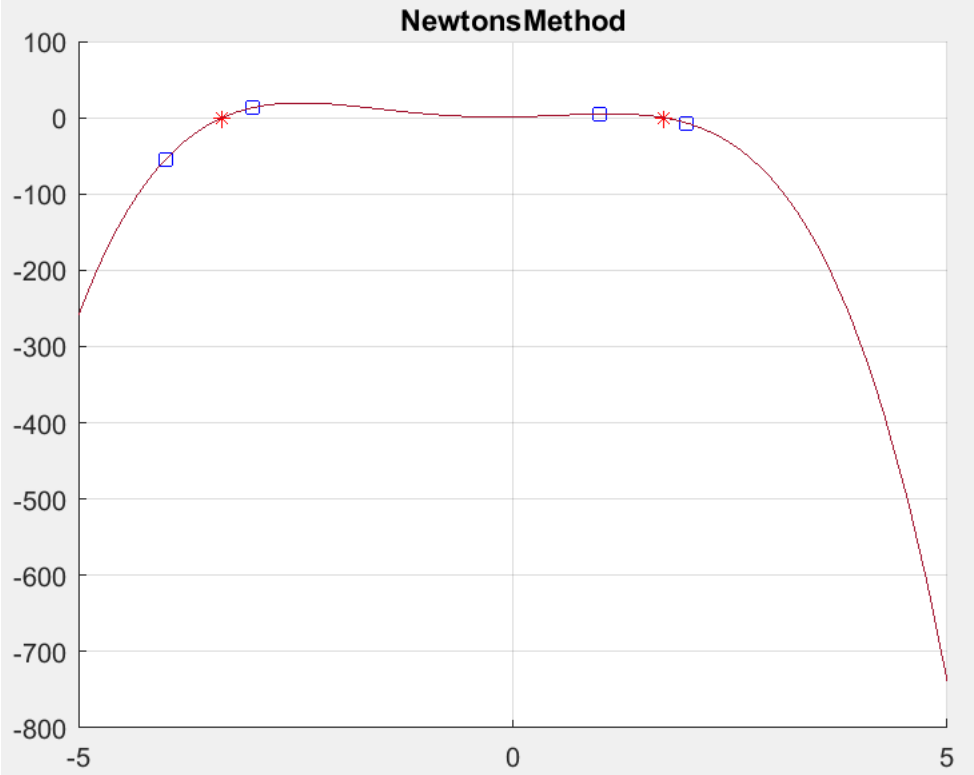
(c) Newton's method

For the first root (real):

Iteration	Answer	Error
1	-3.3606	0.1394
2	-3.3436	0.0170
3	-3.3433	2.4007e-04
4	-3.3433	4.7491e-08
5	-3.3433	1.3323e-15

For the second root (real):

Iteration	Answer	Error
1	1.8438	0.3438
2	1.7476	0.0962
3	1.7359	0.0117
4	1.7357	1.6530e-04
5	1.7357	3.2693e-08
6	1.7357	1.3323e-15



Discussion and comparison:

MM2 is faster than MM1, for both real roots and complex roots. I guess it is because I calculated two derivatives for the formula before the loop begins, so unlike three points in MM1, fewer arguments need to be deal with.

MM2 is faster than Newton's method. Moreover, Newton's method can not calculate complex roots. I suppose that with "normal" polynomials we should use MM2 since the second derivative is easy to get. For other cases (don't need complex roots, hard to calculate the second derivatives) Newton's method should work better.

Task 3:

III Find all (real and complex) roots of the polynomial f(x) from II using the Laguerre's method. Compare the results with the MM2 version of the Müller's method (using the same initial points).

Algorithm:

Use this formula:

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) \pm \sqrt{(n-1)[(n-1)(f'(x_k))^2 - nf(x_k)f''(x_k)]}}, \quad (6.39)$$

The ending condition is the same: when c is close enough to 0.

Result:

For the first root (real):

Iteration	Argument	Function value
1	-3.3434	-5.7096e-04

For the second root (real):

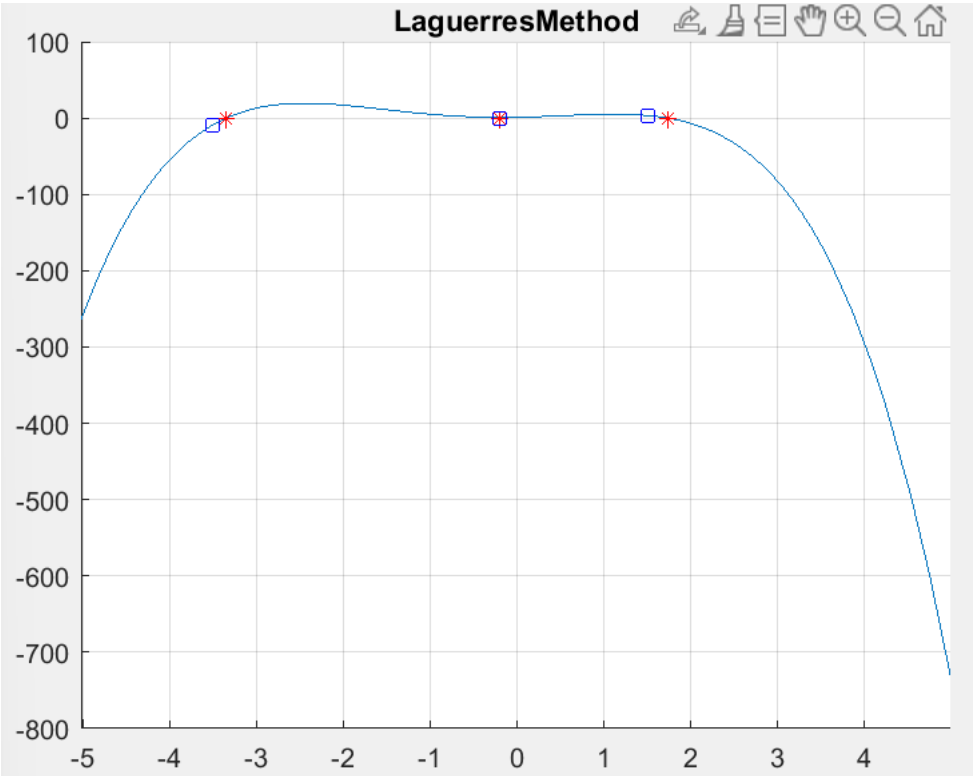
Iteration	Argument	Function value
1	1.7354	0.0064
2	1.7357	1.4925e-11
3	1.7357	0

For the third root (complex):

Iteration	Argument	Function value
1	-0.1963 - 0.3664i	-0.0026 + 0.0003i

For the fourth root (complex):

Iteration	Argument	Function value
1	-0.1963 + 0.3664i	-0.0026 - 0.0003i



Discussion and comparison:

Laguerre's method is significantly faster than MM2, and it's globally convergent. Laguerre's method involves more complex calculations per iteration compared to MM2. However, the faster convergence rate of Laguerre's method often compensates for this increased complexity, leading to quicker overall performance in finding the root.

Reference:

1. *Numerical Methods* by Piotr Tatjewski

2. https://youtu.be/XIIEjwtkONc

Code:

Task 1:

Main:

```
% my function
f = @(x)-2*x-5*exp(-x)+7.2;

disp("SecantMethod:");
% two intervals
disp("First root:");
[num_iteration_1, root_1] = SecantMethod(f,-5,0);
disp("Second root:");
[num_iteration_2, root_2] = SecantMethod(f,0,10);

% plot
hold on;
fplot(f,[-5, 10]);
title("SecantMethod");
hold off;

disp("SecantMethod:");
disp(root_1);
disp(root_2);


figure(2);
disp("NewtonsMethod:");
% two intervals
disp("First root:");
[num_iteration_3, root_3] = NewtonsMethod(f,-5,0,1);
disp("Second root:");
[num_iteration_4, root_4] = NewtonsMethod(f,0,10,1);

% plot
hold on;
fplot(f,[-5, 10]);
title("NewtonsMethod");
hold off;

disp("NewtonsMethod:");
disp(root_3);
disp(root_4);
```

SecantMethod:

```
function [num_iter, ans] = SecantMethod(f,x0,x1)

    num_iter = 1;

    hold on;
    grid on;

    plot(x0, f(x0), 'bs');
    plot(x1, f(x1), 'bs');

    % max number of iterations is 100
    for i = 1:100

        x2=(x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));

        x0 = x1;
        x1 = x2;

        error = abs(x1 - x0);

        ans = x2;

        disp("Iteration number, answer, error:");
        disp(num_iter);
        disp(x2);
        disp(error);

        % if error is small enough
        if error < eps
            break;
        end

        num_iter = num_iter + 1;
    end

    % plot the answers
    plot(ans, f(ans), 'r*');
    hold off;
end
```

NewtonsMethod:

```
function [num_iter, ans] = NewtonsMethod(f,x0,x1,task)

    num_iter = 1;

    hold on;
    grid on;

    plot(x0, f(x0), 'bs');
    plot(x1, f(x1), 'bs');

    ans = (x0 + x1)/2;

    % find derivatives for both tasks
    if task == 1
        df = @(x)5*exp(-x)-2;
    end

    if task == 2
        df = @(x)(-4)*x^3-6*x^2+10*x+2;
    end

    % max number of iterations is 100
    for i=1:100

        % formula from textbook
        x1 = ans;
        ans = ans - (f(ans)/df(ans));

        error = abs(ans - x1);

        disp("Iteration number, answer, error:");
        disp(num_iter);
        disp(ans);
        disp(error);

        if error < eps
            break;
        end

        num_iter = num_iter + 1;
    end
```

```
    % plot the answers
    plot(ans, f(ans), 'r*');
    hold off;
end
```

Task 2:

Main:

```
% my function
f = @(x)(-1)*x^4 + (-2)*x^3 + 5*x^2 + 2*x + 1;
p = [-1 -2 5 2 1];

% MM1 method
hold on;
grid on;
fplot(f);

% first root(real)
x0 = -4;
x1 = -3.5;
x2 = -3;
[num_iteration_1, root_1] = MM1(p, x0, x1, x2);

plot(x0,f(x0),'bs');
plot(x1,f(x1),'bs');
plot(x2,f(x2),'bs');
plot(root_1,f(root_1), 'r*');


% second root(real)
x0 = 1;
x1 = 1.5;
x2 = 2;
[num_iteration_2, root_2] = MM1(p, x0, x1, x2);

plot(x0,f(x0),'bs');
plot(x1,f(x1),'bs');
plot(x2,f(x2),'bs');
plot(root_2,f(root_2), 'r*');
```

```
% third root(complex)
x0 = -1;
x1 = -0.2-0.5i;
x2 = 0;
[num_iteration_3, root_3] = MM1(p, x0, x1, x2);

plot(x0,f(x0),'bs');
plot(x1,f(x1),'bs');
plot(x2,f(x2),'bs');
plot(root_3,f(root_3), 'r*');

% fourth root(complex)
x0 = -1;
x1 = -0.2+0.5i;
x2 = 0;
[num_iteration_4, root_4] = MM1(p, x0, x1, x2);

plot(x0,f(x0),'bs');
plot(x1,f(x1),'bs');
plot(x2,f(x2),'bs');
plot(root_4,f(root_4), 'r*');

title("MM1");
hold off;

disp(root_1);
disp(root_2);
disp(root_3);
disp(root_4);

% MM2 method
figure(2);
hold on;
grid on;
fplot(f);

% first root(real)
x = -3.5;
[num_iteration_5, root_5] = MM2(p, x);

plot(x,f(x),'bs');
plot(root_5,f(root_5), 'r*');
```

```
% second root(real)
x = 1.5
[num_iteration_6, root_6] = MM2(p, x);

plot(x,f(x),'bs');
plot(root_6,f(root_6), 'r*');

% third root(complex)
x = -0.2-0.5i;
[num_iteration_7, root_7] = MM2(p, x);

plot(x,f(x),'bs');
plot(root_7,f(root_7), 'r*');

% fourth root(complex)
x = -0.2+0.5i;
[num_iteration_8, root_8] = MM2(p, x);

plot(x,f(x),'bs');
plot(root_8,f(root_8), 'r*');

title("MM2");
hold off;

disp(root_5);
disp(root_6);
disp(root_7);
disp(root_8);

% Newton's method
figure(3);
[num_iteration_9, root_9] = NewtonsMethod(f, -4, -3, 2);
[num_iteration_10, root_10] = NewtonsMethod(f, 1, 2, 2);

hold on;
fplot(f);
title("NewtonsMethod");
hold off;
disp(root_9);
disp(root_10);
```

MM1:

```
function [num_iter, x3] = MM1(p,x0,x1,x2)

    num_iter = 1;

    % c is y(0). When it comes small enough, end the loop
    c = 1;

    while num_iter < 100 && abs(c) > 1e-14
        z0 = x0 - x2;
        z1 = x1 - x2;

        % find a,b according to the formula
        c = polyval(p,x2);
        a = (polyval(p,x1)*z0 - c*z0 - polyval(p,x0)*z1 + c*z1)/(z1^2*z0 -
z0^2*z1);
        b = (polyval(p,x0) - c - a*z0^2)/z0;

        delta = b^2-4*a*c;

        % choose the smaller one to be zmin
        if abs(b + sqrt(delta)) >= abs(b - sqrt(delta))
            zmin = (-2*c)/(b + sqrt(delta));
        else
            zmin = (-2*c)/(b - sqrt(delta));
        end

        % x3 is the latest root
        x3 = x2 + zmin;

        % length between x3 and others
        length_x0 = abs(x0 - x3);
        length_x1 = abs(x1 - x3);
        length_x2 = abs(x2 - x3);

        % x2 is the root from last iteration
        % we use it to replace the not nearer one: x1 or x0
        if length_x2 >= length_x1 && length_x2 >= length_x0
            x2 = x3;
        end

        if length_x1 >= length_x2 && length_x1 >= length_x0
            x1 = x2;
```

```
            x2 = x3;
        end

        if length_x0 >= length_x2 && length_x0 >= length_x1
            x0 = x2;
            x2 = x3;
        end

        disp("Iteration number, argument, function value:");
        disp(num_iter);
        disp(x3);
        disp(polyval(p,x3));

        num_iter = num_iter+1;
    end
end
```

MM2:

```
function [num_iter, ans] = MM2(p,ans)

    num_iter = 1;

    % find derivatives
    dp = polyder(p);
    d2p = polyder(dp);

    % c is y(0). When it comes small enough, end the loop
    c = 1;

    while num_iter < 100 && abs(c) > 1e-14

        % find a,b according to the formula
        c = polyval(p,ans);
        b = polyval(dp,ans);
        a = polyval(d2p,ans)/2;

        delta = b^2-4*a*c;

        % choose the smaller one to be zmin
        if abs(b + sqrt(delta)) >= abs(b - sqrt(delta))
            zmin = (-2*c)/(b + sqrt(delta));
        else
```

```
            zmin = (-2*c)/(b - sqrt(delta));
        end

        ans = zmin + ans;

        disp("Iteration number, argument, function value:");
        disp(num_iter);
        disp(ans);
        disp(polyval(p,ans));

        num_iter = num_iter+1;
    end
end
```

Task 3:

Main:

```
% my function
f = @(x)(-1)*x^4 + (-2)*x^3 + 5*x^2 + 2*x + 1;
p = [-1 -2 5 2 1];

% Laguerre's method
hold on;
grid on;
fplot(f);

% first root
x = -3.5;
[num_iter_1, root_1] = LaguerresMethod(p, x);
plot(root_1, f(root_1), 'r*');
plot(x, f(x), 'bs');

% second root
x = 1.5;
[num_iter_2, root_2] = LaguerresMethod(p, x);
plot(root_2, f(root_2), 'r*');
plot(x, f(x), 'bs');

% third root
x = -0.2-0.5i;
```

```
[num_iter_3, root_3] = LaguerresMethod(p, x);
plot(root_3, f(root_3), 'r*');
plot(x, f(x), 'bs');

% fourth root
x = -0.2+0.5i;
[num_iter_4, root_4] = LaguerresMethod(p, x);
plot(root_4, f(root_4), 'r*');
plot(x, f(x), 'bs');

title("LaguerresMethod");
hold off;
```

LaguerresMethod:

```
function [num_iter, ans] = LaguerresMethod(p,ans)

    num_iter = 1;

    % find derivatives
    dp = polyder(p);
    d2p = polyder(dp);

    % n denotes the order of the polynomial
    n = length(p) - 1;

    % c is y(0). When it comes small enough, end the loop
    c = 1;

    while num_iter < 100 && c > 1e-14

        c = polyval(p,ans);

        denominator1 = polyval(dp,ans) + sqrt((n-1)*((n-
1)*(polyval(dp,ans)^2)-n*polyval(p,ans)*polyval(d2p,ans)));
        denominator2 = polyval(dp,ans) - sqrt((n-1)*((n-
1)*(polyval(dp,ans)^2)-n*polyval(p,ans)*polyval(d2p,ans)));

        % larger denominator means less distance
        if abs(denominator1) > abs(denominator2)
            denominator = denominator1;
        else
            denominator = denominator2;
```

```
        end

        ans = ans - (n*polyval(p,ans))/denominator;

        disp("Iteration number, argument, function value:");
        disp(num_iter);
        disp(ans);
        disp(polyval(p,ans));

        num_iter = num_iter + 1;
    end
end
```