Damian Bukowski

303828

Project 2

Task 1

In this task We are supposed to find real roots of a function over a given interval using the bisection method and Newton's method. Firstly, in the bisection method we calculate the middle point of the interval (using formula $c_n = \frac{a_n + b_n}{2}$) and evaluate the value of the function at the new interval. With that value, we evaluate products $f(a_n) * f(c_n)$ and $f(b_n) * f(c_n)$, then choose the next interval as the subinterval corresponding to the result with the negative value. *n* depicts the n-th iteration. The endpoints of the new interval are a_{n+1} and b_{n+1}. We repeat the algorithm until the solution $f(x_{n+1})$ is less or equal than the assumed accuracy. In my case, the function is:

$$f(x) = 3.1 - 3 * x - e^{-x}$$

And the interval is [-5, 10]. In this case, the function looks like below:

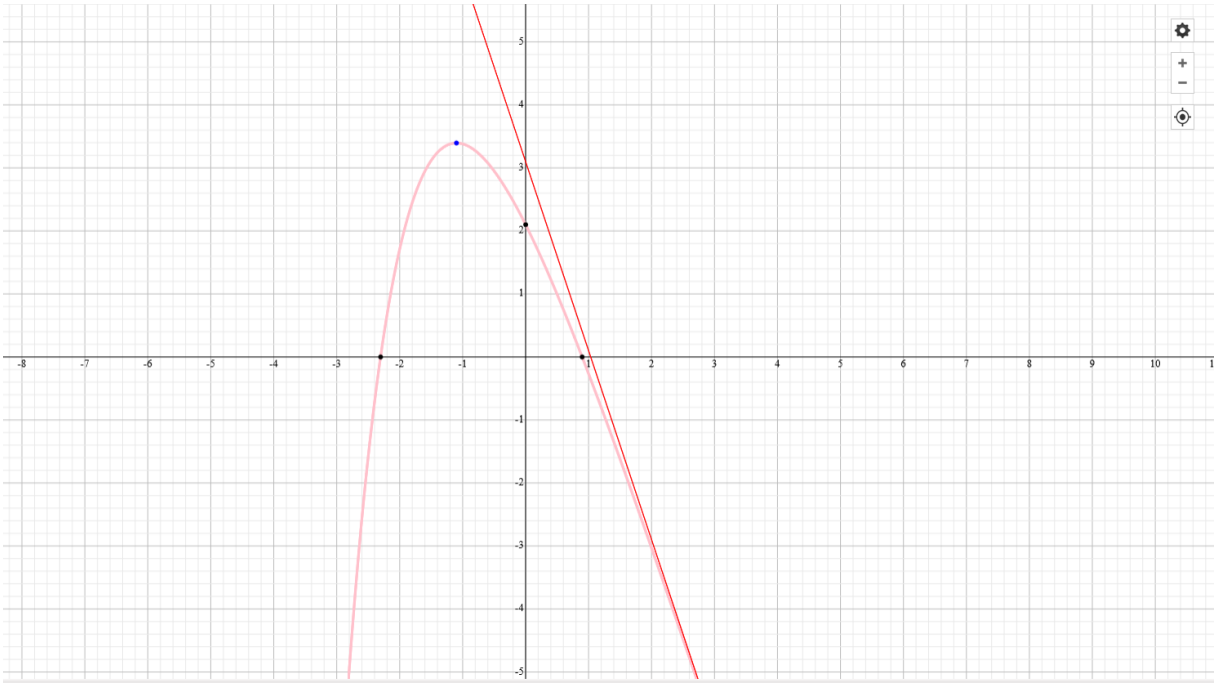


Fig 1 Used Symbolab to draw the graph, the red line is an asymptote

As can be seen, this function isn't exactly "flat" so there is no need to calculate the length of the interval. The accuracy of this method Is dependent solely on the number of iterations. Hence the fact that it's linearly convergent and the convergence factor is $\frac{1}{2}$.

The second method is the Newton's method. It works by approximating the function f(x) by the first order part of its expansion into a Taylor series at the current approximation of the root.

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

The next point x_{n+1} Is the result of the linear function:

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

Which leads to the formula:

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

As opposed to the bisection method, Newton's method is locally convergent. This means that if the initial guess is too far from the actual root the method might be divergent. On the other hand, if the method converges than it's faster than bisection method, since it has quadratic convergence (instead of linear). Newton's method works especially well when the first derivative of the function at root is far enough from 0, which is not the case in my example.

Here are the results acquired using matlab (the first number is the amount of iterations):

```
>> bisection(-5,0)
    22


ans =

   -2.3037
```

```
>> bisection(0,10)
    23


ans =

    0.8975
```

```
>> newton(-2)
     5

   -2.3037
```

```
>> newton(5)
     4

    0.8975
```

As can be seen, the Newton's method is much faster in this case.

Task 2

In this task We are supposed to calculate all roots (real and imaginary) of a given function using MM1 and MM2 versions of Muller's method. The idea behind Muller's method is similar to secant's method, but instead of creating a line through 2 points, we create a parabola through 3 points. We implemented two versions of this method, denoted MM1 and MM2.

In version MM1 we create a parabola through points x_0, x_1, x_2, create a quadratic function and select one of the roots of the parabola as the next approximation of the solution. We describe the parabola using the formula $y(z) = a*z^2 + b*z + c$ and we can inquire:

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

Therefore, the following system of equations can be deduced:

$$\begin{aligned} az_0^2 + bz_0 = f(x_0) - f(x_2), \\ az_1^2 + bz_1 = f(x_1) - f(x_2). \end{aligned}$$

Then, roots of the 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 the smaller absolute value is chosen for the next iteration (denoted as x_3). Along with it, we chose two points from x_0, x_1, x_2, such that they are as close to x_3 as possible. Worth noting is the fact that this method works for both real and complex roots, so there is no need for an assumption that $b^2 - 4*a*c > 0$.

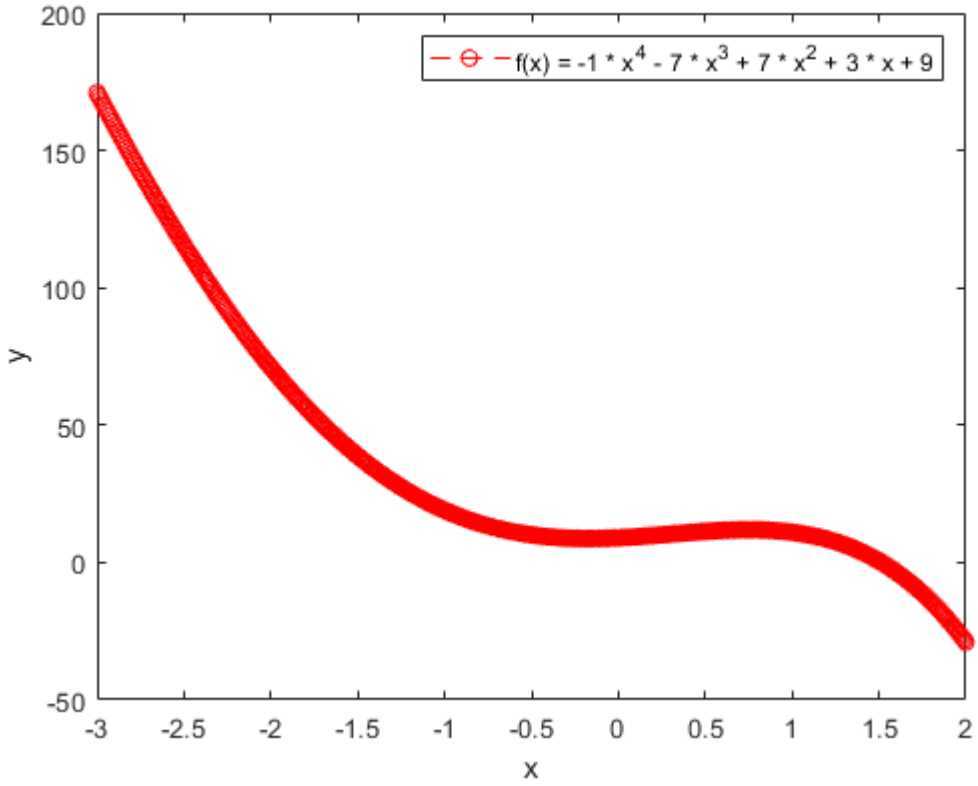
In method MM2, instead of using 3 values of the polynomial, we use only one and values of its first and second derivative. The formula for calculating the roots is:

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

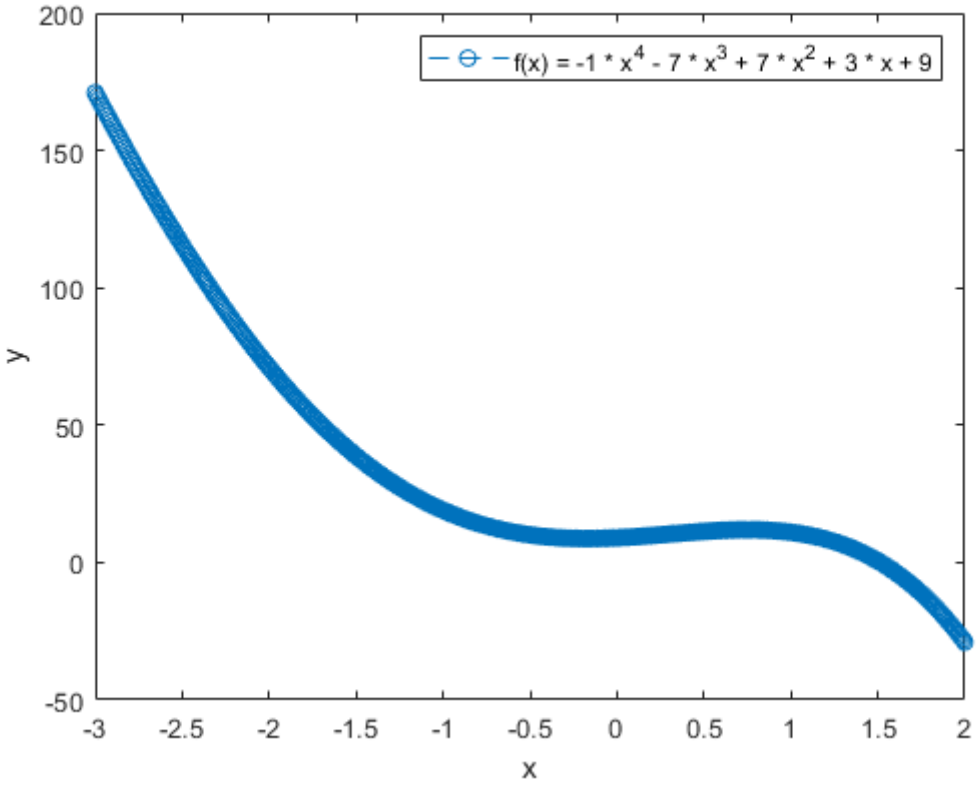
Then, we select the root with smaller absolute value as our approximation.

The MM2 version of Muller's method should also be applied in complex arithmetic, same as the MM1 version. It is locally convergent and locally more effective than the secant method. Additionally, it is almost as fast as the newton's method and able to find complex roots of a function.

```
>> Muller(1,1)
Iterations MM1: 9, Result MM1: -0.327282-0.805359j
Iterations MM1: 7, Result MM1: -0.327282+0.805359j
Iterations MM1: 4, Result MM1: 1.515070
Iterations MM1: 4, Result MM1: -7.860505
```




```
>> Muller(1,2)
Iterations MM2: 4, Result MM2: -0.327282+0.805360j
Iterations MM2: 4, Result MM2: -0.327282-0.805359j
Iterations MM2: 3, Result MM2: 1.515070
Iterations MM2: 3, Result MM2: -7.860505
```



As can be seen, both version are almost identical, except MM2 is slightly faster in our case. As for comparison to Newton's method:

```
>> newton_task2(0)
    16

    1.5151
```

```
>> newton_task2(-6)
    11

   -7.8605
```

Using the same initial values (0 and -6) the Muller's method yields the following results:

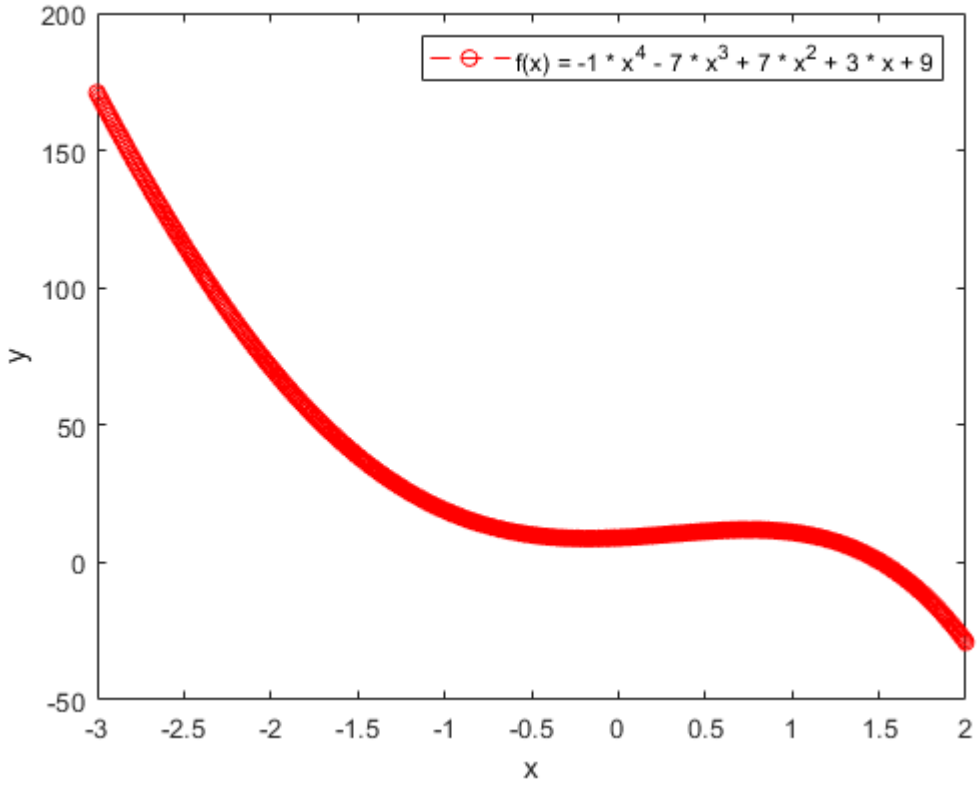
```
>> Muller(1,2)
Iterations MM2: 4, Result MM2: -0.327282+0.805360j
Iterations MM2: 4, Result MM2: -0.327282-0.805359j
Iterations MM2: 3, Result MM2: 1.515070
Iterations MM2: 3, Result MM2: -7.860505

>> Muller(1,2)
Iterations MM2: 4, Result MM2: -7.860505
Iterations MM2: 5, Result MM2: -0.327282+0.805359j
Iterations MM2: 2, Result MM2: -0.327282-0.805359j
Iterations MM2: 2, Result MM2: 1.515070
>>
```

As can be seen, the MM2 method in this case is much faster and on top of that it shows the complex roots.

Task 3

In this task we were asked to find complex and real roots using the Laguerre's method and compare the results to the MM2 version of Muller's method.
Laguerre's method is almost identical to MM2, the only differences being the fact that the first takes into consideration the order of the polynomial and the fact that it's globally convergent. Using initial point 0 Laguerre's method gives the following result:

```
>> task3(1)
Iterations Laguerres: 6, Result Laguerres: -0.327282+0.805359j
Iterations Laguerres: 5, Result Laguerres: -0.327282-0.805359j
Iterations Laguerres: 4, Result Laguerres: 1.515070
Iterations Laguerres: 3, Result Laguerres: -7.860505
```



As can be seen, the MM2 method again wins the race for the fastest method. Again, worth noting is the fact that the MM2 method is less general due to the fact that it's locally convergent as opposed to global convergence of the Laguerre's method.