ANALYSIS MIDTERM PROJECT 1

1. **Implementing a MATLAB root finder**

For the function findroot, we will have five inputs with two of them are optional.

In order to achieve optional inputs, we use an if statement and exist function to test whether the optional inputs were assigned values. If they were not preassigned values, then we will assign default values to them.

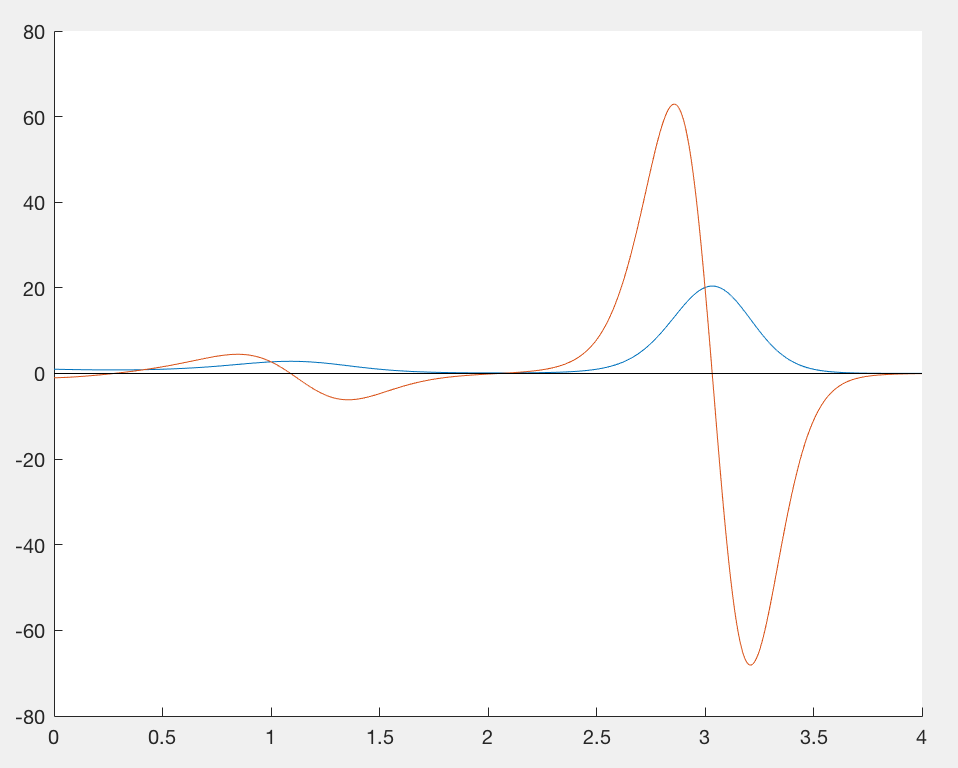
The core structure of the function is from class MATLAB notes. oneRoot will displace the approximate root and if there is not a root, oneRoot will return nan.

All the iterates were recorded into a vector V. We will ask whether people want to display V by asking for an input (Y/N).

The code of asking for displaying of V and plotting graph are hided for the following questions for the sake of continence. For problem 1, please enable these codes.

1. **Adventures in root finding**

(a)



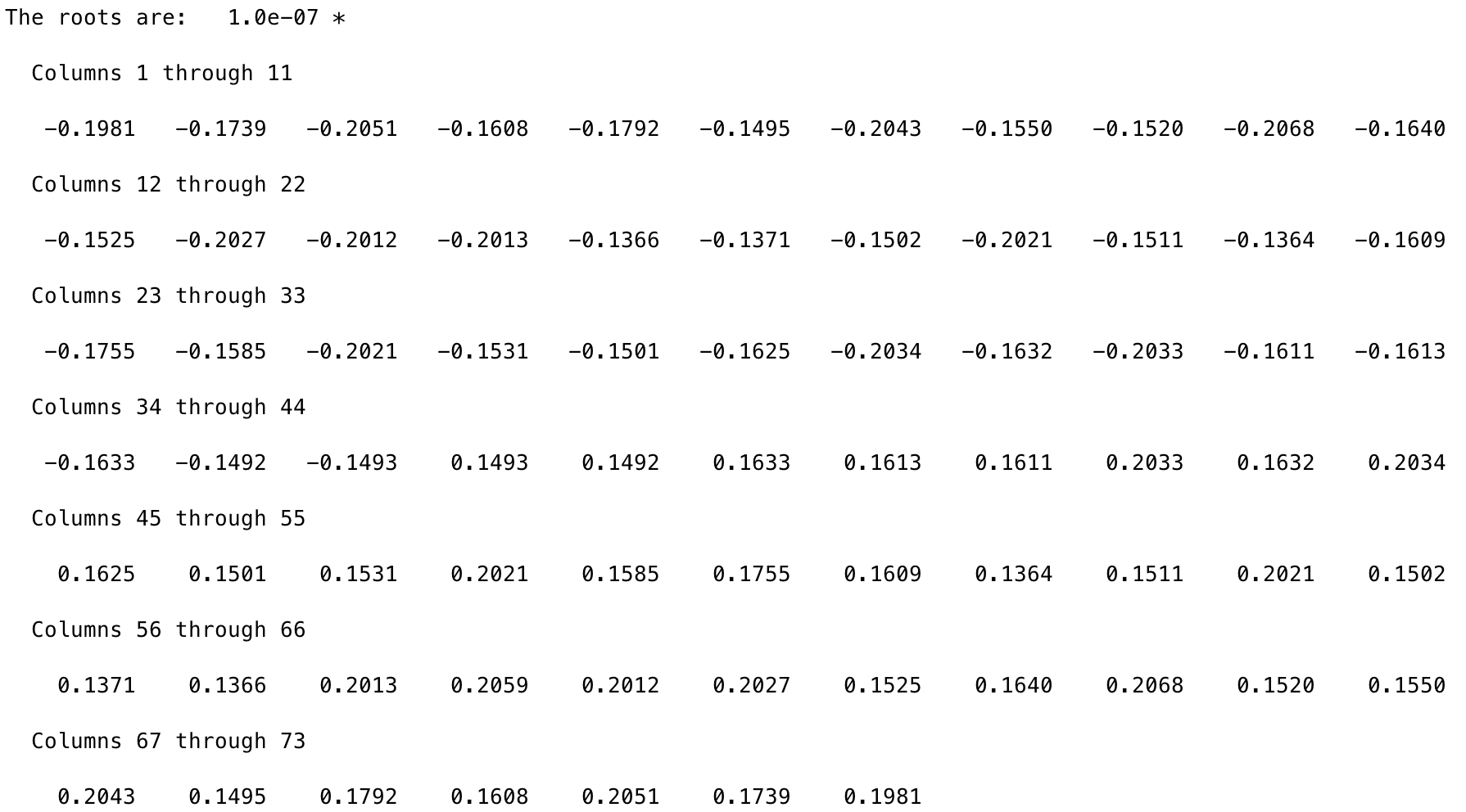
From the graph above, we can see that there are four roots with [0,4]. Therefore, after using L to store all the approximate roots of df=0, we divide the roots into four different ranges, since these roots are all converging to the real roots.

Therefore, we get the local extrema are 0.2739, 1.0904, 2.0491, and 3.0333.

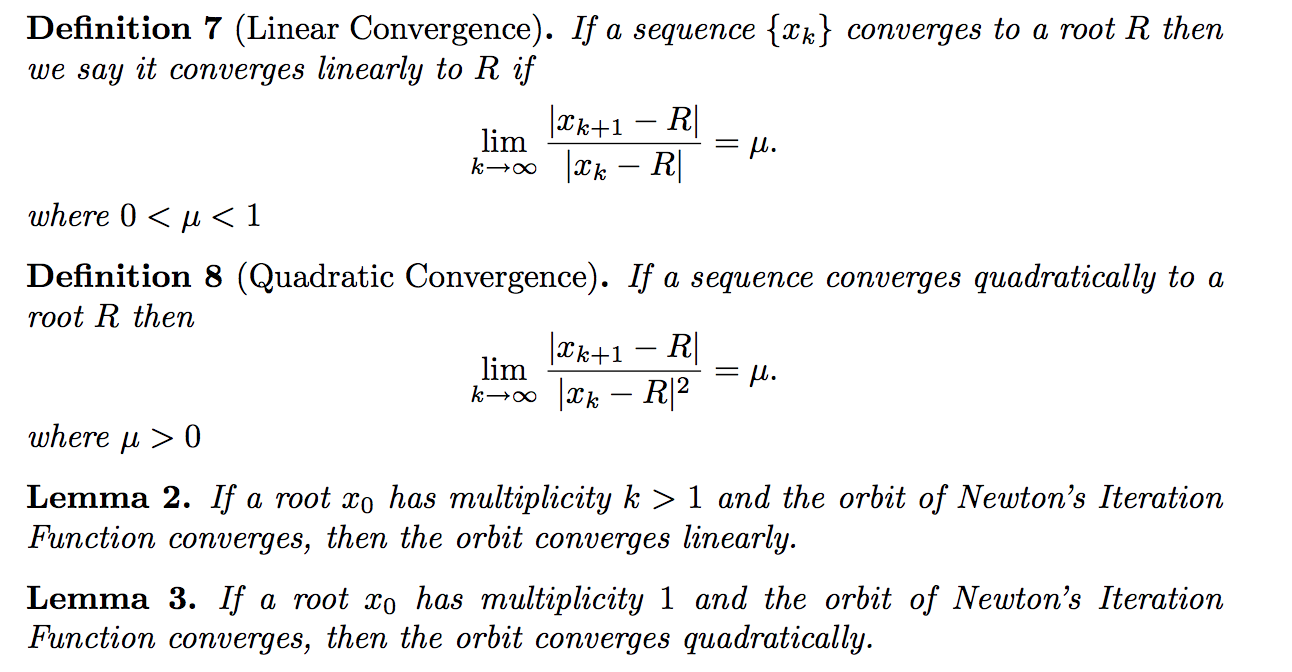
4.025 are the roots converging to x axis but there is not a root.

(b)

We use for i = -0.5:0.01:0.5 to simulate the varying initial points. And we use L to store the roots. Since from the current guesses, we can see the guesses are decreasing to zero digit by digit slowly instead of having a lot of jumps. Therefore, we are assuming that the Newton iteration appear to be converging, and in the order of linear.



In order to illustrate linear convergence, we also try to obtain this:



From the codes, R=0 represents the real root x=0. S will store the ratio, mu.

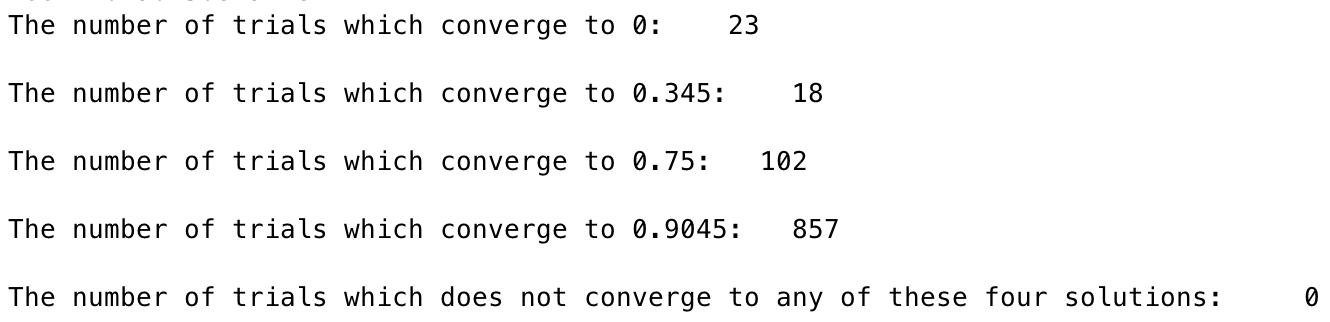
After the calculation, the average mu is about 1.0161. Slightly larger than zero, but could still be considered as a not very rigorous linear convergence.

(c)

To get initial data randomly generated in [0.8, 1], we use initialdata = 0.8 + (1-0.8).\*rand(1).

Put g(x)=4x-4x2 into g(g(x))=x, we get f(t) = -64\*x^4 +128\*x^3 - 80\*x^2 +15\*x.

Then we do for loop for 1000 times, and record all the approximate roots into vector L. Since we knows that there are four true roots, and all the approximate roots are converging to these four true roots, we set four interval bins to categorize these 1000 roots, and count them.

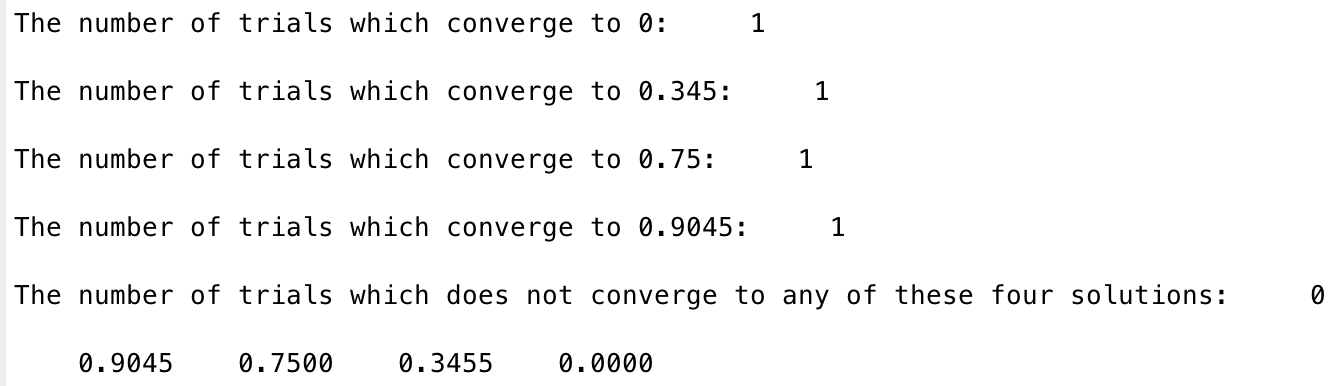


(d)

All roots are not found equally often, and they did not even find all once. For problem (c), n=100 won’t be able find all four roots once. Since our initial guess is in the range of [0.8,1], therefore, roots inside this range are more likely to be find.

By using root deleting, we use a for loop to run the Newton iterates, then if the solution is not nan, delete this root. By doing so, we don’t need to worry about repeated roots.

If a function has four roots, then for a for loop running 10 times, there could be six times of Newton iterates generates no roots. In this case, we set that if there are three time of nan roots generated by Newton iterates, we stop the for loop.

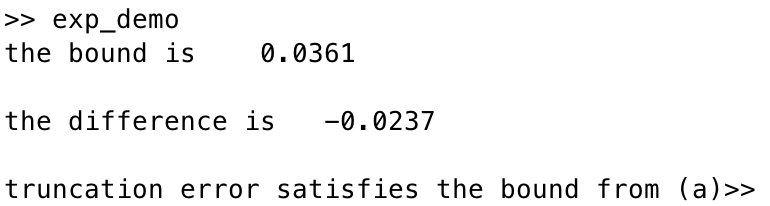


1. **Truncation error analysis**

(a)

(b)

By calculating A = 1 + (1/2) + 0.5\*(1/2)^2 and B=exp(1/2), we have A equals to the approximate value of exp(1/2) and B equals the builtin function exp(1/2). And by comparing the difference of A and B and the truncation error with x=1/2, we could get that the absolute difference is 0.0237, which is smaller than the bound, 0.0361.

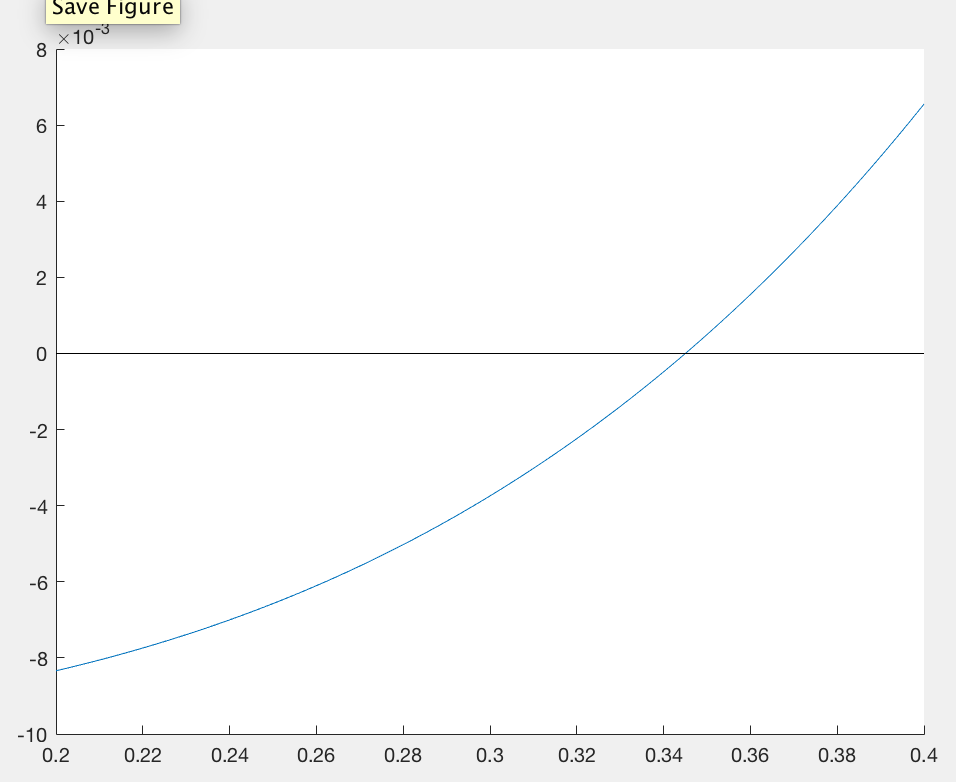


(c)

To turn this question into root finding problem, we build function handle f, which equals to the truncation error minus the bound.

f = @(x)((3.^x).\*(x.^3))./6 - (10.^(-2));

df = @(x)((3.^(x)).\*(x.^2))/2+(log(3).\*(x.^3).\*(3.^(x-1)))/2;

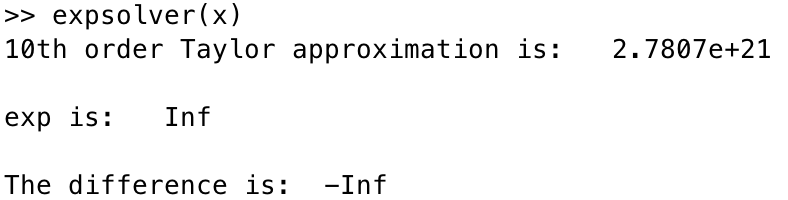
Then to approximate initial value x0 to put into findroot function, we plot f and obtain the interval of the root in [0,2,0.4].

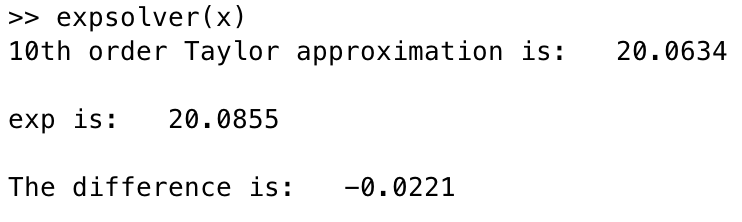
Use x0=0.2 for the findroot function and we get the result that Root: 0.345. Since from the graph we could see that (3.^x).\*(x.^3)-1/(60.^2)<=0 if x<=0.345, therefore, the largest value b =0.345

(d)

The function only computes the 10th order Taylor approximation and exp(x) with the input x.

When x=3, the difference is comparable. When x=1000, the difference is incomparable, since when x=1000, x^3 becomes very big to compute which leads to the infinite difference.

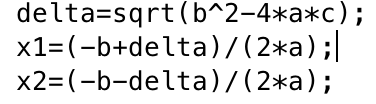




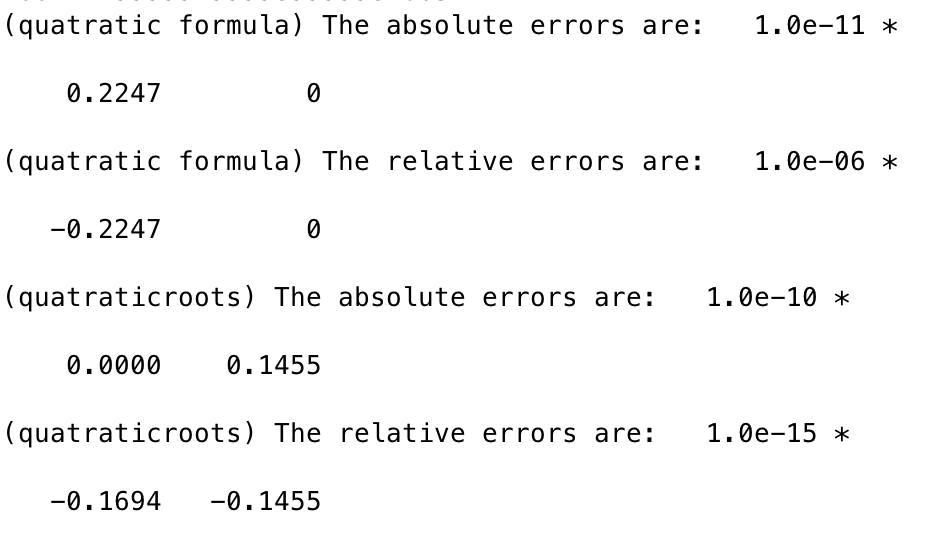
1. **Newton vs quadratic formula**

(a)

To call findroot for r1 and r2, we would have to call findroot twice with two initial data. To get these two initial data, we decided to use quadratic formula to use coefficients a, b, c to calculate x1 and x2 to implement r1 and r2.



(b)



To calculate the absolute error and relative error, the idea was written in the comment of the codes. We can see from the results that the error by using quatraticroots are smaller than using quadratic formula. (0.1455<0.2247)

(c)

Since sqrt(b^2-4ac)=999.9999…in this question, and b=1000. And –b+ sqrt(b^2-4ac) could also be written in a form of sqrt(b^2-4ac)-b. Since if there are two very closed value minus each other, there would exist catastrophic cancellation leading to this large error.