Damian Franco

CS 375 – 001

Homework #5:

1. For number 1, we oversaw finding the three roots for the function below (f(*x*)) using Newton’s method. The *newton.m* MATLAB function that Professor Lau has supplied for us helped with finding the roots through MATLAB. The main formula to approximate a root using Newton’s method is with **r = *x0* – f(*x0*)/df(*x0*)** where f is the function, df is the first derivative of f, *x0* is the initial guess value and *r* is the root approximation. This function will be performed recursively until it reaches convergence. The MATLAB function will use a while loop to perform this. I did some small updates to the *newton.m* MATLAB and had it return a list of the iters and the root approximation. The term *S* is also an approximation of the root at the function and that was given to us in the question PDF for the homework, which is the same as the bolded formula above. In the output screenshots below, I show the root approximation from *newton.m*, and the the *S* approximations. Next, I printed out the error approximation of the Newton method. Then the relevant error ratio is calculated with ***ei + 1 / ei2*** or ***ei + 1*** / ***ei***. Lastly, I print out the *M* approximation of the function using the equation ***M =* |*f’’(r)/(2f’(r))*|**. When comparing the *M* and relevant error ratio approximation, I noticed the for the first root and last root they are going toward the same numbers and look very similar with decimal differences. With the middle root, that where it gets crazy because the relevant error ratio is massive. When comparing it to the *M* approximation at that root, there is a huge difference. The relevant error ratio seems to be increasing to a similar number like *M*, but my MATLAB code only allowed a total of 29 approximations and if I did more, it would break, unfortunately. Overall, this question has taught me that the Newton method is very trustworthy when implemented in MATLAB. The comparisons, screenshots and graph of the function are shown below.

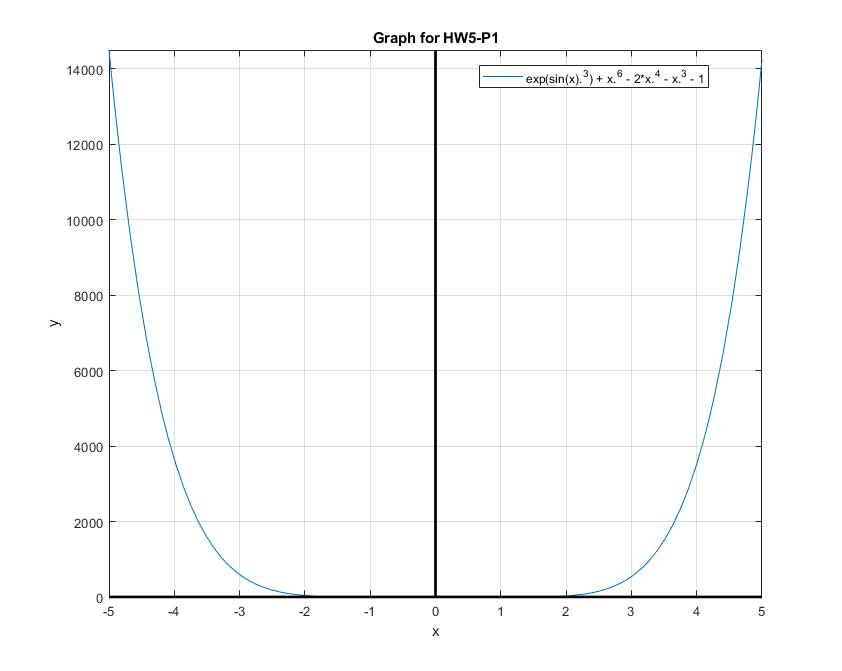
*f(x)* = exp(sin^3(x)) + x^6 - 2\*x^4 - x^3 – 1

*f’(x)* = 6\*x^5 - 8\*x^3 - 3\*x^2 + 3\*exp(sin^3(x))\*sin^2(x)\*cos(x)

*f’’(x­)* = 30\*x^4 - 24\*x^2 - 6\*x + 3 \* ...

(3\*exp(sin^3(x))\*sin^4(x)\*cos^2(x) + ...

exp(sin^3(x)\*(sin(2\*x)\*cos(x) – sin^3(x))))



**MATLAB Graph:**

*HW5-P1-Graph.jpg*

**MATLAB Source code:**

*newton.m*

1 function [x iters] = newton(f,df,x0,tol,kmax);

2 % function x = newton(f,df,x0,tol,kmax);

3 % Given a differentiable function f with df = df/dx, routine,

4 % when convergent returns an approximate root x obtained via

5 % Newton-Raphson iteration. Other inputs are an error tolerance

6 % tol (max over error between successive iterations and abs(f)),

7 % the max number kmax allowed iterations, and initial iteration

8 % x0. tol=1e-8 and kmax = 1e5 are defaults, if left unspecified.

9 switch nargin

10 case 3,

11 tol = 1e-8;

12 kmax = 1e5;

13 case 4,

14 kmax = 1e5;

15 case 5,

16 % Fall through.

17 otherwise,

18 error('newton called with incorrect number of arguments')

19 end

20

21 iters = zeros(0, 15);

22 x = x0; err=100; k = 0; % x=x0 here allows return for large tol.

23

24 while err >= tol

25 iters = [iters x];

26 %iters = x;

27 y = f(x0);

28 x = x0 - y/df(x0);

29 err = max(abs(y),abs(x-x0));

30 x0 = x;

31 k = k+1;

32 if k>=kmax

33 disp(['No convergence after ' num2str(kmax) ' iterations.'])

34 disp(['Error at this stage is ' num2str(err)])

35 return

36 end

37 end

38

39 if isfinite(x)

40 disp(['Converged in ' num2str(k) 'iterations with tol' num2str(tol)])

41 else

42 disp(['No convergence after' num2str(k)'iterations. Iterations' ...

43 ' overflowed to infinity.']);

44 end

*HW5\_1.m*

1 %

2 % function: HW5\_1

3 % This function will set up and approximate the root

4 % for the function given to us in problem 1 using

5 % Newtons law and will graph the function. It also will

6 % find the error based off the last iteration.

7 %

8 % function set up

9 f = @(x) exp(sin(x).^3) + x.^6 - 2\*x.^4 - x.^3 - 1;

10 % first derivative

11 fp = @(x) 6\*x.^5 - 8\*x.^3 - 3\*x.^2 + 3\*exp(sin(x).^3)\*sin(x).^2\*cos(x);

12 % second derivative

13 ffp = @(x) 30\*x.^4 - 24\*x.^2 - 6\*x + 3 \* ...

14 (3\*exp(sin(x).^3)\*sin(x).^4\*cos(x).^2 + ...

15 exp(sin(x).^3\*(sin(2\*x)\*cos(x) - sin(x).^3)));

16 % Plot the equation

17 fplot(f)

18 line([0,0], ylim, 'Color', 'k', 'LineWidth', 2);% Draw line for Y axis.

19 line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2);% Draw line for X axis.

20 title('Graph for HW5-P1')

21 xlabel('x')

22 ylabel('y')

23 legend('exp(sin(x).^3) + x.^6 - 2\*x.^4 - x.^3 - 1')

24 grid on

25 % Uncomment the root you would like to approximate

26 % root 1

27 x0 = -2; %9 iters

28 % root 2

29 % x0 = 0.5; %31 iters

30 % root 3

31 % x0 = 2; %7 iters

32 % Find roots

33 [x iters] = newton(f, fp, x0);

34 it = 9; % change it to the # of iterations

35 r = iters(it);

36 % Find S in case of linear convergence

37 S = r - f(r)./fp(r);

38 % Find M

39 M = abs(0.5\*ffp(r)./fp(r));

40 % Find quadratic errors

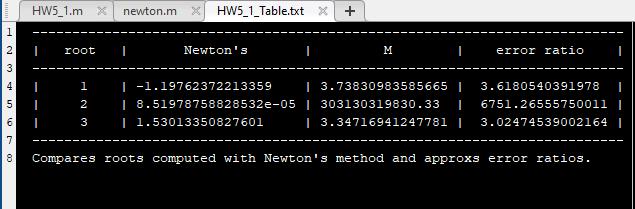
41 errs = abs(iters-r);

42 % Estimates the relevant error ratio

43 % errs(2:end)./(errs(1:end-1).^2)

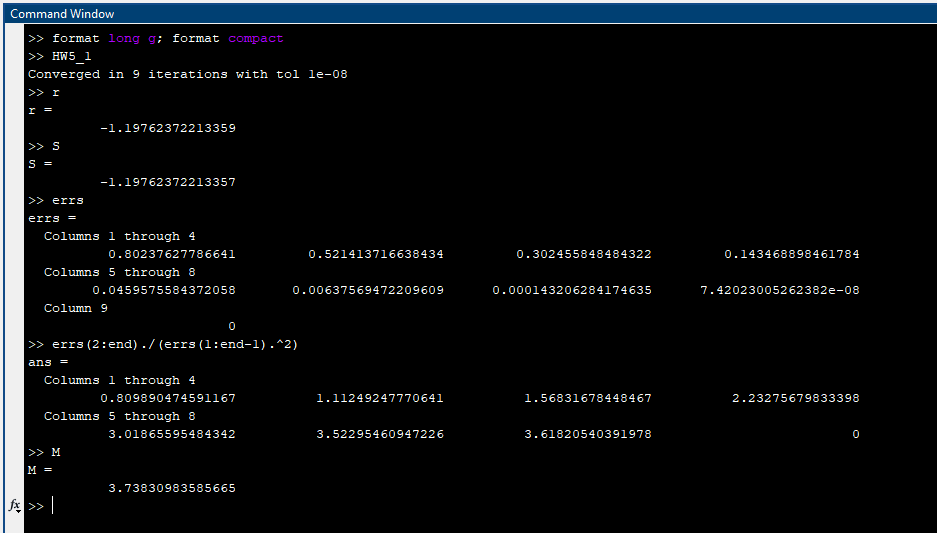
**Table of findings:**

*HW5\_1\_Table.txt*

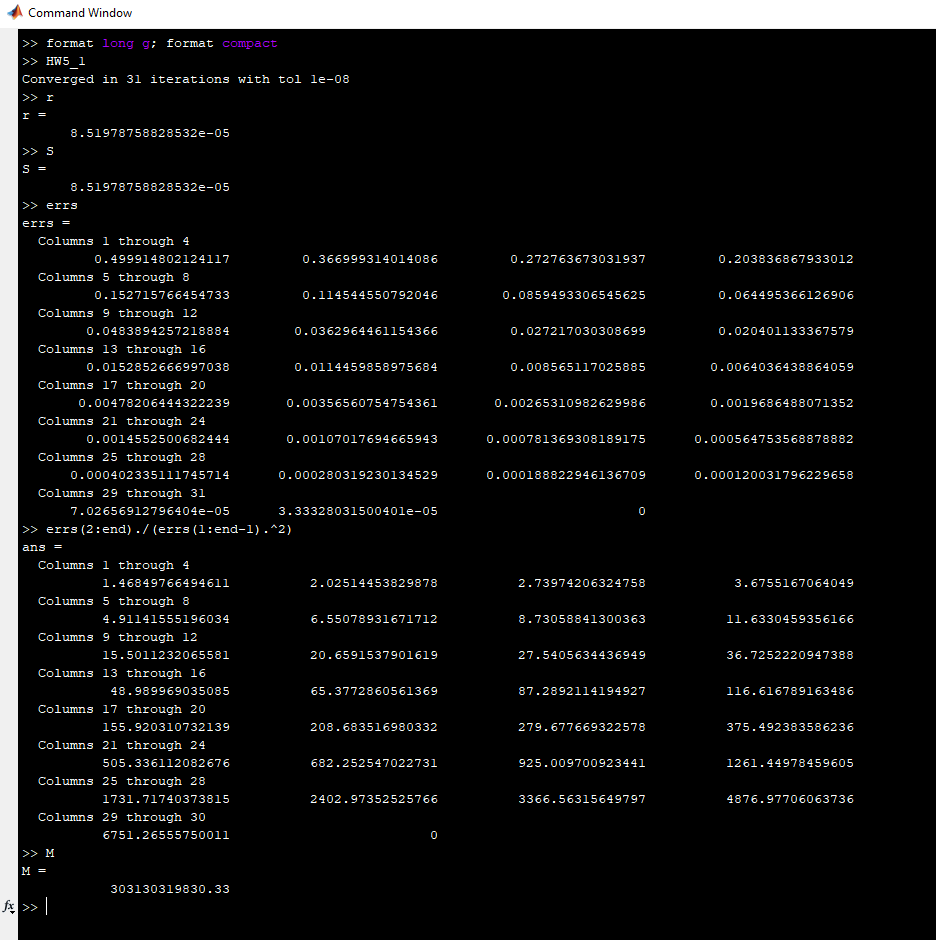


**MATLAB Console Outputs:**

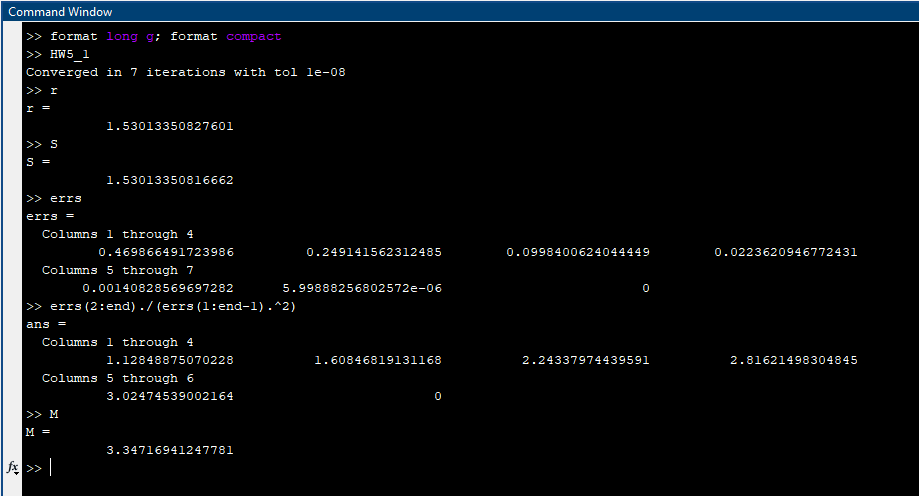
*First root:*



*Middle root:*



*Last root:*



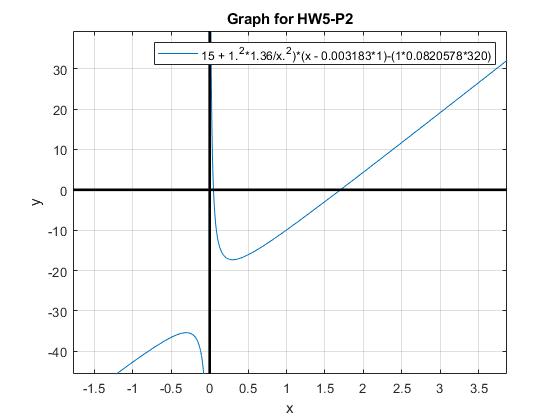
1. We are give an equation for the ideal gas law here which is (***P + n^2\*a/V^2)(V – nb) = nRT*** where *P* is pressure (in atm), *V* is volume (in L), *T* is temperature (in K), *n* is the amount of gas (in mol), and *R* = 0.0820578. In this case we are give that *n* = 1 mol of O, *T* = 320 K, *P* = 15 atm, *a* = 1.36 L 2atm/mol2 and *b* = 0.003183 L/mol. That will now give the equation with everything plugged (***15 + 1^2\*1.36/V^2)(V – 0.003183\*1) = 1\*0.0820578\*320*.** We must now solve for *V*, so that is where my MATLAB function below comes in. It will approximate the root at the initial guess *x0* at 1. Then after that the newton method is applied with Professors *newton.m* MATLAB function. I found that when graphed, there are two roots to this equation but for some reason I cannot approximate to the root near 1.5. I tried multiple ways and it just did not work, but it did work for the root near 0 and what I found is that this equation had a solution for *V* that is approximately equal to ***0.0498097324299803.*** This looks correct from the graph, and because of the high number of iteration, I excluded printing out the errors or the function but here is what I found. The errors bottomed out around ***0.0071609344999409***, which seems to be correct and the relevant error ratio topped out at ***14.4157682116974*** which is not far off from the *M* value that is approximated, ***18.6054565915662***. Once again, not quite there but it is decently close to each other. Overall, I found that this function was a very interesting one seeing that there are two roots and only converging to one and never converging to the other.

*f(x)* = (15 + 1^2\*1.36/x^2) \* (x – 0.003183\*1) – (1\*0.0820578\*320)

*f’(x)* = 16.39(-x+0.006366) / x^3

*f’’(x­)* = 16.39(-2x+0.019098) / x^4

**MATLAB Graph:**

*HW5-P2-Graph.jpg*

**MATLAB Source code:**

*HW5\_2.m*

1 %

2 % function: HW5\_2

3 % This function will set up and approximate the root

4 % for the function given to us in problem 2 using

5 % Newtons law and also graphs the function. It will

6 % find the error based off of the last iteration.

7 %

8 % function set up

9 f = @(x) (15 + 1.^2\*1.36/x.^2)\*(x - 0.003183\*1)-(1\*0.0820578\*320);

10 % first derivative

11 fp = @(x) 16.39\*(-x+0.006366)/x.^3;

12 % second derivative

13 ffp = @(x) 16.39\*(-2\*x+0.019098)/x.^4;

14 % Plot the equation

15 fplot(f)

16 line([0,0], ylim, 'Color', 'k', 'LineWidth', 2);% Draw line for Y axis.

17 line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2);% Draw line for X axis.

18 title('Graph for HW5-P2')

19 xlabel('x')

20 ylabel('y')

21 legend('15 + 1.^2\*1.36/x.^2)\*(x - 0.003183\*1)-(1\*0.0820578\*320)')

22 grid on

23 % Uncomment the root you would like to approximate

24 % root 1

25 x0 = 1.5; %258 iters

26 % Find roots

27 [x iters] = newton(f, fp, x0);

28 it = 257; % change it to the # of iterations

29 r = iters(it);

30 % Find S in case of linear convergence

31 S = r - f(r)./fp(r);

32 % Find M

33 M = abs(0.5\*ffp(r)./fp(r));

34 % Find quadratic errors

35 errs = abs(iters-r);

36 % Estimates the relevant error ratio

37 % errs(2:end)./(errs(1:end-1).^2)

**MATLAB Console Outputs:**

*First root (does not print iters and errs for the high number of iterations):*

