

Damian Franco

CS 375 – 001

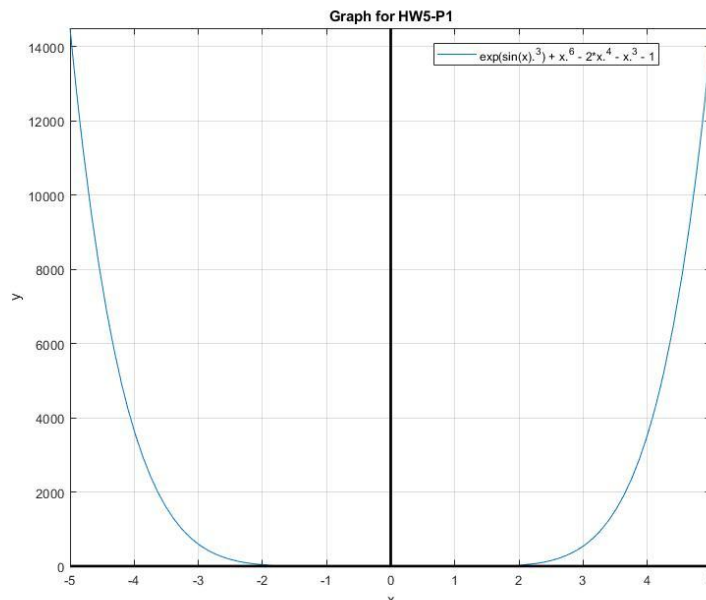
Homework #5:

- 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 $r = x_0 - f(x_0)/df(x_0)$ where f is the function, df is the first derivative of f , x_0 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 e_{i+1}/e_i^2 or e_{i+1}/e_i . 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.

$$\begin{aligned}f(x) &= \exp(\sin^3(x)) + x^6 - 2x^4 - x^3 - 1 \\f'(x) &= 6x^5 - 8x^3 - 3x^2 + 3\exp(\sin^3(x))\sin^2(x)\cos(x) \\f''(x) &= 30x^4 - 24x^2 - 6x + 3 * \dots \\&\quad (3\exp(\sin^3(x))\sin^4(x)\cos^2(x) + \dots \\&\quad \exp(\sin^3(x))(\sin(2x)\cos(x) - \sin^3(x)))\end{aligned}$$

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

HW5_1.m

newton.m

HW5_1_Table.txt

+

1

2

| root | Newton's | M | error ratio |

3

4

| 1 | -1.19762372213359 | 3.73830983585665 | 3.6180540391978 |

5

| 2 | 8.51978758828532e-05 | 303130319830.33 | 6751.26555750011 |

6

| 3 | 1.53013350827601 | 3.34716941247781 | 3.02474539002164 |

7

8

Compares roots computed with Newton's method and approxs error ratios.

MATLAB Console Outputs:

First root:

```
Command Window
>> format long g; format compact
>> HW5_1
Converged in 9 iterations with tol 1e-08
>> r
r =
    -1.19762372213359
>> S
S =
    -1.19762372213357
>> errs
errs =
Columns 1 through 4
    0.80237627786641    0.521413716638434    0.302455848484322    0.143468898461784
Columns 5 through 8
    0.0459575584372058    0.00637569472209609    0.000143206284174635    7.42023005262382e-08
Column 9
         0
>> errs(2:end)./(errs(1:end-1).^2)
ans =
Columns 1 through 4
    0.809890474591167    1.11249247770641    1.56831678448467    2.23275679833398
Columns 5 through 8
    3.01865595484342    3.52295460947226    3.61820540391978    0
>> M
M =
    3.73830983585665
fx >> |
```

Middle root:

Command Window

```
>> format long g; format compact
>> HW5_1
Converged in 31 iterations with tol 1e-08
>> r
r =
    8.51978758828532e-05
>> S
S =
    8.51978758828532e-05
>> errs
errs =
Columns 1 through 4
    0.499914802124117    0.366999314014086    0.272763673031937    0.203836867933012
Columns 5 through 8
    0.152715766454733    0.114544550792046    0.0859493306545625    0.064495366126906
Columns 9 through 12
    0.0483894257218884    0.0362964461154366    0.027217030308699    0.020401133367579
Columns 13 through 16
    0.0152852666997038    0.0114459858975684    0.008565117025885    0.0064036438864059
Columns 17 through 20
    0.00478206444322239    0.00356560754754361    0.00265310982629986    0.0019686488071352
Columns 21 through 24
    0.0014552500682444    0.00107017694665943    0.000781369308189175    0.000564753568878882
Columns 25 through 28
    0.000402335111745714    0.000280319230134529    0.000188822946136709    0.000120031796229658
Columns 29 through 31
    7.02656912796404e-05    3.33328031500401e-05    0
>> errs(2:end)./(errs(1:end-1).^2)
ans =
Columns 1 through 4
    1.46849766494611    2.02514453829878    2.73974206324758    3.6755167064049
Columns 5 through 8
    4.91141555196034    6.55078931671712    8.73058841300363    11.6330459356166
Columns 9 through 12
    15.5011232065581    20.6591537901619    27.5405634436949    36.7252220947388
Columns 13 through 16
    48.989969035085    65.3772860561369    87.2892114194927    116.616789163486
Columns 17 through 20
    155.920310732139    208.683516980332    279.677669322578    375.492383586236
Columns 21 through 24
    505.336112082676    682.252547022731    925.009700923441    1261.44978459605
Columns 25 through 28
    1731.71740373815    2402.97352525766    3366.56315649797    4876.97706063736
Columns 29 through 30
    6751.26555750011    0
>> M
M =
    303130319830.33
```

f5 >> |

Last root:

```
Command Window
>> format long g; format compact
>> HW5_1
Converged in 7 iterations with tol 1e-08
>> r
r =
    1.53013350827601

>> S
S =
    1.53013350816662

>> errs
errs =
Columns 1 through 4
    0.469866491723986    0.249141562312485    0.0998400624044449    0.0223620946772431
Columns 5 through 7
    0.00140828569697282    5.99888256802572e-06    0
>> errs(2:end)./(errs(1:end-1).^2)
ans =
Columns 1 through 4
    1.12848875070228    1.60846819131168    2.24337974439591    2.81621498304845
Columns 5 through 6
    3.02474539002164    0

>> M
M =
    3.34716941247781

fx >> |
```

- 2) We are given 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 given that $n = 1$ mol of O, $T = 320$ K, $P = 15$ atm, $a = 1.36$ L²atm/mol² and $b = 0.003183$ L/mol. That will now give the equation with everything plugged $(15 + 1^2 \cdot 1.36/V^2)(V - 0.003183 \cdot 1) = 1 \cdot 0.0820578 \cdot 320$. 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 x_0 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.

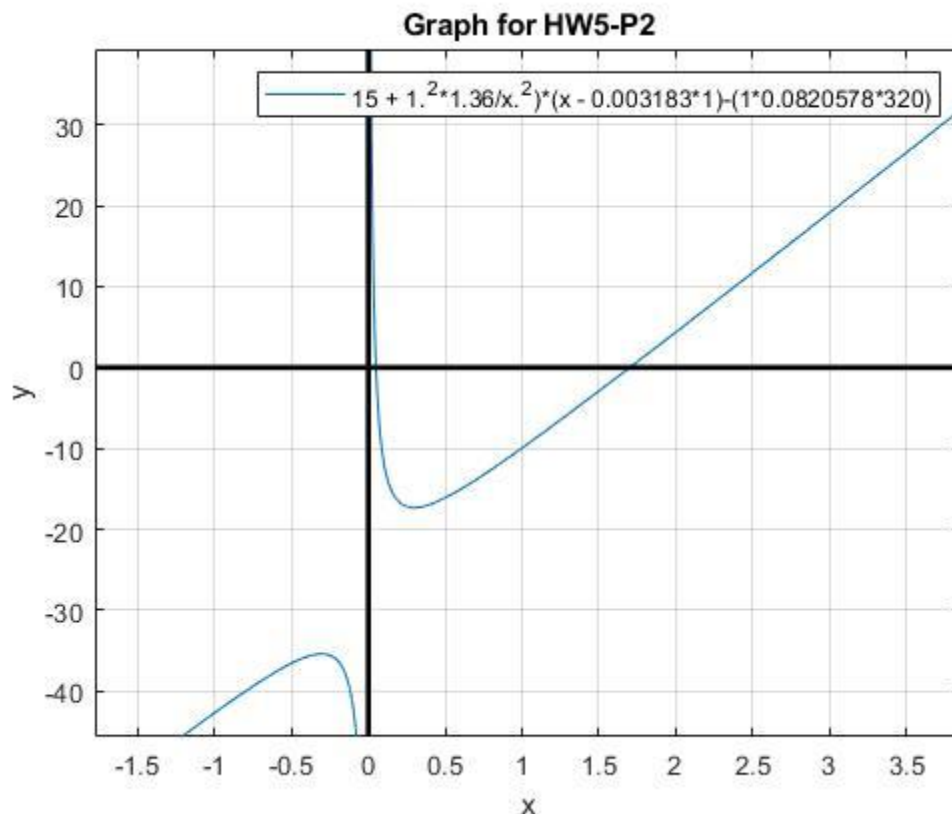
$$f(x) = (15 + 1^2 \cdot 1.36/x^2) \cdot (x - 0.003183 \cdot 1) - (1 \cdot 0.0820578 \cdot 320)$$

$$f'(x) = 16.39(-x + 0.006366) / x^3$$

$$f''(x) = 16.39(-2x + 0.019098) / x^4$$

MATLAB Graph:

HW5-P2-Graph.jpg



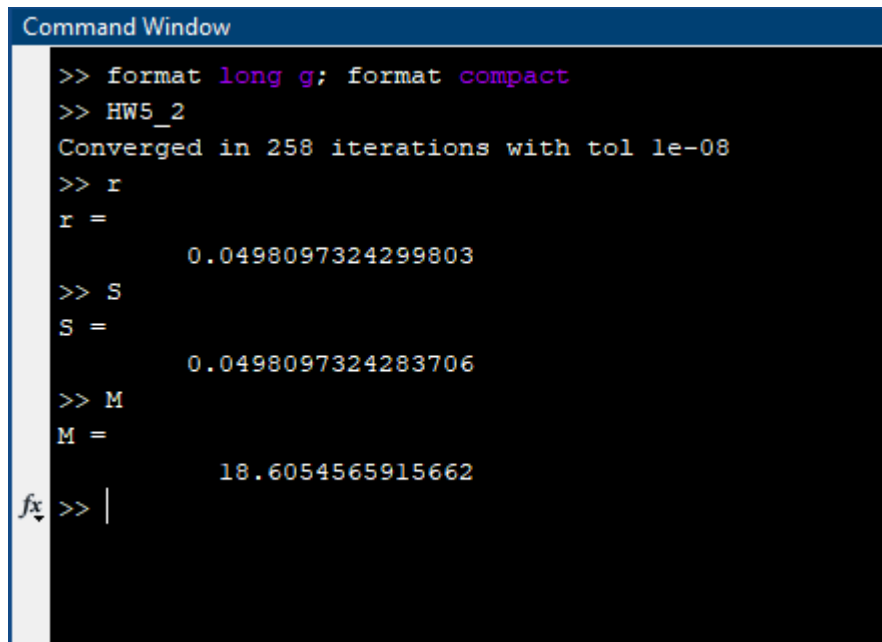
MATLAB Source code:

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):

A screenshot of the MATLAB Command Window. The window has a blue title bar that says "Command Window". The background is black with white text. The text shows the execution of a script named "HW5_2", which has converged in 258 iterations with a tolerance of 1e-08. The user then displays the values of variables r, S, and M. The value of r is 0.0498097324299803, S is 0.0498097324283706, and M is 18.6054565915662. The prompt is currently at the start of a new line.

```
>> format long g; format compact
>> HW5_2
Converged in 258 iterations with tol 1e-08
>> r
r =
    0.0498097324299803
>> S
S =
    0.0498097324283706
>> M
M =
    18.6054565915662
fx >> |
```