NAME: (Signature) [illegible] Amelia Rotondo - 887925113

Problem	Points
1	/25
2	/25
3	/25
Total	/75

INSTRUCTIONS:

1. Any form of collaboration on the exam is strictly forbidden. All work must be yours and yours alone. Include this sheet with your signature attesting that you did not receive any help on the exam.
2. You may use any of the built-in MATLAB functions, and the help files, as well as the course textbook and notes. Use of the internet is not allowed.
3. Clearly mark the number of the problem.
4. Show all details of your work. Include any code used in your solutions, as well as all figures and explanations.
5. Compile your exam into a single pdf file before submitting to Canvas.
6. The exam is due at 1:00 PM, Tuesday, Oct 3, 2023.

1. The point of this exercise is to illustrate an interesting phenomenon about eigenvalues known as "avoidance of crossing." Recall that if A is a symmetric matrix, the spectral theorem says that the eigenvalues of A are real numbers. Another result says that "generically" the eigenvalues are distinct. That is, with probability 1, if the entries are chosen at random, the eigenvalues will be distinct. To illustrate this, we will choose two matrices at random, A and B, and calculate the eigenvalues of

$$A + tB$$

The eigenvalues $\lambda_i(t)$ are functions of t. We will plot them as a function of t from $t = 0$ to $t = 1$.

Write a code to do the following:

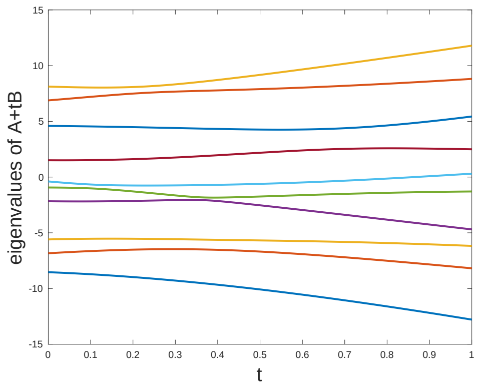
(a) Choose two different matrices, A and B, at random, and then make symmetric matrices, by

```
A = randn(10);
A = A + A';
```

and similarly for B.

(b) Make a vector t of values between 0 and 1. For each entry in t, calculate the eigenvalues of $A+tB$.

(c) Plot the eigenvalues as a function of t. You should get a figure that looks something like this:



Notice that sometimes the eigenvalue trajectories look they are going to cross, but veer away at the last moment. This is "avoidance of crossing."

2. The following are census figures for the population of the U.S. in millions in the years 1910, 1920, . . . , 2010:

1910	1920	1930	1940	1950	1960	1970	1980	1990	2000	2010
91.97	105.71	122.78	131.67	151.33	179.32	203.30	226.54	248.72	281,42	308.75

(a) Plot the data, marked with circles. On the same graph, plot the polynomial of degree 10, the shape-preserving pchip, and the spline interpolating the points.

(b) Plot the polynomial to the year 2020. Determine the year of "doomsday."

(c) Which of these interpolations would you trust? Which would you not trust, and why?

(d) Based on the data, estimate the population in 2005.

3. *Halley's method* is another iterative method for solving $f(x) = 0$. The Halley iteration formula for finding a simple zero is $x_{n+1} = g(x_n)$, where

$$g(x) = x - \frac{2f(x)f'(x)}{2[f'(x)]^2 - f(x)f''(x)}$$

   (a) Use Halley's method to find a solution of $2 - e^x = 0$.

   (b) Compare the performance of Halley's method with that of Newton's method by computing the iterates in both, starting from the same initial guess.

   (c) From the previous part, deduce the order of convergence of Halley's method.

   (d) Show that Halley's method for solving $a - e^x = 0$ is

$$x_{n+1} = x_n + 2\left(\frac{ae^{-x_n} - 1}{ae^{-x_n} + 1}\right)$$

   Comment on this as a practical means for computing $\log a$.

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

Note:

Code is broken up for readability, but each segment-per-number is built off of itself (i.e: the code for 1.b. relies on the code from 1.a. and so on for every letter in question 1).

1.

a.

```
% 1a)
% Two Different Random Matrices
A = rand(10);
B = rand(10);
% Make Them Symmetric
A = A + A';
B = B + B';
```

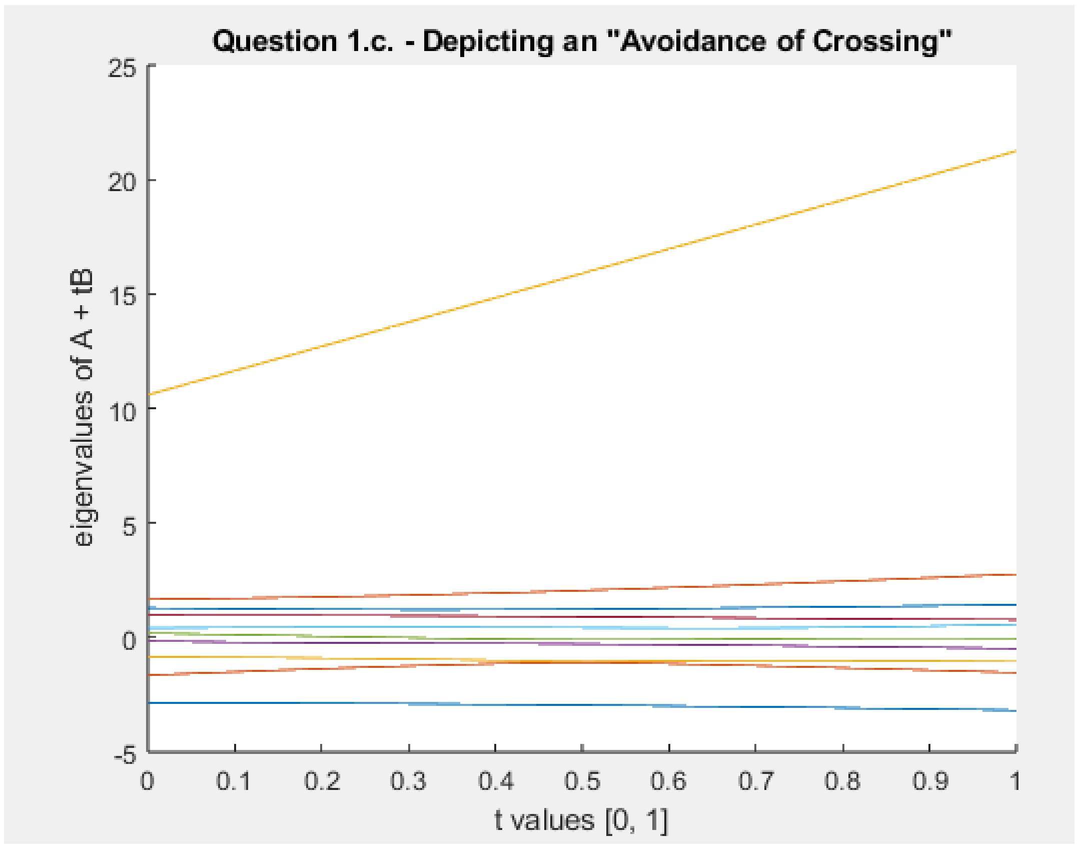
b.

```
% 1b)
% Vector t of 101 val.s between 0 and 1
t = 0:0.01:1;
% Array to hold Eigenvalues of A + tB
ab_eigs = zeros(10, size(t, 2));
% Iterate through every t-value:
for i = 1:length(t)
% our working value of t is t_val
t_val = t(i);
% C = A + tB for a specific t_val
C = A + t_val*B;
% Finds Eigenvalues of C (10 of em)
c_eig = eig(C);
% Stores a Specific c_eig in the Array of Eigenvalues
ab_eigs(:, i) = c_eig;
end
```

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

c.
```
% 1c)
% Plot your results (x_axis = t, y_axis = ab_eigs)
figure;
hold on;
% One Line for Each Eigenvalue
for i = 1:size(ab_eigs, 1)
plot(t, ab_eigs(i, :));
end
% Plot Formatting
hold off;
xlabel('t values [0, 1]');
ylabel('eigenvalues of A + tB');
title('Question 1.c. - Depicting an "Avoidance of Crossing"');
```

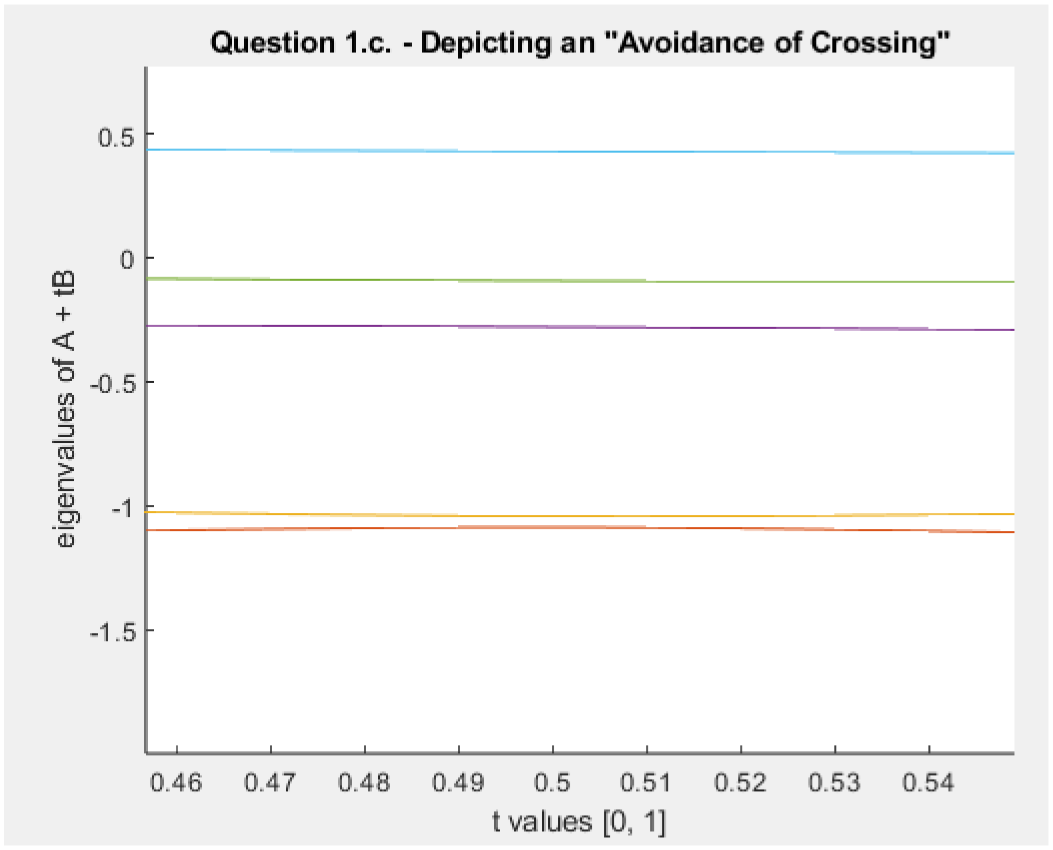


(Explanation of Graph Results is on the next page)

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

This plot clearly depicts an avoidance of crossing. For instance, at 0.5 it appears that the lines 2nd and 3rd from the bottom are going to intersect; However, if we zoom in at that point it becomes evident that those lines never actually intersect.



Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

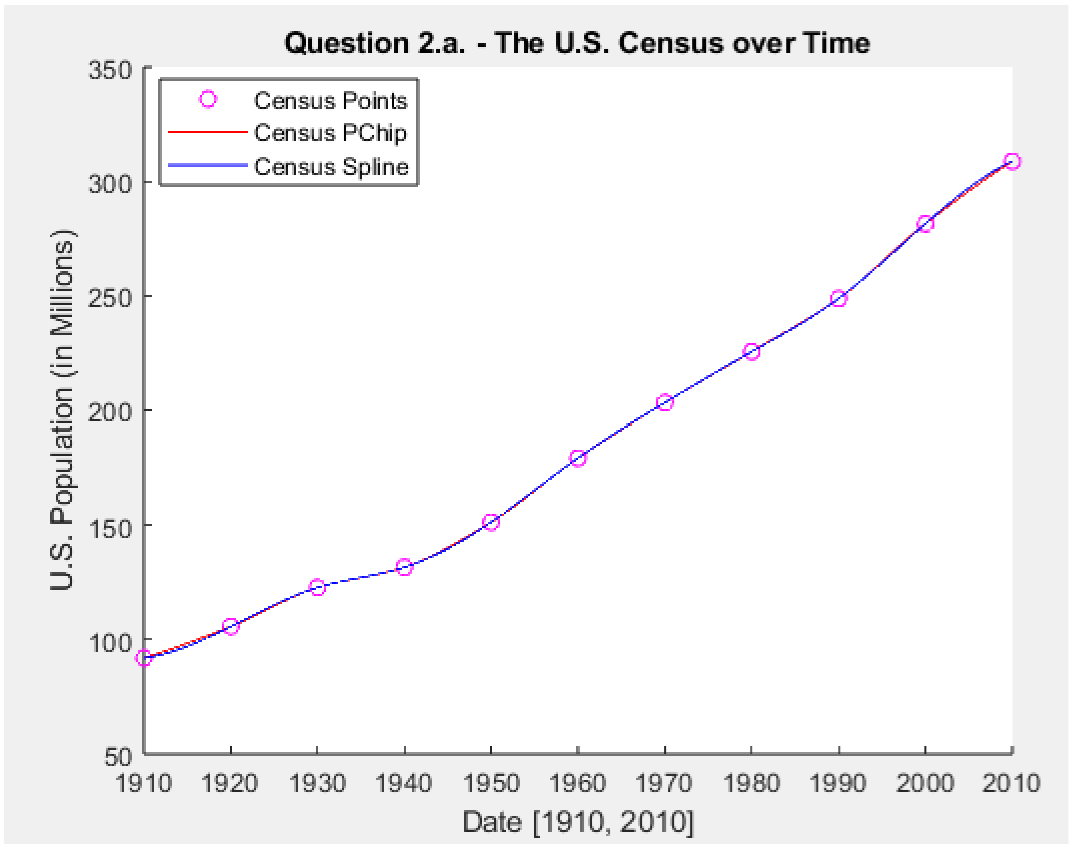
2. The following are census figures for the population of the U.S. in millions in the years 1910, 1920, . . . , 2010:
   a.

```
% 2a)
% "Populate" the Population Data (no pun intended)
Pop = zeros(2, 11);
% Establish the Census Information (given)
Census = [91.97, 105.71, 122.78, 131.67, 151.33, 179.32, 203.3,
225.54, 248.72, 281.42, 308.75];
% Row 1 is the Years, Row 2 is the Census Value
for i = 1:11
Pop(1, i) = 1900 + 10*i;
Pop(2, i) = Census(i);
end
% Establish x-values, y-values, and Domain for the plot
x = Pop(1, :);
y = Pop(2, :);
domain = min(x):0.01:max(x);
% Find the shape-preserving pchip of Pop
pop_pchip = pchip(x, y, domain);
% Find the spline of Pop
pop_spl = spline(x, y, domain);
% Set-up the Plot (census data marked in Magenta Circles)
figure;
hold on;
scatter(x, y, 'om', 'DisplayName', 'Census Points');
plot(domain, pop_pchip, '-r', 'DisplayName', 'Census PChip');
plot(domain, pop_spl, '-b', 'DisplayName', 'Census Spline');
hold off;
title('Question 2.a. - The U.S. Census over Time');
xlabel('Date [1910, 2010]');
ylabel('U.S. Population (in Millions)');
legend();
```

(Graph is on the next page)

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions



Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

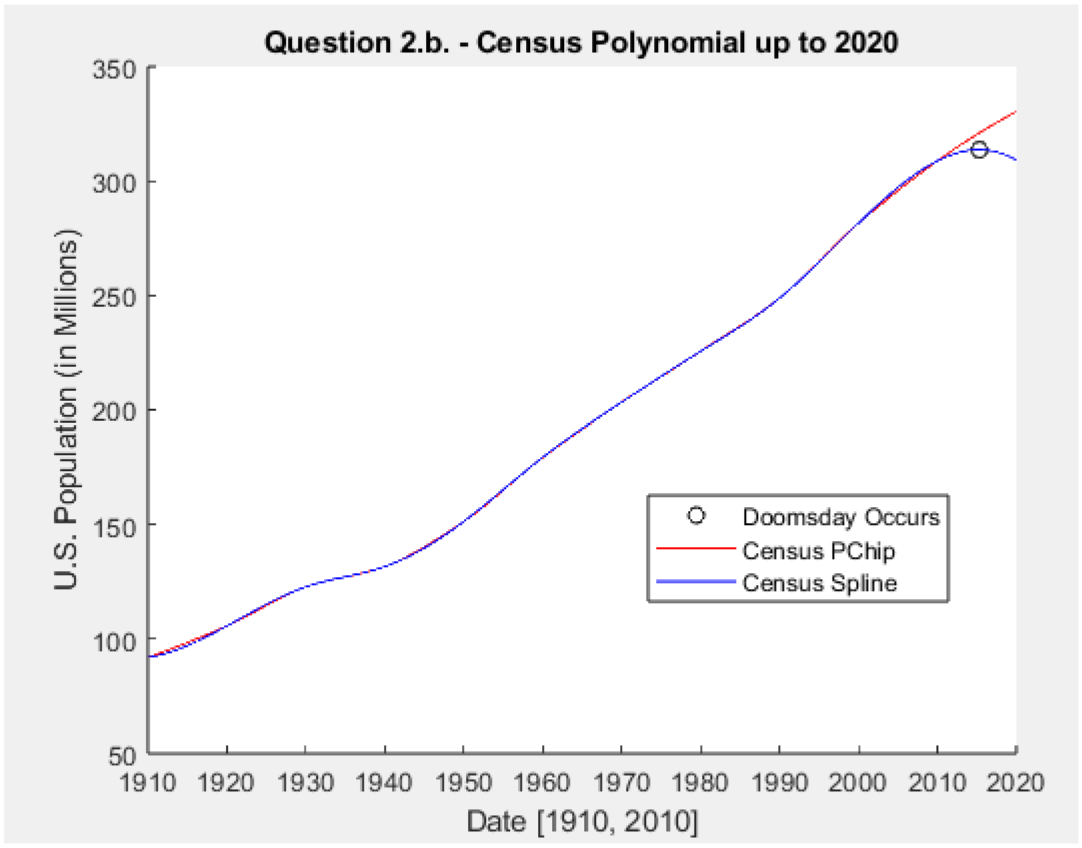
b. NOTE: I am unsure which polynomial to plot, so I am using both the pchip and spline from 2.a. I will show both of these in my code and graph. Also, note that in the graph the spline is the only line that starts decreasing between 2010 and 2020, so I'm using solely the spline to find the "doomsday" date.

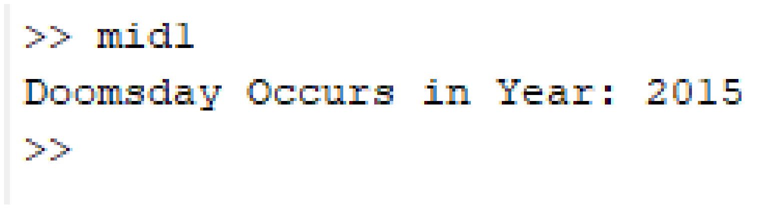
```
% 2b)
% Extend the Current Plot to the year 2020
new_domain = min(x):0.01:2020;
% Extend the Polynomial(s) to the year 2020
new_pop_pchip = pchip(x, y, new_domain);
new_pop_spl = spline(x, y, new_domain);
%{
Find "Doomsday"
Doomsday implies the day the year the pop. starts shrinking
%}
% Find the Max Population achieved
max_pop = max(new_pop_spl);
% Find the Year of the Max Population
max_pop_index = find(new_pop_spl == max_pop);
doomed_year = new_domain(max_pop_index);
fprintf('Doomsday Occurs in Year: %.0f\n', doomed_year);
% Plot the Polynomial to the Year 2020
figure;
hold on;
scatter(doomed_year, max_pop, 'ok', 'DisplayName', 'Doomsday
Occurs');
plot(new_domain, new_pop_pchip, '-r', 'DisplayName', 'Census
PChip');
plot(new_domain, new_pop_spl, '-b', 'DisplayName', 'Census
Spline');
hold off;
title('Question 2.b. - Census Polynomial up to 2020');
xlim([1910, 2020]);
xlabel('Date [1910, 2010]');
ylabel('U.S. Population (in Millions)');
legend('location','best');
```

(Graph is on the next page)

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions





```
>> mid1
Doomsday Occurs in Year: 2015
>>
```

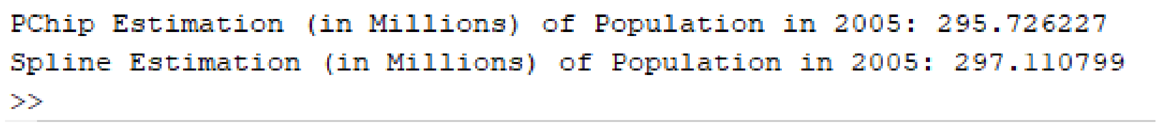
(mid1 is the name of my working matlab script)

c. Which of these interpolations would you trust? Which would you not trust, and why?

Between the pchip and spline interpolations shown in questions 2.a. and 2.b, I believe that despite the spline's mathematical precision, the pchip interpolation does a better job of accurately representing the change in population over time. Not only are these lines nearly-identical in graph 2.a, but also the spline graph appears to dip downwards around 2015- which provides a false representation of the change in population from 2010 to 2020. While it's clear that neither graph correctly estimates the exponential growth of population past the year 2010 (shown by both having a gradual decrease in population growth past the year 2010), I believe that the pchip interpolation provides a more rational estimation of the data than the spline interpolation does.

d.

```
% 2d)
% Find the Year 2005
index_2005 = find(domain == 2005);
% Find the Population Estimation in 2005
pchip_est = pop_pchip(index_2005);
spline_est = pop_spl(index_2005);
% Print Your Results
fprintf('\nPChip Estimation (in Millions) of Population in 2005:
%f\n', pchip_est);
fprintf('Spline Estimation (in Millions) of Population in 2005:
%f\n', spline_est);
```



```
PChip Estimation (in Millions) of Population in 2005: 295.726227
Spline Estimation (in Millions) of Population in 2005: 297.110799
>>
```

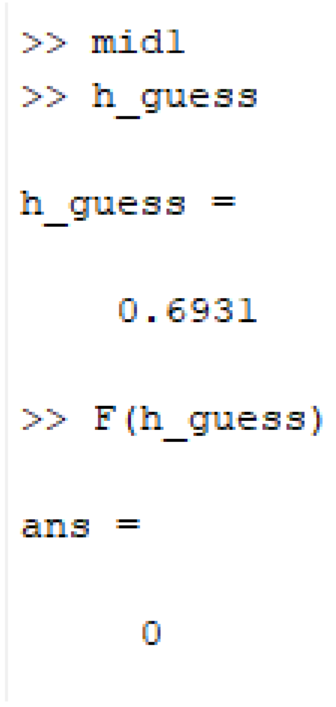
Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

3.

a.

```
% 3a)
% Original Function f, and it's two derivatives (df and d2f)
syms x;
f = 2 - exp(x);
df = diff(f, x);
d2f = diff(df, x);
% Halley's Iteration Formula
g = x - (2*f*df)/(2*(df)^2 - f*d2f);
% Matlab Function from Halley's Symbolic Representation
G = matlabFunction(g);
% Matlab Function for the Original Equation
F = matlabFunction(f);
% Boundaries for Halley's Method
h_guess = 1;
tolerance = eps;
h_iter = 0;
% Halley's Method
while(abs(F(h_guess)) > tolerance)
h_guess = G(h_guess); % Update the guess with G(h_guess)
h_iter = h_iter + 1;
end
```



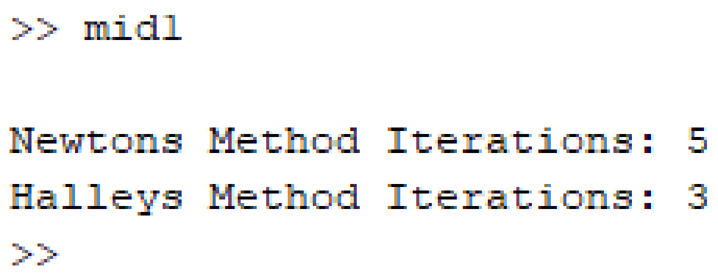
```
>> midl
>> h_guess

h_guess =

    0.6931

>> F(h_guess)

ans =

     0
```

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

b.

```
% 3b)
% Newton's Iterative Formula (lowercase is Symbolic, uppercase is
Matlab)
n = x - (f)/(df);
N = matlabFunction(n);
% Boundaries for Newton's Method (same initial guess of x = 1)
n_guess = 1;
n_iter = 0;
% Newton's Method
while(abs(F(n_guess)) > tolerance)
n_guess = N(n_guess); % Update the guess with N(n_guess)
n_iter = n_iter + 1;
end
fprintf('\nNewtons Method Iterations: %d\n', n_iter);
fprintf('Halleys Method Iterations: %d\n', h_iter);
```



```
>> mid1

Newtons Method Iterations: 5
Halleys Method Iterations: 3
>>
```

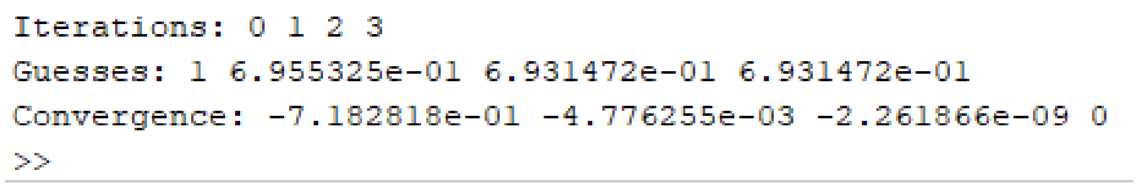
Since 3 is less than 5, Halley's Method calculates the zero of the function faster than Newton's Method. In this specific case, it took Halley's Method 60% of the time that it took Newton's Method to calculate the zero. Therefore, we can conclude that Halley's Method performs more efficiently than Newton's Method.

Amelia Rotondo
CWID: 887925113

Numerical Analysis: Midterm 1 Take-Home Solutions

c.

```
% 3c)
% Matrix to Find the Order of Convergence
order_conv = [];
order_conv(1, 1) = 0;
order_conv(2, 1) = 1;
order_conv(3, 1) = F(1);
% Halley's Method (With Deductions)
h_guess = 1;
h_iter = 0;
while(abs(F(h_guess)) > tolerance)
h_guess = G(h_guess); % Update the guess with G(h_guess)
h_iter = h_iter + 1;
order_conv(1, h_iter+1) = h_iter;
order_conv(2, h_iter+1) = h_guess;
order_conv(3, h_iter+1) = F(h_guess);
end
% Display the Order of Convergence
descriptions = {'Iterations:', 'Guesses:', 'Convergence:'};
for i = 1:size(order_conv, 1)
fprintf('%s ', descriptions{i});
fprintf('%d ', order_conv(i, :));
fprintf('\n');
end
```

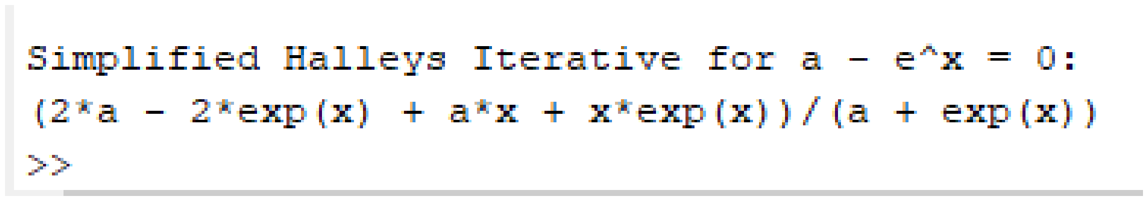


```
Iterations: 0 1 2 3
Guesses: 1 6.955325e-01 6.931472e-01 6.931472e-01
Convergence: -7.182818e-01 -4.776255e-03 -2.261866e-09 0
>>
```

This shows that the function converges from to 0 across 3 iterations. The row of guesses shows each guess that was used to determine the corresponding converging value. The row of convergence shows how the row of guesses are used to bring the initial guess from -0.7182818 to 0.

d.

```
% 3d)
% symbolic representations of a - e^x = 0
% most methods here are borrowed from 3a
syms x;
syms a;
f = a - exp(x);
df = diff(f, x);
d2f = diff(df, x);
g = x - (2*f*df)/(2*(df)^2 - f*d2f);
% Simplify the Symbolic Expression for Halley's Iterative Formula
g_simp = simplify(g);
% Display the Result
disp('Simplified Halleys Iterative for a - e^x = 0: ');
disp(char(g_simp));
```



```
Simplified Halleys Iterative for a - e^x = 0:
(2*a - 2*exp(x) + a*x + x*exp(x))/(a + exp(x))
>>
```

Since Halley's Iterative Formula is designed to find some value x_{n+1} from some given value x_n, we can substitute in these values to rewrite the simplified expression as follows:

$$x_{n+1} = x_n + \frac{2a-2e^{x_n}ax_n + xe^{x_n}}{a+e^{x_n}}$$

Which can then be simplified to:

$$x_{n+1} = x_n + 2\left(\frac{ae^{-x_n}-1}{ae^{-x_n}+1}\right)$$

Using natural log, the second segment of Halley's Iterative Formula could be simplified further to create a fraction of logarithms, which could then be used in-junction with the logarithmic Change of Base Rule to find specific logarithms of a.