ME 318 FINAL (TAKE-HOME) EXAM

Assigned on November 30th, 2020
Due on Friday, December 4th 2020, no later than 23:59 (11:59PM). Please turn in ALL your relevant work as a single pdf file submission. Format your exams as neat, professional reports and <u>do not</u> embed your answers as comments within codes.

Directions: You are to complete this exam entirely by yourself with no outside discussion or help from other individuals. The exam is due on Friday, December 4th, 2020 by 23:59 (11:59PM). You may use a calculator, Matlab programming environment, lab notes, homeworks and textbooks. If you are using any external resources, please credit them and refer to them properly. **Please turn in ALL your relevant work as a single pdf file submission. Include all m-files and code used to answer each problem EVEN IF NOT SPECIFICALLY REQUESTED. <u>Label and title all graphs</u>**. Attach all requested material **in a clear, ordered and organized manner**. Format your susmissions as professional, neat reports – do not submit your answers as comments embedded in code! The professor and TA-s will be available during the week to clarify whatever issues you have with the exam. Every effort will be made to answer e-mails within 24 hours.

Please sign the following contract after you have completed this exam only if it applies to you. Conduct yourself in a manner that will allow this contract to apply to you when completing all portions of this exam.

In accordance with ethical standards maintained by the University of Texas, the attached material represents original work prepared exclusively by me. I have neither offered, nor received help from classmates or other students while preparing this document.

Signature: [illegible] Date: 12/04/2020

Name (please print): Jingsi Zhou

UT EID: JZ24729

Lab time/unique number: Mon 6:30-8PM; 17460

Problem 1: Numerical Solution of an ODE IVP

(a)

$a_1 = 0.25m \qquad a_2 = 0.5m$

$m_1 = 4.0kg \qquad m_1 = 7.0kg$

$\theta_1(0) = 5^\circ \qquad \theta_2(0) = 0^\circ \qquad g = 9.81 m/s^2$

$\dot{\theta}_1(0) = 0 \qquad \dot{\theta}_2(0) = 0$

$$\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \dfrac{-g(2m_1+m_2)\sin\theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2\sin(\theta_1-\theta_2)m_2(\dot{\theta}_2^2 a_2 - \dot{\theta}_1^2 a_1 \cos(\theta_1-\theta_2))}{a_1(2m_1+m_2-m_2\cos(2(\theta_1-\theta_2)))} \\[2ex] \dfrac{2\sin(\theta_1-\theta_2)(\dot{\theta}_1^2 a_1(m_1+m_2) + g(m_1+m_2)\cos(\theta_1) + \ddot{\theta}_2^2 a_2 m_2 \cos(\theta_1-\theta_2))}{a_2(2m_1+m_2-m_2\cos(2(\theta_1-\theta_2)))} \end{bmatrix}$$

~~$\theta_1(\theta)$~~ $\theta_1 = z_1 \qquad \dot{\theta}_1 = z_3 \qquad \dot{\theta}_1 = z_1' = z_3$

$\theta_2 = z_2 \qquad \dot{\theta}_2 = z_4 \qquad \dot{\theta}_2 = z_2' = z_4$

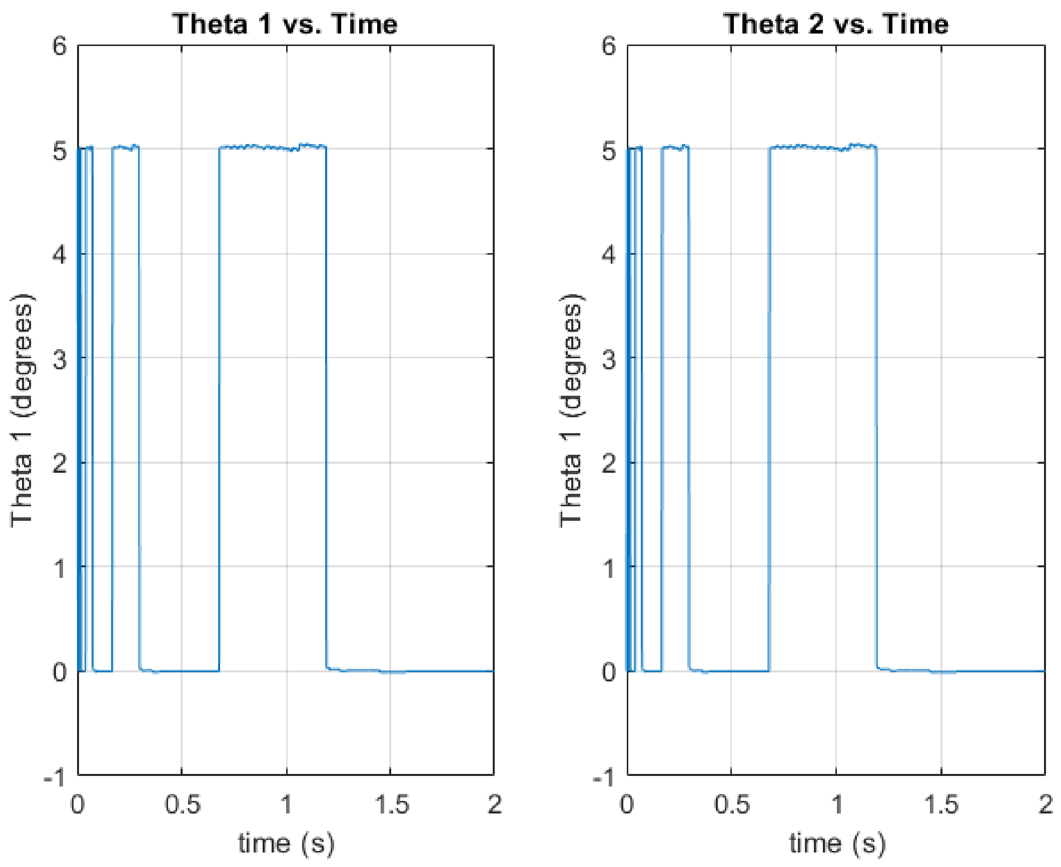
$$\begin{bmatrix} z_3' \\ z_4' \end{bmatrix} = \begin{bmatrix} \dfrac{-g(2m_1+m_2)\sin(z_1) - m_2 g \sin(z_1 - 2z_2) - 2\sin(z_1-z_2)m_2(z_4^2 a_2 - z_3^2 a_1 \cos(z_1-z_2))}{a_1(2m_1+m_2-m_2\cos(2(z_1-z_2)))} \\[2ex] \dfrac{2\sin(z_1-z_2)(z_3^2 a_1(m_1+m_2) + g(m_1+m_2)\cos(z_1) + z_4^2 a_2 m_2 \cos(\theta_1-\theta_2))}{a_2(2m_1+m_2-m_2\cos(2(z_1-z_2)))} \end{bmatrix}$$

Correction: theta1 should be z1 and theta2 should be z2.

(b) Using the Runge-Kutta formula from class, a function can be written that calculates all the k_n values and the next z value vector. This is done by using a for loop. For every element in t, the dz matrix can be obtained by using the initial z value vector. This dz vector is then used to calculate the k_n values in the Runge-Kutta formula. These k_n values are then used to find the next vector of z values for the given differential equations. After the for loop is completed for all the elements in the t vector, the subplot() function can be used to insert two graphs on the same figure. The first row of the z array holds the values for theta 1 while the second row holds the values for theta 2. These two data are gathered from the z matrix into two vectors, which are both graphed vs. time in two separate plots.

Because the z1' to z4' matrix was hard to linearise, I solved for dz directly using the given initial values. Then, for each iteration (next z vector), I used the new z value vector to calculate the next dz value.

Graph:



MATLAB Code for Problem 1:

```
clear
z(:, 1) = [5; 0; 0; 0];
t0 = 0;
h = 0.002;
t_end = 2;
t = t0:h:t_end;
for i = 1:numel(t)-1
    k1n = angles(z(:,i), t(i));
    k2n = angles(z(:,i) + h/2*k1n, t(i) + h/2);
    k3n = angles(z(:,i) + h/2*k2n, t(i) + h/2);
    k4n = angles(z(:,i) + h*k3n, t(i) + h);
    z(:, i+1) = z(i) + h/6*(k1n + 2*k2n + 2*k3n + k4n);
end

subplot(1, 2, 1)
a1 = plot(t, z(1,:));
xlabel('time (s)')
ylabel('Theta 1 (degrees)')
title('Theta 1 vs. Time')
grid on

subplot(1, 2, 2)
```

```
a2 = plot(t, z(2,:));
xlabel('time (s)')
ylabel('Theta 1 (degrees)')
title('Theta 2 vs. Time')
grid on

function dz = angles(z,t)
a1 = 0.25; m1 = 4.0;
a2 = 0.5; m2 = 7.0;
g = 9.81;
dz = [z(3);
        z(4);
(-g*(2*m1 + m2)*sind(z(1)) - m2*g*sind(z(1) - 2*z(2)) - 2*sind(z(1)-
z(2))*m2*(a2*((z(4))^2) - a1*((z(3))^2) * cosd(z(1) - z(2))))/(a1*(2*m1 + m2
- m2*cosd(2*(z(1)-z(2)))));
(2*(sind(z(1)-z(2)))*(((z(3))^2) * a1* (m1 + m2) + g*(m1 + m2)* (cosd(z(1)))
+ ((z(4))^2) * a2 * m2 * (cosd(z(1)-z(2)))))/(a2*(2*m1 + m2 -
m2*cosd(2*(z(1)-z(2)))))];
end
```

Problem 2: Numerical Integration

For this problem, I first decided on an integration method to use. I chose the Simpson method of Integration because in the given .mat file data, there are an even number of x values that are equidistant with h = 0.1. Also, looking at the provided plot, I thought the Simpson method would best represent the integral because of the quadratic shapes of the plots; Simpson method of numerical integration uses quadratic functions to approximate values of definite integrals. After deciding on the numerical integration method, I wrote the script in MATLAB for the Simpson method.

Implementing the method in MATLAB was quite straightforward. Using the given formula where $\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4 * \sum_{i=1,3,\ldots,N-1} f(x_i) + 2 * \sum_{i=2,4,\ldots,N-2} f(x_i) + f(b)\right]$, I wrote a corresponding function in MATLAB. The function takes in an array of x and y values and produces the integral result. This is done by calculating two sums — one for the even x values after x_0 (a) and one for the odd x values after x_0 (a). To calculate these sums, the variable for each sum is first initialized and set to zero. Then, for the odd x values, a for loop is used where the index begins at 2 (y_1 of the pdf_* values), is incremented by 2, and ends at length of y minus 1. For each index value (i), the total_odds variable is incremented by the value of pdf_* at an odd x value. Thus, at the termination of the for loop, the total sum for the pdf_* values at the odd x values is calculated properly. Also, the i variable in the for loop ends at length(y) – 1 to make sure that the last value of the y array is not counted in the summation process. Similarly, for the even x values, the index (i) begins at 3 (where x value and y value are x_2 and y_2, respectively) and is incremented in the same manner such that the last value, y(length(y)), is not included in the summation process. After these for loops calculate the sums of the pdf_* values for the odd and even x values, the formula is used to calculate the integral.

This function is then used in the script. First, the .mat file is loaded into the workspace. Then, the given CV formula is implemented in code. The pdf_fresh and pdf_fatigued values are multiplied (.*) and square rooted to give the pdf_combined array. The pdf_combined array is then used as input into the Simpson method function for the y value array. The same x values from the .mat file are used as the x input. Then, the function returns the CV value for the given Problem 2 .mat file. The performance CV was evaluated to be 0.6379.

MATLAB Code for Problem 2:

```
clear
load('TakeHomeExamProblem2.mat')

%Calculate Performance Confidence Value:
pdf_combined = sqrt(pdf_fresh .* pdf_fatigued);
CV = UltimateSimp(x, pdf_combined);
fprintf('\nCV = %.5f \n', CV)

%Simpson's Method of Integration is used. More accurate than trapezoidal,
%especially because the given data results in a series of quadratic-like
%functions that can easily be integrated using this method. Also x values
%are equidistant and there are an even number of them.
function area = UltimateSimp(x, y);
    total_odds = 0;
    total_evens = 0;
    h = x(2) - x(1);
    for i = 2:2:length(y) - 1
        total_odds = total_odds + y(i);
    end
    for i = 3:2:length(y) - 1
        total_evens = total_evens + y(i);
    end
    area = (h/3)*(y(1) + 4*total_odds + 2*total_evens + y(length(y)));
end
```

Command Window for Problem 2:

```
>> Problem2_Exam2

CV = 0.63793
```

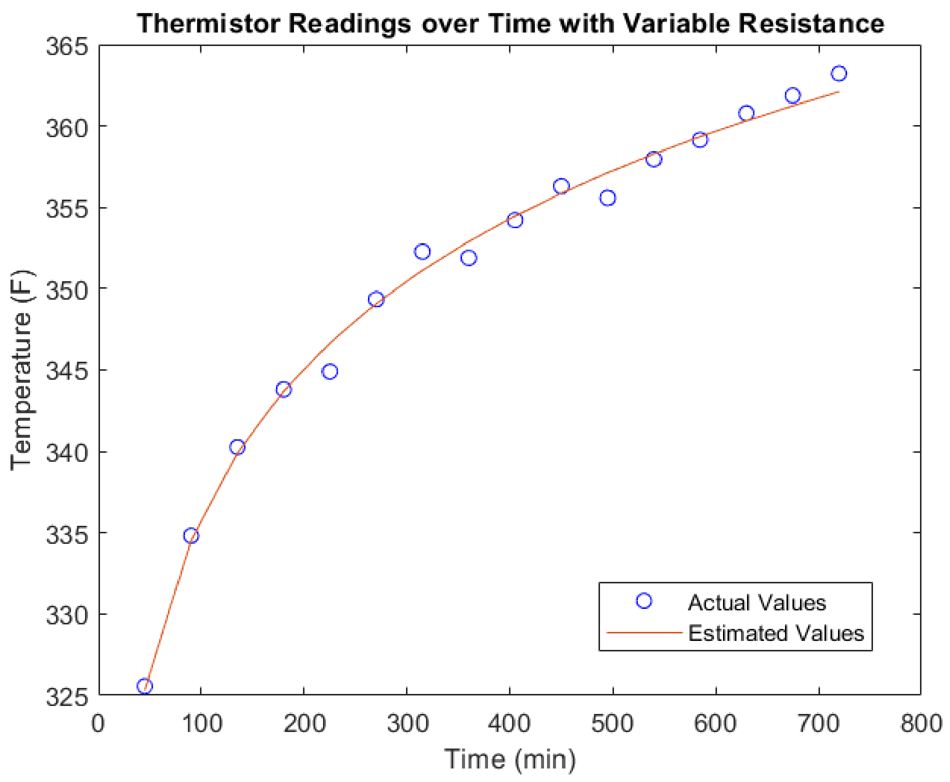
Problem 3: Thermistor Calibration

Given the linear approximation function that relates temperature and resistance along with measured values, the Stein-Hart Parameters were calculated. This was done by a simple system of equations calculation. The given function was rewritten for each set of values. Then, for each rewritten function, the variables a, b, and c were separated out into an array x such that x * A (the rewritten functions minus the variables) = b (1/the temperature values). By multiplying the inverse of A by b, the Stein-Hart parameters were obtained. **The values of a, b, and c were found to be 1.40186e-03, 2.36564e-04, and 1.01762e-07, respectively.**

The second part of this problem gives a linear equation. I assumed that $T_{internal}$ was the T calculated by the first given equation. This second equation can be written in the form y = mx + b, with $T_{initial}$ and h being b and m, respectively. The "x" value is $T_{ambient}$ * ln(t). Using these assumptions, $T_{initial}$ and h can be found using the linear fitting formula from class. The sum of x values, sum of squared x values, number of data points, sum of y values, and sum of product of x and y values were all calculated in the MATLAB script. Then, the matrices corresponding to the formula were initialized and filled with the corresponding values needed. **The equation returns an h value of 0.06802 and a $T_{initial}$ value of 274.82971F.**

The data points given in the table were graphed along with the numerically estimated curve.

<u>Graph:</u>



<u>MATLAB Code for Part 1 of Problem 3:</u>

```
clear
T1 = FtoK(32); R1 = 10030;
T2 = FtoK(77); R2 = 3070;
T3 = FtoK(212); R3 = 207.9;
A = [1 log(R1) (log(R1))^3;
        1 log(R2) (log(R2))^3;
            1 log(R3) (log(R3))^3]
b = [1/T1; 1/T2; 1/T3]
format shortE
x = inv(A) * b
format short

function T_Kelvin = FtoK(temp)
    T_Kelvin = (temp-32)*(5/9) + 273.15;
end
```

<u>Command Window for Part 1 of Problem 3:</u>

```
>> Problem3_Exam2

x =

   1.4019e-03
   2.3656e-04
   1.0176e-07
```

<u>MATLAB Code for Part 2 of Problem 3:</u>

```
clear

load('Part1Problem3Workspace.mat', 'x')
time = [45, 90, 135, 180, 225, 270, 315, 360, 405, 450, 495,...,
             540, 585, 630, 675, 720];
resistance = [1008, 718, 593, 525, 506, 436, 396, 401, ...,
             372, 348, 356, 330, 318, 302, 292, 280];
a = x(1);
b = x(2);
c = x(3);
T_int = [];
for i = 1:length(resistance)
    T_int(i) = 1/(a + b*log(resistance(i)) + c * (log(resistance(i)))^3);
end

x_vals = 195.*log(time);
x_vals_squared = x_vals.^2;
sum_xvals = sum(x_vals);
sum_sqvals = sum(x_vals_squared);
N = length(time);

A = [sum_sqvals sum_xvals;
    sum_xvals N];

product_x_y = x_vals .* T_int;
sum_products = sum(product_x_y);
sum_y = sum(T_int);
b = [sum_products; sum_y];
```

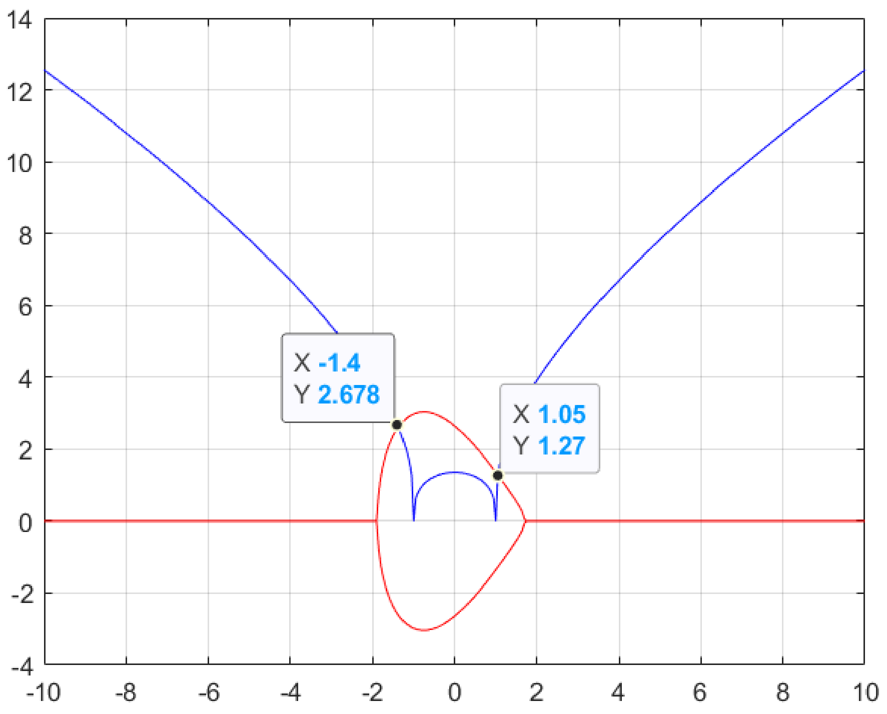
```
z = inv(A) * b;

for i = 1:length(z)
    if i == 1
        fprintf('%-12s', 'h = ')
    else
        fprintf('%-12s', 'initial T = ')
    end
    fprintf('%10.5f', z(i));
    fprintf('\n');
end

a1 = plot(time, T_int, 'ob', 'DisplayName', 'Actual Values');
hold on
T_int_est =  z(2) + 195*z(1)*log(time);
a2 = plot(time, T_int_est, 'DisplayName', 'Estimated Values');
xlabel('Time (min)')
ylabel('Temperature (F)')
title('Thermistor Readings over Time with Variable Resistance')
legend
```

<u>Command Window for Part 2 of Problem 3:</u>

```
>> Problem3_Exam2_Part2
h =            0.06802
initial T =  274.82971
```

Problem 4: System of nonlinear equations

(a)



(b) Taking an initial guess as input, the Newton-Raphson method function utilizes the formula we learned in class to refine the estimate of point intersection of the two functions. The Jacobian matrix is calculated and implemented as a function that takes in an x and y value. Functions for the two functions are written; these take in an x and y value and returns the value of f(x,y) for each function. These three functions are then used in the Vectorial Newton-Raphson method function. The Newton-Raphson function utilizes a while loop that uses a convergence condition of evaluating the two given functions such that they do not differ from zero by more than 5 decimal places. This function then returns a new x and y value that meets the convergence criteria and the number of loops the function went through to get to the result.

(c) The two numerically calculated intersection coordinates are: (-1.37375, 2.60825) and (1.05110, 1.27982).

MATLAB Code for Problem 4:

```
x = [-10:0.05:10];
y_1 = sqrt((1/2)*(4.*x.^3 - 6.*exp(x) - 4*x + 20));
y_1_2 = -sqrt((1/2)*(4.*x.^3 - 6.*exp(x) - 4*x + 20));
y_2 = (20.*x.^2 - 20).^(1/3);
plot(x, y_1, '-r')
hold on
plot(x, y_1_2, '-r')
plot(x, y_2, '-b')
grid on

x = [];
y = [];
disp("Calculate the intersection coordinates:")
n = input("How many estimate points do you have? ");
for i = 1:n
    disp(["Point " + num2str(i) + ": "]);
    x(i) = input("x value: ");
    y(i)= input("y value: ");
end

for i = 1:n
    Point = VectNewR(x(i), y(i));
    fprintf('Point ')
    fprintf(num2str(i))
    fprintf(' = ')
    fprintf('%.5f', Point(1))
    fprintf(', %.5f \n', Point(2))
end

%{
Point1 = VectNewR(-1.4, 2.678);
Point2 = VectNewR(1.05, 1.27);
fprintf('Point 1 = ')
fprintf('%.5f', Point1(1))
fprintf(', %.5f', Point1(2))
fprintf('\nPoint 2 = ')
fprintf('%.5f', Point2(1))
fprintf(', %.5f \n', Point2(2))
%}

function yep = Jacobian(x, y);
    df_1_dx = 12*x^2 - 6*exp(x) - 4;
    df_1_dy = -4*y;
    df_2_dx = 40*x;
    df_2_dy = -3*y^2;
    yep = [df_1_dx df_1_dy; df_2_dx df_2_dy];
end

function out = f_1(x,y)
    out = 4*x^3 - 6*exp(x) - 4*x + 20 - 2*y^2;
end

function out = f_2(x,y)
    out = 20*x^2 - 20 - y^3;
```

```
end

function [out, count] = VectNewR(x_guess, y_guess)
    x_0 = x_guess;
    y_0 = y_guess;
    count = 0;
    while sqrt((f_1(x_0, y_0))^2 + (f_2(x_0, y_0))^2) > 10^-5
        count = count + 1;
        A = [x_0; y_0];
        J = Jacobian(x_0, y_0);
        F = [f_1(x_0, y_0); f_2(x_0, y_0)];
        var_New = A - (inv(J))*F;
        x_nm1 = x_0;
        y_nm1 = y_0;
        x_0 = var_New(1);
        y_0 = var_New(2);
    end
out = [x_0; y_0];
count = count;
end
```

<u>Command Window for Problem 4:</u>

```
>> Problem4a
Calculate the intersection coordinates:
How many estimate points do you have? 2
Point 1:
x value: -1.4
y value: 2.678
Point 2:
x value: 1.05
y value: 1.27
Point 1 = -1.37375, 2.60825
Point 2 = 1.05110, 1.27982
```