
HW1 EMA 471 Intermediate Problem Solving for Engineers

SPRING 2016
ENGINEERING MECHANICS DEPARTMENT
UNIVERSITY OF WISCONSIN, MADISON

INSTRUCTOR: PROFESSOR ROBERT J. WITT

BY

NASSER M. ABBASI

DECEMBER 30, 2019

Contents

0.1	Problem 1	3
0.1.1	Euler method	3
0.1.2	Modified Euler method	3
0.1.3	Second order Runge-Kutta	3
0.1.4	Results	4
0.1.5	Discussion of results	4
0.1.6	Source code listing	4
0.2	Problem 2	6
0.2.1	Results	7
0.2.2	Source code listing	8
0.3	Problem 3	9
0.3.1	Results	10
0.3.2	Source code listing	11

0.1 Problem 1

PROBLEM DESCRIPTION

(1) (12 pts) Consider the first order IVP

$$\frac{dy}{dt} = t^2 - t, \quad y(0) = 1$$

This is a relatively straightforward ODE that has an analytical solution. Find the analytical solution first to serve as a comparison with your numerical solutions. Using a step size $h = 0.1$, advance this solution to $t = 4$ using the Euler, Modified Euler and 2nd order Runge-Kutta methods we discussed in Exercise 1. Compare your numerical results to the analytical solution.

SOLUTION

$$\begin{aligned} y'(t) &= f(t) \\ &= t^2 - t \end{aligned} \tag{1}$$

This ODE is separable. Integrating both sides gives

$$y = \frac{t^3}{3} - \frac{t^2}{2} + C$$

Where C is the constant of integration which is found from the initial conditions. At $t = 0$, we are given $y(0) = 1$. This results in $C = 1$. The solution becomes

$$y = 1 + \frac{t^3}{3} - \frac{t^2}{2}$$

Matlab program was written to implements Euler, modified Euler and Runge-Kutta using time spacing of 0.1 and was run to 4 seconds.

0.1.1 Euler method

$$y_{n+1} = y_n + hy'_n + O(h^2)$$

In the above, y_0 was taken from given initial conditions and $y'_n = f_n$ is the RHS of (1) evaluated at each time step. h is the time step used (which is 0.1 seconds in this problem). Hence $t_n = nh$. Euler method has local error (per step) $O(h^2)$ and an overall global error $O(h)$.

0.1.2 Modified Euler method

$$y_{n+1} = y_n + h \left(\frac{y'_n + y'_{n+1}}{2} \right) + O(h^3)$$

Modified Euler method has local error (per step) $O(h^3)$ and overall global error $O(h^2)$, therefore it is more accurate than the standard Euler method. In this problem, the RHS $y'_n = f_n(t)$ only depends on t and not on y_n .

0.1.3 Second order Runge-Kutta

This method has local error $O(h^3)$ and global error $O(h^2)$

$$\begin{aligned} t_n &= nh \\ k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \alpha h, y_n + \beta k_1) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

In this problem y do not appear in the RHS (hence β is not used). The above reduces to

$$\begin{aligned} k_1 &= hf(t_n) \\ k_2 &= hf(t_n + \alpha h) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

Using $a = \frac{2}{3}, b = \frac{1}{3}, \alpha = \frac{3}{2}, \beta = \frac{3}{2}$, (as was done in exercise one handout) the above now becomes

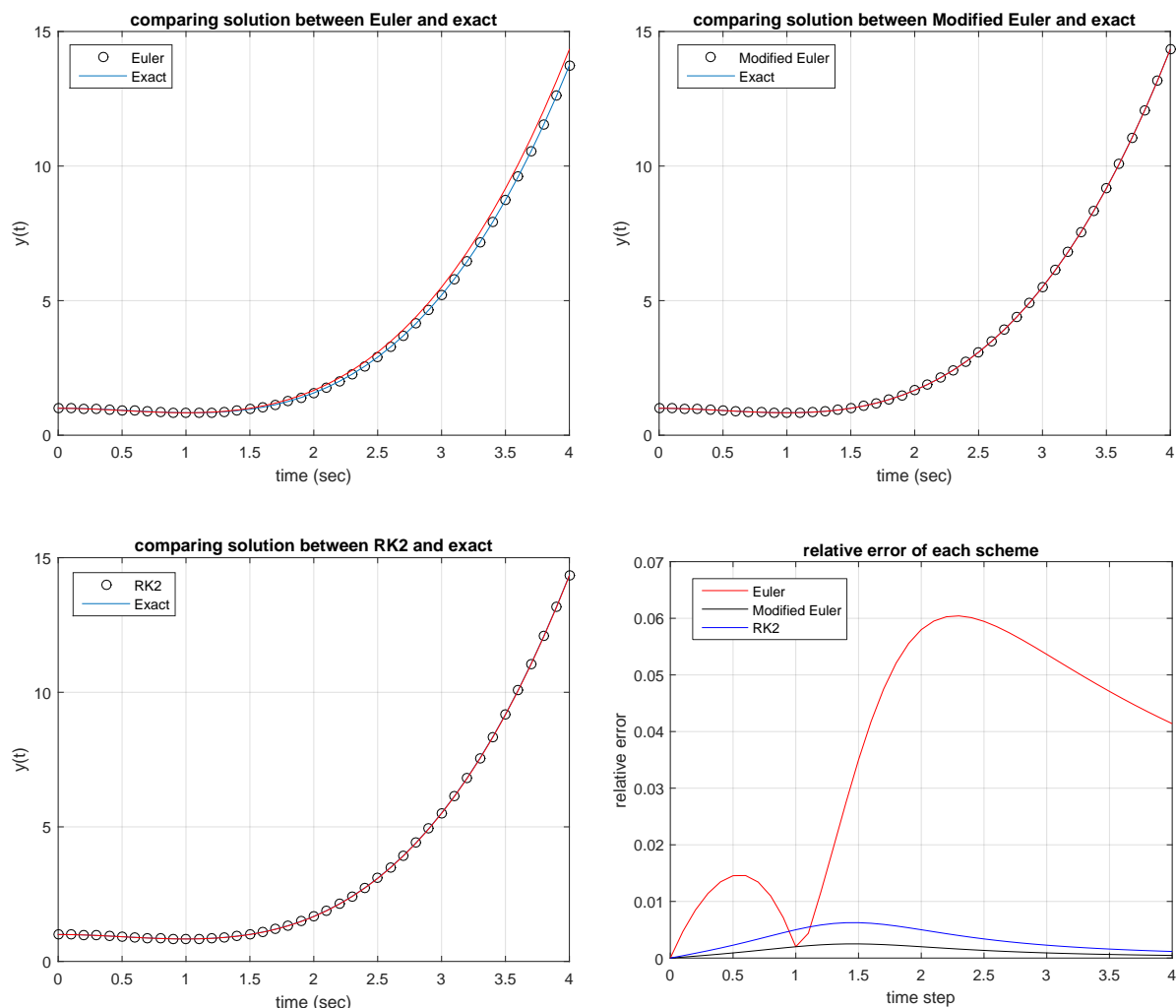
$$\begin{aligned} k_1 &= hf(nh) \\ k_2 &= hf\left(nh + \frac{3}{2}h\right) = hf\left(\left(n + \frac{3}{2}\right)h\right) \\ y_{n+1} &= y_n + \frac{2}{3}hf(nh) + \frac{1}{3}hf\left(\left(n + \frac{3}{2}\right)h\right) \\ &= y_n + \frac{2}{3}h\left[f(nh) + \frac{1}{2}f\left(\left(n + \frac{3}{2}\right)h\right)\right] \end{aligned}$$

The following is a summary table of the three methods.

scheme name	main computation	global error
Euler	$y_{n+1} = y_n + hy'_n$	$O(h)$
Modified Euler	$y_{n+1} = y_n + \frac{h}{2}(y'_n + y'_{n+1})$	$O(h^2)$
RK2	$y_{n+1} = y_n + \frac{2}{3}h\left(y'_n + \frac{1}{2}y'_{n+\frac{3}{2}}\right)$	$O(h^2)$

0.1.4 Results

Four plots were generated. three plots to show the solution by each scheme next to the exact solution. The fourth plot shows the relative error of each scheme. The relative error was found by calculating $\frac{|\text{exact solution}-\text{numerical}|}{|\text{exact}|}$ at each time instance.



0.1.5 Discussion of results

Euler method was least accurate as expected since it has global error $O(h)$. There is no large difference between RK2 and modified Euler since both have global error $O(h^2)$.

0.1.6 Source code listing

```

1 function nma_EMA_471_HW1_part_a
2 %This function solves HW1, part(a). EMA 471, spring 2016, UW Madison
3 %Please see report for detailed information about the problem
4 %by Nasser M. Abbasi
5

```

```

6 clear; close all;
7
8 %First call is to initialize data before each scheme run is made
9 [h,t,yExact,y] = initializeData();
10 yEuler      = Euler(h,t,y,yExact);
11
12 [h,t,yExact,y] = initializeData();
13 yModifiedEuler = modifiedEuler(h,t,y,yExact);
14
15 [h,t,yExact,y] = initializeData();
16 yRK2          = RK2(h,t,y,yExact);
17
18 %Now we Plot the relative error for each scheme, using returned results
19 figure();
20 plot(t,abs(yEuler-yExact)./abs(yExact),'r', ...
21      t,abs(yModifiedEuler-yExact)./abs(yExact),'k',...
22      t,abs(yRK2-yExact)./abs(yExact),'b');
23
24 title('relative error of each scheme');
25 xlabel('time (sec)'); ylabel('relative error');
26 legend('Euler','Modified Euler','RK2','Location','NorthWest');
27 grid;
28 end
29
30 %-----
31 %Implements basic Euler method
32 function y = Euler(h,t,y,yExact)
33 %y(1) has been initialized to correct value by caller
34 %
35 % o-----o-----o-----.....
36 % t=0      t=h      t=2h
37 % y(1)     y(2)     y(3)
38 %
39 %This function starts computing y(2),y(3),....
40
41 currentTime = 0;
42 for n = 2:length(y)
43     f      = rhs(currentTime); %RHS of the ODE
44     y(n)   = y(n-1)+h*f;      %Euler scheme
45     currentTime = currentTime + h;
46 end
47
48 makePlots(t,y,yExact,'Euler');
49 end
50
51 %-----
52 %Implements Modified Euler solver
53 function y = modifiedEuler(h,t,y,yExact)
54 currentTime = 0;
55
56 %y(1) has been initialized to correct value by caller
57 for n = 2:length(y)
58     fn1    = rhs(currentTime);
59     fn2    = rhs(currentTime+h);
60     f      = (fn1+fn2)/2;
61     y(n)   = y(n-1)+h*f;
62     currentTime = currentTime + h;
63 end
64
65 makePlots(t,y,yExact,'Modified Euler');
66 end
67
68 %-----
69 %Implements RK2 solver
70 function y = RK2(h,t,y,yExact)
71 currentTime = 0;
72 a          = 2/3;
73 b          = 1/3;
74 alpha     = 3/2; %RK2 parameters used
75 %beta     = 3/2; %not needed, since RHS f(.) do not depend on y.
76
77 %y(1) has been initialized to correct value by caller

```

```

78 for n = 2:length(y)
79     k1      = h*rhs(currentTime);
80     k2      = h*rhs(currentTime+alpha*h);
81     y(n)    = y(n-1)+a*k1+b*k2;
82     currentTime = currentTime + h;
83 end
84
85 makePlots(t,y,yExact,'RK2');
86 end
87
88 %-----
89 %Called to initialize data and counters before each solver is called
90 function [h,t,yExact,y] = initializeData()
91
92 h      = 0.1;      %time step we are asked to use
93 t      = (0:h:4)';
94 y      = zeros(length(t),1);
95 y(1)   = 1;       %initial conditions, from problem statement
96 yExact = 1+t.^3/3-t.^2/2;
97 end
98
99 %-----
100 %Called to generate plots from each solver
101 function makePlots(t,y,yExact,schemeName)
102
103 figure();
104 plot(t,y,'ok',t,y); hold on;
105 plot(t,yExact,'r');
106 legend(schemeName,'Exact','Location','NorthWest');
107 title(sprintf('comparing solution between %s and exact',schemeName));
108 xlabel('time (sec)'); ylabel('y(t)');
109 grid;
110
111 end
112
113 %-----
114 %RHS function, called by each scheme solver. Notice only t appears in RHS
115 function v = rhs(t)
116     v = t^2-t;
117 end

```

0.2 Problem 2

PROBLEM DESCRIPTION

(2) (12 pts) Consider the second order system of equations:

$$\begin{aligned} \frac{dx}{dt} &= xy + t, & x(0) &= 1 \\ \frac{dy}{dt} &= ty + x, & y(0) &= -1 \end{aligned}$$

As this is a non-linear system of equations, it's unlikely that you will be able to find an analytical solution. You can, however, construct a Taylor series expansion in the neighborhood of $t = 0$ that will be a reasonable approximation of the solution as long we include enough terms in the expansion and don't wander too far away from $t = 0$. Construct Taylor series expansions out to terms t^5 for $x(t)$ and $y(t)$ and plot these approximate solutions over the interval $0 \leq t \leq 0.5$. Then use `ode45` to solve the system over the interval $0 \leq t \leq 2$. Plot your numerical solutions on the same plot as the Taylor series expansions and discuss your results.

SOLUTION

$$\begin{aligned} x'(t) &= xy + t \\ y'(t) &= ty + x \end{aligned}$$

With $x(0) = 1, y(0) = -1$.

The following calculation shows the evaluation of all the derivatives needed for use with Taylor expansion. The expansion is around $t = 0$. This table gives the result.

derivative
$x'(t) = xy + t$
$y'(t) = ty + x$
$x''(t) = x'y + xy' + 1$
$y''(t) = y + ty' + x'$
$x'''(t) = x''y + x'y' + x'y' + xy''$
$y'''(t) = y' + y' + ty'' + x''$
$x^{(4)}(t) = x'''y + x''y' + x''y' + x'y'' + x''y' + x'y'' + x'y'' + xy'''$
$y^{(4)}(t) = y'' + y'' + y'' + ty''' + x'''$
$x^{(5)}(t) = x^{(4)}y + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y' + x^{(4)}y'$
$y^{(5)}(t) = y''' + y''' + y''' + y''' + ty^{(4)} + x^{(4)}$

The numerical value of the above derivatives at $t = 0$ is now calculated and given in another table below

	derivative at $t = 0$	value
$x'(0)$	$(1)(-1) + 0$	-1
$y'(0)$	$(0)(-1) + 1$	1
$x''(0)$	$(-1)(-1) + (1)(1) + 1 = 1 + 1 + 1$	3
$y''(0)$	$(-1) + 0(1) - 1$	-2
$x'''(0)$	$(3)(-1) + (-1)(1) + (-1)(1) + (1)(-2)$	-7
$y'''(0)$	$1 + 1 + 0 + 2$	4
$x^{(4)}(0)$	$(-7)(-1) + 2 + 2 + (-1)(-2) + 2 + (-1)(-2) + (-1)(-2) + 4$	23
$y^{(4)}(0)$	$(-2) + (-2) + (-2) + 0 + (-7)$	-13
$x^{(5)}(0)$	$(-13)(-1) + 4 + 4 + (2)(-2) + (-7) + (2)(-2) + (2)(-2) + (-1)(4) + (-7) + (2)(-2) + (2)(-2) + 4 + (2)(-2) + 4 + 4 - 13$	-22
$y^{(5)}(0)$	$(4) + (4) + (4) + (4) + 0 + 23$	39

The Taylor expansion for $x(t), y(t)$ is

$$x(t) = x(0) + tx'(0) + \frac{t^2}{2}x''(0) + \frac{t^3}{3!}x'''(0) + \frac{t^4}{4!}x^{(4)}(0) + \frac{t^5}{5!}x^{(5)}(0) + O(t^6)$$

$$y(t) = y(0) + ty'(0) + \frac{t^2}{2}y''(0) + \frac{t^3}{3!}y'''(0) + \frac{t^4}{4!}y^{(4)}(0) + \frac{t^5}{5!}y^{(5)}(0) + O(t^6)$$

Applying values from the table above gives

$$x(t) = 1 - t + \frac{3}{2}t^2 - 7\frac{t^3}{3!} + 23\frac{t^4}{4!} - 22\frac{t^5}{5!} + O(t^6)$$

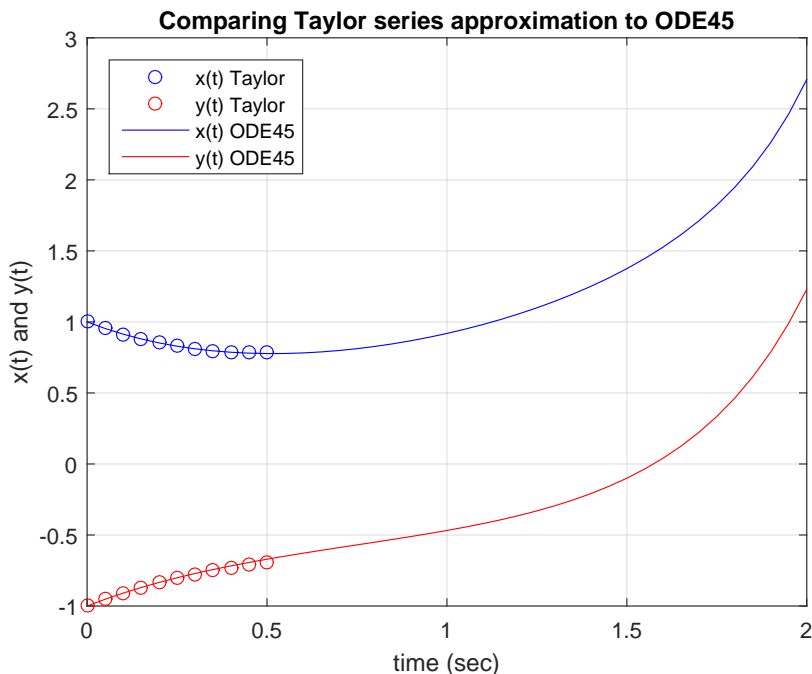
$$= 1 - t + \frac{3}{2}t^2 - \frac{7}{6}t^3 + \frac{23}{24}t^4 - \frac{22}{120}t^5 + O(t^6)$$

$$y(t) = -1 + t - t^2 + 4\frac{t^3}{3!} - 13\frac{t^4}{4!} + 39\frac{t^5}{5!} + O(t^6)$$

$$= -1 + t - t^2 + \frac{2}{3}t^3 - \frac{13}{24}t^4 + \frac{39}{120}t^5 + O(t^6)$$

0.2.1 Results

The following diagram shows the solution using Taylor series approximation and using ODE45

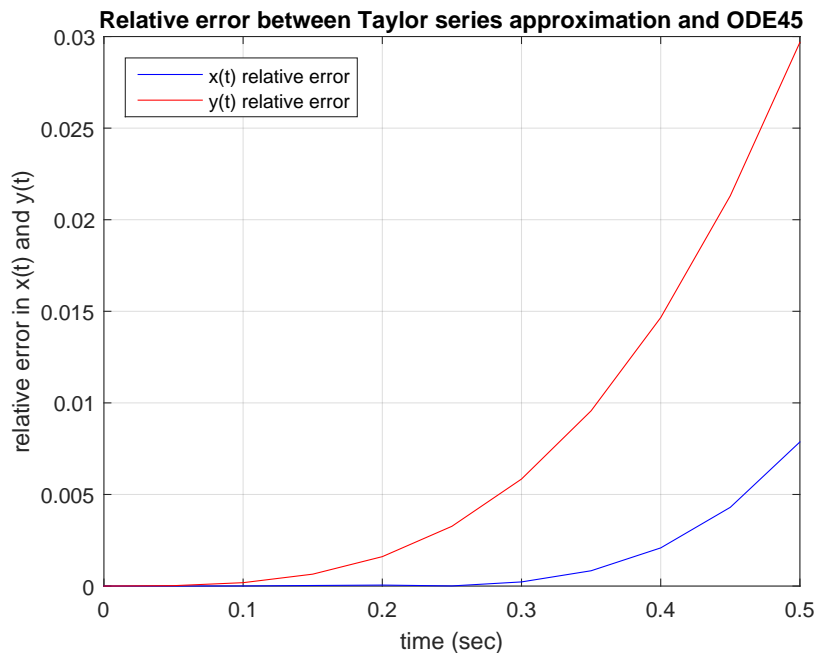


We see from the above that Taylor series approximation was good for only small distance from the

expansion point, $t = 0$. The $x(t)$ solution using Taylor became worst faster than the $y(t)$ solution did. The relative error was computed and plotted for up to $t = 0.5$ to better compare the results.

The relative error for both $x(t)$ and $y(t)$ was computed using

$$\frac{|\text{ODE45 solution} - \text{Taylor solution}|}{|\text{ODE45 solution}|}$$



The above plot shows that the error compared to ODE45 increased as the distance from the expansion point becomes larger with $y(t)$ solution showing a worst approximation than $x(t)$.

0.2.2 Source code listing

```

1 function nma_EMA_471_HW1_part_b
2 %This function solves HW1, problem 2. EMA 471, spring 2016,
3 %UW Madison see report for detailed information about the problem
4 %by Nasser M. Abbasi
5
6 close all;
7
8 h = 0.05; %step size
9 t1 = 0:h:0.5; %time for the Taylor series
10 t2 = 0:h:2; %for ODE45 we are asked to go for 2 seconds.
11
12 [xApprox,yApprox] = doTaylorApproximation(t1);
13 [xODE45,yODE45] = doODE45(t2);
14
15 %done, now plot and compare.
16 compare(t1,xApprox,yApprox,t2,xODE45,yODE45);
17 end
18
19 %-----
20 %Do the Taylor series approximation
21 function [xApprox,yApprox] = doTaylorApproximation(t)
22 xApproxF = @(t) 1 - t + (3/2)* t.^2 - 7/6 * t.^3 + 23/24 * ...
23 t.^4 - 22/120 * t.^5;
24 yApproxF = @(t)-1 + t - t.^2 + 2/3 * t.^3 - 13/24 *...
25 t.^4 + 39/120 * t.^5;
26
27 xApprox = xApproxF(t);
28 yApprox = yApproxF(t);
29 end
30
31 %-----
32 %Do ODE45. This uses Matlab's ode45 to solve the problem.
33 function [xODE45,yODE45] = doODE45(t)
34 xInitial = 1;
35 yInitial = -1;
36
37 [~,v] = ode45( @rhs, t, [xInitial yInitial] );

```



```

38 %extract solutions
39 xODE45 = v(:,1);
40 yODE45 = v(:,2);
41
42 %internal function, for RHS
43 function v = rhs(t,X)
44     x = X(1);
45     y = X(2);
46     v = [x*y+t;t*y+x];
47 end
48 end
49
50 %-----
51 %function to compare Taylor series with ODE 45 output
52 function compare(t1,xApprox,yApprox,t2,xODE45,yODE45)
53
54 figure();
55 plot(t1,xApprox,'bo',t1,yApprox,'ro'); hold on;
56 plot(t2,xODE45,'b',t2,yODE45,'r');
57
58 legend('x(t) Taylor','y(t) Taylor','x(t) ODE45','y(t) ODE45',...
59         'Location','NorthWest');
60
61 title('Comparing Taylor series approximation to ODE45');
62 xlabel('time (sec)'); ylabel('x(t) and y(t)');
63 grid;
64
65 figure();
66 plot(t1,abs(xODE45(1:length(t1))-xApprox')./...
67         abs(xODE45(1:length(t1))), 'b',...
68         t1,abs(yODE45(1:length(t1))-yApprox')./...
69         abs(yODE45(1:length(t1))), 'r');
70 title('Relative error between Taylor series approximation and ODE45');
71 xlabel('time (sec)');
72 ylabel('relative error in x(t) and y(t)');
73 legend('x(t) relative error','y(t) relative error',...
74         'Location','NorthWest');
75 grid;
76 end

```

0.3 Problem 3

PROBLEM DESCRIPTION

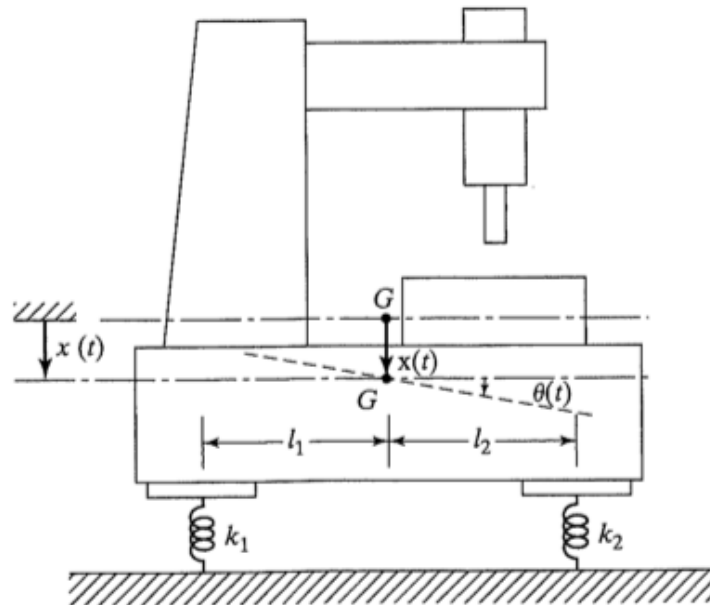
(3) (16 pts) Note: This is a problem you might encounter in EMA 545. A machine tool is mounted on two nonlinear elastic mounts, as shown in the figure at the top of the next page. The equations of motion, in terms of the coordinates $x(t)$ and $\theta(t)$, are given by:

$$m\ddot{x} + k_{11}(x - l_1\theta) + k_{12}(x - l_1\theta)^3 + k_{21}(x + l_2\theta) + k_{22}(x + l_2\theta)^3 = 0$$

$$J_o\ddot{\theta} - k_{11}(x - l_1\theta)l_1 - k_{12}(x - l_1\theta)^3l_1 + k_{21}(x + l_2\theta)l_2 + k_{22}(x + l_2\theta)^3l_2 = 0$$

Here m is the mass and J_o is the mass moment of inertia about G of the machine tool. Using ode45, find $x(t)$ and $\theta(t)$ over the interval $0 \leq t \leq 10$ s using the initial conditions:

$$x(0) = 0, \quad \dot{x}(0) = 10 \text{ mm/s}, \quad \theta(0) = 0, \quad \dot{\theta}(0) = 10 \text{ mrad/s}$$



Parameters in the governing equations have values: $m = 1000 \text{ kg}$, $J_0 = 2500 \text{ kg-m}^2$, $l_1 = 1 \text{ m}$, $l_2 = 1.5 \text{ m}$, $k_{11} = 40 \text{ kN/m}$, $k_{12} = 10 \text{ kN/m}^3$, $k_{21} = 50 \text{ kN/m}$, $k_{22} = 5 \text{ kN/m}^3$.

NOTE: The governing equations are equations of motion based on $\Sigma F = ma_G$, $\Sigma M_G = I\alpha$, so **units are important**. The values for the parameters given are in SI units, so be sure everything you input into your script is consistent with that system of units.

Plot x and θ over the interval with x in [mm] and θ in [mrad]. What are the peak amplitudes of x and θ over the interval?

SOLUTION

$$mx'' + k_{11}(x - l_1\theta) + k_{12}(x - l_1\theta)^3 + k_{21}(x + l_2\theta) + k_{22}(x + l_2\theta)^3 = 0 \quad (1)$$

$$J_0\theta'' - k_{11}(x - l_1\theta)l_1 - k_{12}(x - l_1\theta)^3l_1 + k_{21}(x + l_2\theta)l_2 + k_{22}(x + l_2\theta)^3l_2 = 0$$

The first step is to convert the two second order differential equations to a set of four first order differential equations, since ODE numerical solvers work with first order ode's. We need to select the states to use. Using

$$x_1 = x$$

$$x_2 = x'$$

$$x_3 = \theta$$

$$x_4 = \theta'$$

After taking time derivatives, the above becomes

$$\dot{x}_1 = x' = x_2$$

$$\dot{x}_2 = x'' = -\frac{1}{m} [k_{11}(x_1 - l_1x_3) + k_{12}(x_1 - l_1x_3)^3 + k_{21}(x_1 + l_2x_3) + k_{22}(x_1 + l_2x_3)^3]$$

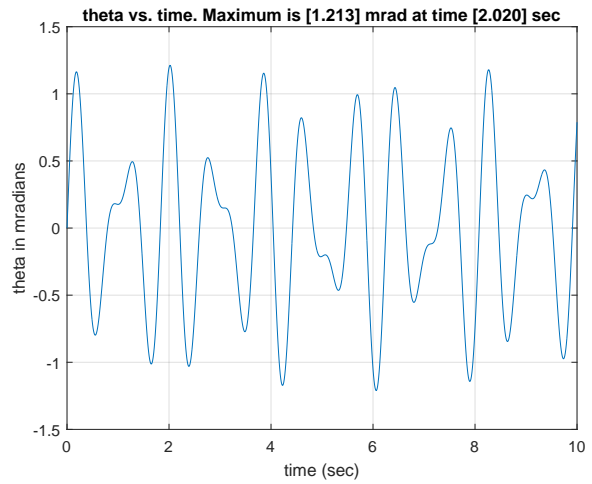
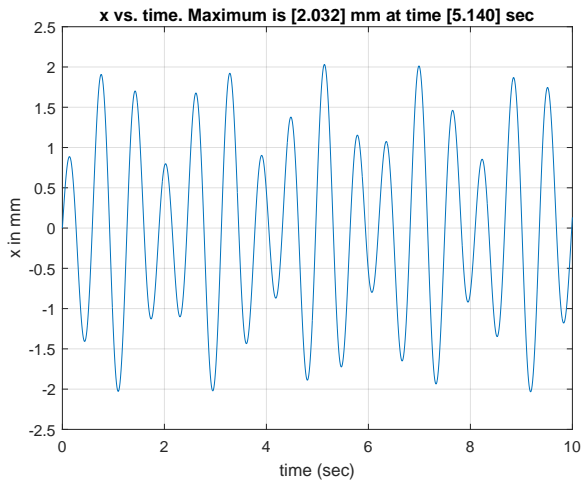
$$\dot{x}_3 = \theta' = x_4$$

$$\dot{x}_4 = \theta'' = -\frac{1}{J_0} [-k_{11}(x_1 - l_1x_3)l_1 - k_{12}(x_1 - l_1x_3)^3l_1 + k_{21}(x_1 + l_2x_3)l_2 + k_{22}(x_1 + l_2x_3)^3l_2]$$

The above vector \dot{X} in the LHS, is what the Matlab ode45 function will return when called. The following shows the Matlab implementation and the plots generated. It was found that maximum displacement is $x_{\max} = 2.032 \text{ mm}$ and occurred at $t = 5.14$ seconds. For the angle, the maximum angular displacement was $\theta_{\max} = 1.213 \text{ mrad}$ (or 0.069 degree) and occurred at $t = 2.02$ seconds.

0.3.1 Results

The following are the x and θ solutions plots for $t = 10$ seconds. The Matlab source code used is given in the next section.



0.3.2 Source code listing

```

1 function nma_EMA_471_HW1_part_c
2 %This function solves HW1, part c. EMA 471, spring 2016,
3 %UW Madison report contains detailed information about the problem
4 %by Nasser M. Abbasi
5
6 close all;
7
8 %physical parameters of problem
9 m = 1000; %kg
10 J0 = 2500; %kg-m^3
11 L1 = 1; %meter
12 L2 = 1.5; %meter
13 k11 = 40*10^3; %N/m
14 k12 = 10*10^3; %N/m^3
15 k21 = 50*10^3; %N/m
16 k22 = 5*10^3; %N/m
17
18 %initialization parameters and initial conditions
19 h = 0.01; %step size. ODE45 did not work
20 %well with large step
21 t = 0:h:10; %time for ODE45
22 xInitial = 0; %meter
23 xVInitial = 10*10^(-3); %10 mm/sec
24 thetaInitial = 0; %radians
25 thetaVInitial = 10*10^(-3); %mrad/sec
26
27 %call ODE45 to numerically solve the equations of motions
28 [t,sol] = ode45(@rhs, t, ...
29 [xInitial xVInitial thetaInitial thetaVInitial]);
30
31 %Extract the first and third columns. These represent x
32 %and theta solutions
33 x = sol(:,1);
34 [value,I] = max(x);
35
36 %make x(t) vs. time plot
37 figure();
38 plot(t,x*1000); grid;
39 title(sprintf(...
40 'x vs. time. Maximum is [%3.3f] mm at time [%3.3f] sec',...
41 value*1000,t(I)));
42 xlabel('time (sec)'); ylabel('x in mm');
43
44 theta = sol(:,3);
45 [value,I] = max(theta);
46
47 %make theta(t) vs. time plot
48 figure();
49 plot(t,theta*1000); grid;
50 title(sprintf(...
51 'theta vs. time. Maximum is [%3.3f] mrad at time [%3.3f] sec',...
52 value*1000,t(I)));
53 xlabel('time (sec)'); ylabel('theta in mrad');
54

```

```
55 %internal function, for RHS. Since this is internal, it will
56 %see all the physical parameters defined in its parent function,
57 %hence no need to pass them or make them global
58
59 function v = rhs(~,x)
60     f1 = x(2);
61     f2 = (-k11*(x(1)-L1*x(3))- k12*(x(1)-L1*x(3))^3 - ...
62           k21*(x(1)+L2*x(3)) - k22*(x(1)+L2*x(3))^3)/m;
63     f3 = x(4);
64     f4 =(k11*(x(1)-L1*x(3))*L1 + k12*(x(1)-L1*x(3))^3*L1 - ...
65           k21*(x(1)+L2*x(3))*L2 - k22*(x(1)+L2*x(3))^3*L2)/J0;
66
67     v = [f1;f2;f3;f4];
68 end
69 end
```