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 . . . 4
0.1.4 Results . . . 4
0.1.5 Discussion of results . . . 5
0.1.6 Source code listing . . . 5
0.2 Problem 2 . . . 8
0.2.1 Results . . . 9
0.2.2 Source code listing . . . 11
0.3 Problem 3 . . . 12
0.3.1 Results . . . 14
0.3.2 Source code listing . . . 14

0.1 Problem 1

PROBLEM DESCRIPTION

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

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

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

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

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

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

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) $O\left(h^2\right)$ and an overall global error $O(h)$.

0.1.2 Modified Euler method

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

Modified Euler method has local error (per step) $O\left(h^3\right)$ and overall global error $O\left(h^2\right)$, 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 $O\left(h^3\right)$ and global error $O\left(h^2\right)$

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

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

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

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

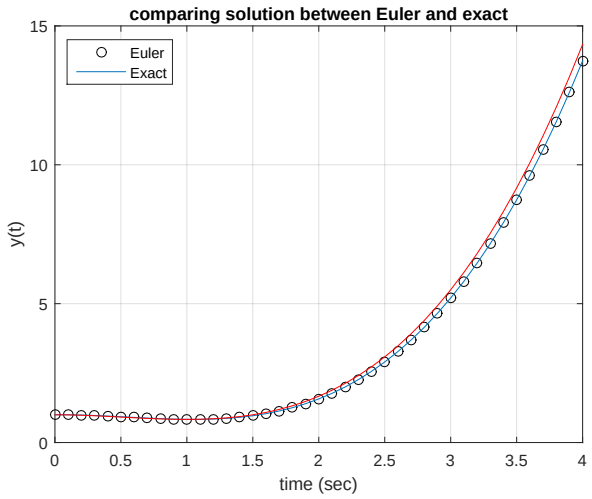
$$\begin{aligned} k_1 &= hf\left(nh\right) \\ k_2 &= hf\left(nh + \frac{3}{2}h\right) = hf\left(\left(n + \frac{3}{2}\right)h\right) \\ y_{n+1} &= y_n + \frac{2}{3}hf\left(nh\right) + \frac{1}{3}hf\left(\left(n + \frac{3}{2}\right)h\right) \\ &= y_n + \frac{2}{3}h\left[f\left(nh\right) + \frac{1}{2}f\left(\left(n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

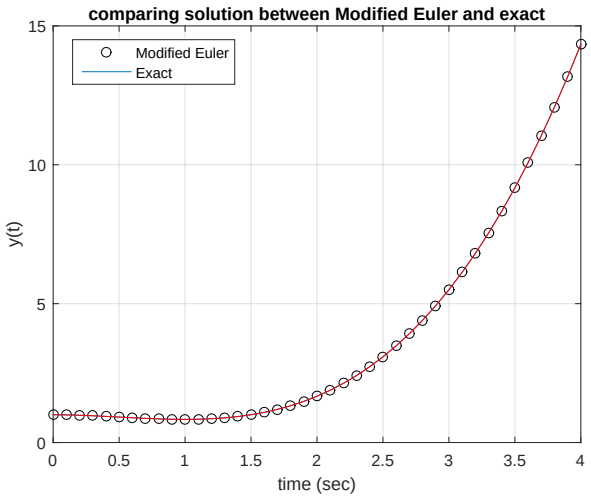
The following is a summary table of the three methods.

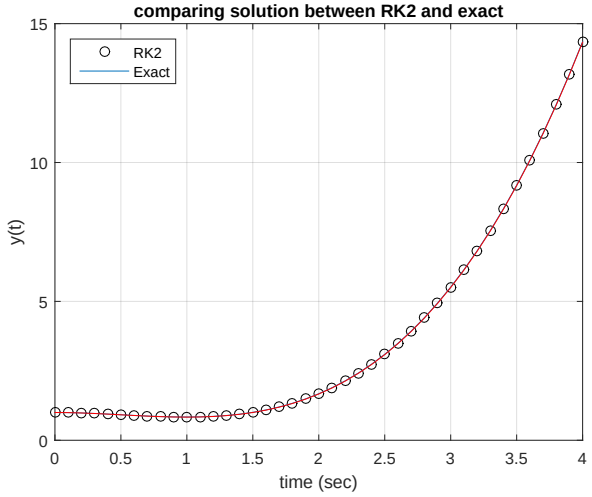
scheme name	main computation	global error
Euler	$y_{n+1} = y_n + hy'_n$	$O(h)$
Modified Euler	$y_{n+1} = y_n + \frac{h}{2}\left(y'_n + y'_{n+1}\right)$	$O(h^2)$
RK2	$y_{n+1} = y_n + \frac{2}{3}h\left(y'_n + \frac{1}{2}y'_{n+\frac{3}{2}}\right)$	$O(h^2)$

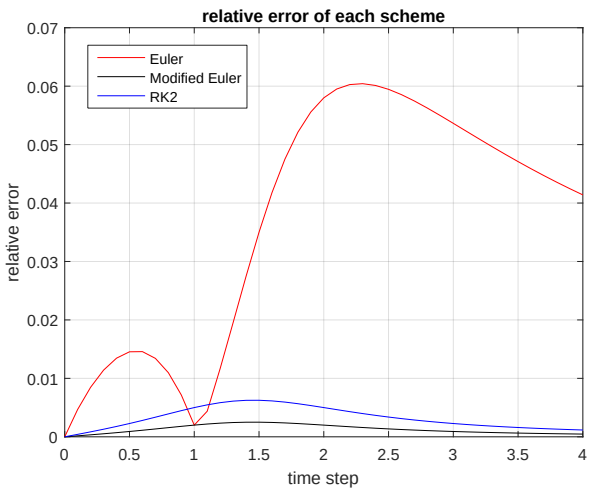
0.1.4 Results

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 $\frac{\text{|exact solution-numerical|}}{\text{|exact|}}$ 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

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

clear; close all;

%First call is to initialize data before each scheme run is made
[h,t,yExact,y] = initializeData();
yEuler         = Euler(h,t,y,yExact);

[h,t,yExact,y] = initializeData();
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
```

```
%--------------------------------------
%Implements RK2 solver
function y  = RK2(h,t,y,yExact)
currentTime = 0;
a           = 2/3;
b           = 1/3;
alpha       = 3/2;   %RK2 parameters used
%beta       = 3/2;   %not needed, since RHS f(.) do not depend on y.

%y(1) has been initialized to correct value by caller
for n = 2:length(y)
    k1          = h*rhs(currentTime);
    k2          = h*rhs(currentTime+alpha*h);
    y(n)        = y(n-1)+a*k1+b*k2;
    currentTime = currentTime + h;
end

makePlots(t,y,yExact,'RK2');
end

%---------------------------------
%Called to initialize data and counters before each solver is called
function [h,t,yExact,y] = initializeData()

h      = 0.1;      %time step we are asked to use
t      = (0:h:4)';
y      = zeros(length(t),1);
y(1)   = 1;        %initial conditions, from problem statement
yExact = 1+t.^3/3-t.^2/2;
end

%-------------------------
%Called to generate plots from each solver
function makePlots(t,y,yExact,schemeName)

figure();
plot(t,y,'ok',t,y); hold on;
plot(t,yExact,'r');
legend(schemeName,'Exact','Location','NorthWest');
title(sprintf('comparing solution between %s and exact',schemeName));
xlabel('time (sec)'); ylabel('y(t)');
grid;

end

%---------------
%RHS function, called by each scheme solver. Notice only t appears in RHS
function v = rhs(t)
  v = t^2-t;
end
```

0.2 Problem 2

PROBLEM DESCRIPTION

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

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

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 $0 \le t \le 0.5$. Then use `ode45` to solve the system over the interval $0 \le t \le 2$. Plot your numerical solutions on the same plot as the Taylor series expansions and discuss your results.

SOLUTION

$$x'(t) = xy + t$$
$$y'(t) = ty + x$$

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

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

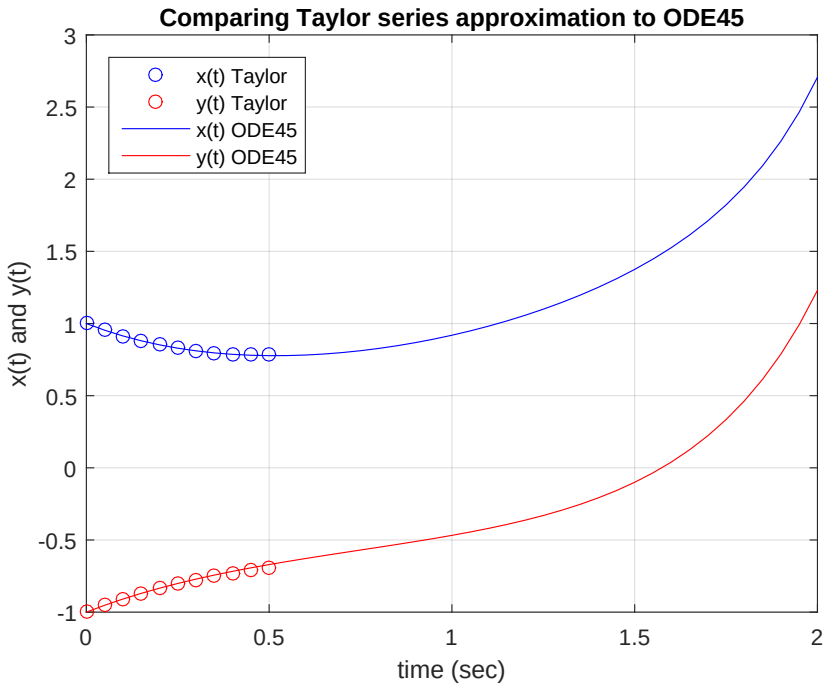
$$x(t) = x(0) + tx'(0) + \frac{t^2}{2}x''(0) + \frac{t^3}{3!}x'''(0) + \frac{t^4}{4!}x^{(4)} + \frac{t^5}{5!}x^{(5)} + O\left(t^6\right)$$
$$y(t) = y(0) + ty'(0) + \frac{t^2}{2}y''(0) + \frac{t^3}{3!}y'''(0) + \frac{t^4}{4!}y^{(4)} + \frac{t^5}{5!}y^{(5)} + O\left(t^6\right)$$

Applying values from the table above gives

$$\begin{aligned} x(t) &= 1 - t + \frac{3}{2}t^2 - 7\frac{t^3}{3!} + 23\frac{t^4}{4!} - 22\frac{t^5}{5!} + O\left(t^6\right) \\ &= 1 - t + \frac{3}{2}t^2 - \frac{7}{6}t^3 + \frac{23}{24}t^4 - \frac{22}{120}t^5 + O\left(t^6\right) \\ y(t) &= -1 + t - t^2 + 4\frac{t^3}{3!} - 13\frac{t^4}{4!} + 39\frac{t^5}{5!} + O\left(t^6\right) \\ &= -1 + t - t^2 + \frac{2}{3}t^3 - \frac{13}{24}t^4 + \frac{39}{120}t^5 + O\left(t^6\right) \end{aligned}$$

0.2.1 Results

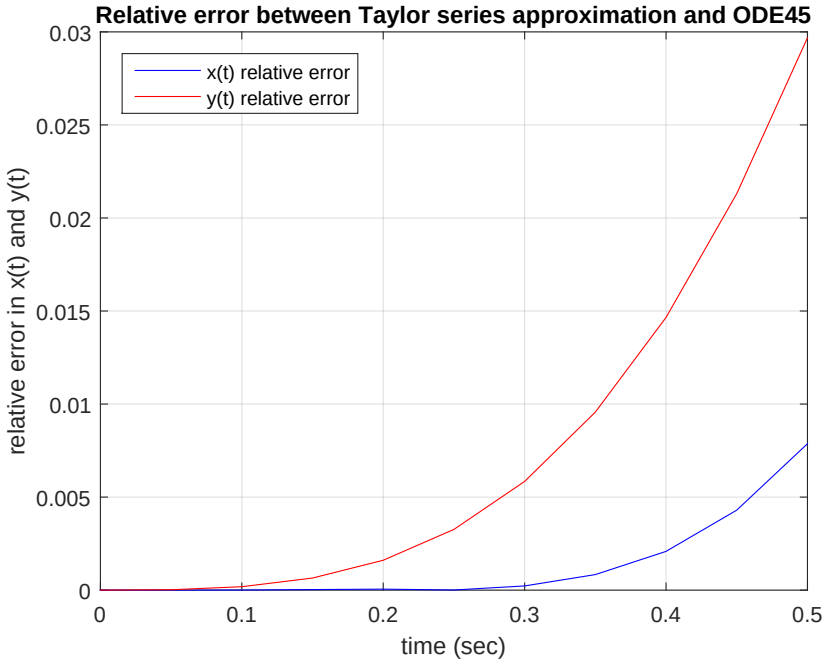
The following diagram shows the solution using Taylor series approximation and using ODE45



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

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



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 $y(t)$.

0.2.2 Source code listing

```
function   nma_EMA_471_HW1_part_b
%This function solves HW1, problem 2. EMA 471, spring 2016,
%UW Madison see report for detailed information about the problem
%by Nasser M. Abbasi

close all;

h  = 0.05;    %step size
t1 = 0:h:0.5; %time for the Taylor series
t2 = 0:h:2;   %for ODE45 we are asked to go for 2 seconds.

[xApprox,yApprox] = doTaylorApproximation(t1);
[xODE45,yODE45]   = doODE45(t2);

%done, now plot and compare.
compare(t1,xApprox,yApprox,t2,xODE45,yODE45);
end

%---------------------------------
%Do the Taylor series approximation
function [xApprox,yApprox] = doTaylorApproximation(t)
xApproxF = @(t) 1 - t + (3/2)* t.^2 - 7/6 * t.^3  + 23/24 *  ...
                                          t.^4 -22/120 * t.^5;
yApproxF = @(t)-1 + t - t.^2 + 2/3 * t.^3  - 13/24 *...
                                          t.^4 +39/120 * t.^5;

xApprox  = xApproxF(t);
yApprox  = yApproxF(t);
end

%-------------------------------------------------
%Do ODE45. This uses Matlab's ode45 to solve the problem.
function [xODE45,yODE45] = doODE45(t)
xInitial = 1;
yInitial = -1;

[~,v]  = ode45( @rhs, t, [xInitial yInitial] );
%extract solutions
xODE45 = v(:,1);
yODE45 = v(:,2);

%internal function, for RHS
    function v = rhs(t,X)
        x = X(1);
        y = X(2);
        v = [x*y+t;t*y+x];
    end
end

```

```
%-------------------------------------------------------
%function to compare Taylor series with ODE 45 output
function compare(t1,xApprox,yApprox,t2,xODE45,yODE45)

figure();
plot(t1,xApprox,'bo',t1,yApprox,'ro'); hold on;
plot(t2,xODE45,'b',t2,yODE45,'r');

legend('x(t) Taylor','y(t) Taylor','x(t) ODE45','y(t) ODE45',...
        'Location','NorthWest');

title('Comparing Taylor series approximation to ODE45');
xlabel('time (sec)'); ylabel('x(t) and y(t)');
grid;

figure();
plot(t1,abs(xODE45(1:length(t1))-xApprox')./...
                                abs(xODE45(1:length(t1))),'b',...
     t1,abs(yODE45(1:length(t1))-yApprox')./...
                                  abs(yODE45(1:length(t1))),'r');
title('Relative error between Taylor series approximation and ODE45');
xlabel('time (sec)');
ylabel('relative error in x(t) and y(t)');
legend('x(t) relative error','y(t) relative error',...
        'Location','NorthWest');
grid;
end
```

0.3 Problem 3

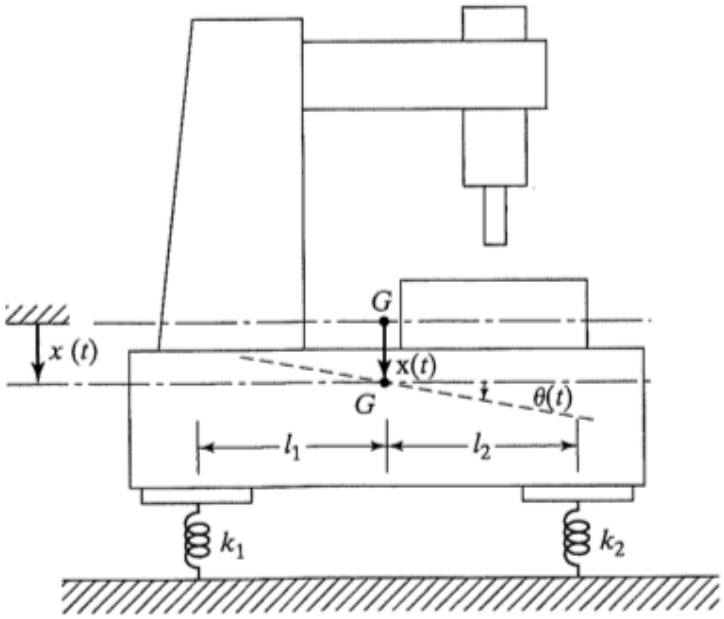
PROBLEM DESCRIPTION

(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:

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

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 $0 \le t \le 10$ 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 kg, J_o = 2500 kg-m^2, l_1 = 1 m, l_2 = 1.5 m, k_{11} = 40 kN/m, k_{12} = 10 kN/m^3, k_{21} = 50 kN/m, k_{22} = 5 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

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

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

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

$$x_1 = x$$
$$x_2 = x'$$
$$x_3 = \theta$$
$$x_4 = \theta'$$

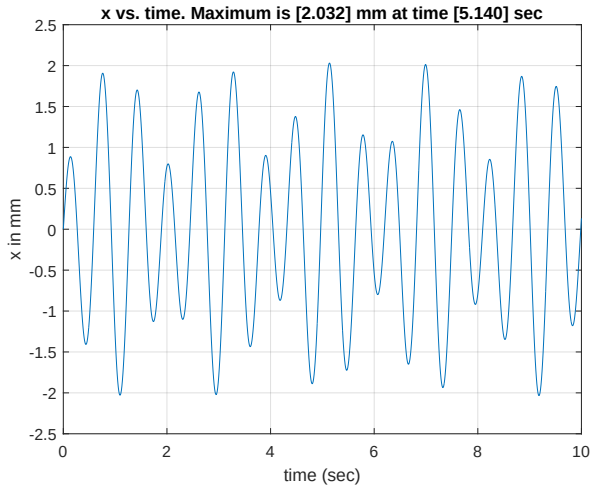
After taking time derivatives, the above becomes

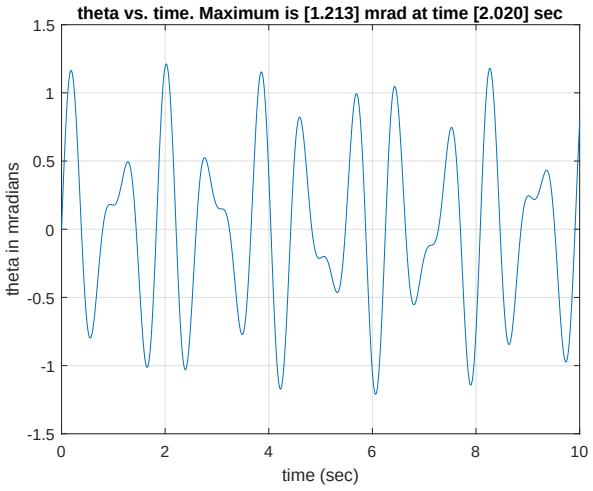
$$\dot{x}_1 = x' = x_2$$
$$\dot{x}_2 = x'' = -\frac{1}{m}\left[k_{11}\left(x_1 - l_1 x_3\right) + k_{12}\left(x_1 - l_1 x_3\right)^3 + k_{21}\left(x_1 + l_2 x_3\right) + k_{22}\left(x_1 + l_2 x_3\right)^3\right]$$
$$\dot{x}_3 = \theta' = x_4$$
$$\dot{x}_4 = \theta'' = -\frac{1}{J_0}\left[-k_{11}\left(x_1 - l_1 x_3\right) l_1 - k_{12}\left(x_1 - l_1 x_3\right)^3 l_1 + k_{21}\left(x_1 + l_2 x_3\right) l_2 + k_{22}\left(x_1 + l_2 x_3\right)^3 l_2\right]$$

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 $\boxed{x_{\max} = 2.032 \text{ mm}}$ and occurred at $t = 5.14$ seconds. For the angle, the maximum angular displacement was $\boxed{\theta_{\max} = 1.213 \text{ mrad}}$ (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

```
function   nma_EMA_471_HW1_part_c
%This function solves HW1, part c. EMA 471, spring 2016,
%UW Madison report contains detailed information about the problem
%by Nasser M. Abbasi

close all;

%physical paramters of problem
m    = 1000;    %kg
J0   = 2500;    %kg-m^3
L1   = 1;       %meter
L2   = 1.5;     %meter
k11  = 40*10^3; %N/m
k12  = 10*10^3; %N/m^3
```

```
k21  = 50*10^3; %N/m
k22  = 5*10^3;  %N/m

%initialization parameters and initial conditions
h             = 0.01;       %step size. ODE45 did not work
                            %well with large step
t             = 0:h:10;     %time for ODE45
xInitial      = 0;          %meter
xVInitial     = 10*10^(-3); %10 mm/sec
thetaInitial  = 0;          %radians
thetaVInitial = 10*10^(-3); %mrad/sec

%call ODE45 to numerically solve the equations of motions
[t,sol] = ode45( @rhs, t, ...
              [xInitial xVInitial thetaInitial thetaVInitial]);

%Extract the first and third columns. These represet x
%and theta solutions
x          = sol(:,1);
[value,I]  = max(x);

%make x(t) vs. time plot
figure();
plot(t,x*1000); grid;
title(sprintf(...
      'x vs. time. Maximum is [%3.3f] mm at time [%3.3f] sec',...
      value*1000,t(I)));
xlabel('time (sec)'); ylabel('x in mm');

theta     = sol(:,3);
[value,I] = max(theta);

%make theta(t) vs. time plot
figure();
plot(t,theta*1000); grid;
title(sprintf(...
 'theta vs. time. Maximum is [%3.3f] mrad at time [%3.3f] sec',...
 value*1000,t(I)));
xlabel('time (sec)'); ylabel('theta in mradians');

%internal function, for RHS. Since this is internal, it will
%see all the  physical parameters defined in its parent function,
%hence no need to pass them or make them global

    function v = rhs(~,x)
        f1 = x(2);
        f2 = (-k11*(x(1)-L1*x(3))- k12*(x(1)-L1*x(3))^3 - ...
                    k21*(x(1)+L2*x(3)) - k22*(x(1)+L2*x(3))^3)/m;
        f3 = x(4);
        f4 =(k11*(x(1)-L1*x(3))*L1 + k12*(x(1)-L1*x(3))^3*L1 - ...
             k21*(x(1)+L2*x(3))*L2 - k22*(x(1)+L2*x(3))^3*L2)/J0;

        v = [f1;f2;f3;f4];
    end
```

```
69 end
```