Dynamics of Mechanical Systems
Academic Year 2023/2024
Homework 4

Afonso Henrique, 98555 - afonso.henrique@tecnico.ulisboa.pt

1 Objectives

Implement and run the Matlab Dynamic Analysis code developed in class, and later explored in the Lab classes, with the spring-mass system associated to your student number. Show, in a plot, the trajectory of the system masses as a function of time for 2 complete oscillations of the slower mass. Deliver the printed Matlab code and the dynamic response requested. The data supplied applies to all systems (even if for some of then it is not realistic). (Note: You must fix the code developed in class as there is no guarantee that it is correct) For extra points: Compare the accuracy and the CPU time of solving the problem with at least 3 different time integrators of your choice. Draw a reasonable conclusion

2 Spring-Mass System

For my student number, the spring-mass system to be used was number 3:

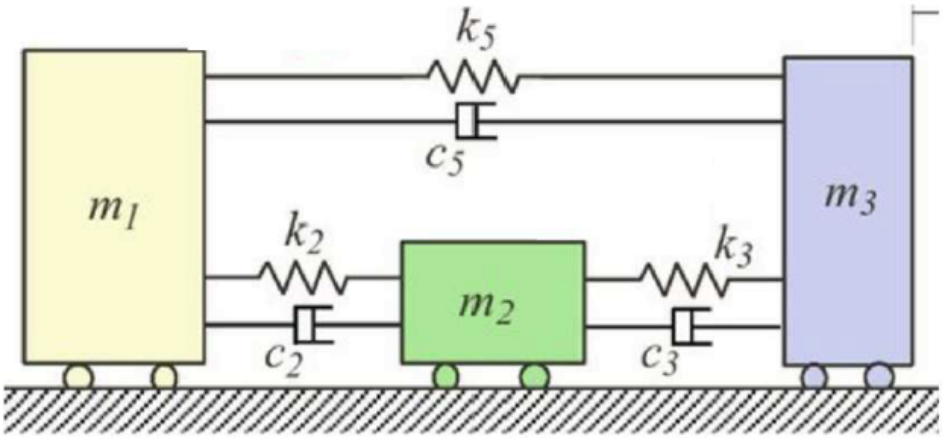

Figure 1: Spring-Mass system number 3.

Properties of the system:

- $m_1 = 15kg$
- $m_2 = 25kg$
- $m_3 = 5kg$
- $k_2 = 25N/m$
- $l_2^0 = 0.5m$
- $c_2 = 2N/m \cdot s$
- $k_3 = 16N/m$
- $l_3^0 = 0.1m$
- $c_3 = 2N/m \cdot s$
- $k_5 = 30N/m$
- $l_5^0 = 0.2m$
- $c_5 = 1N/m \cdot s$

3 Equations of Motions & Referential

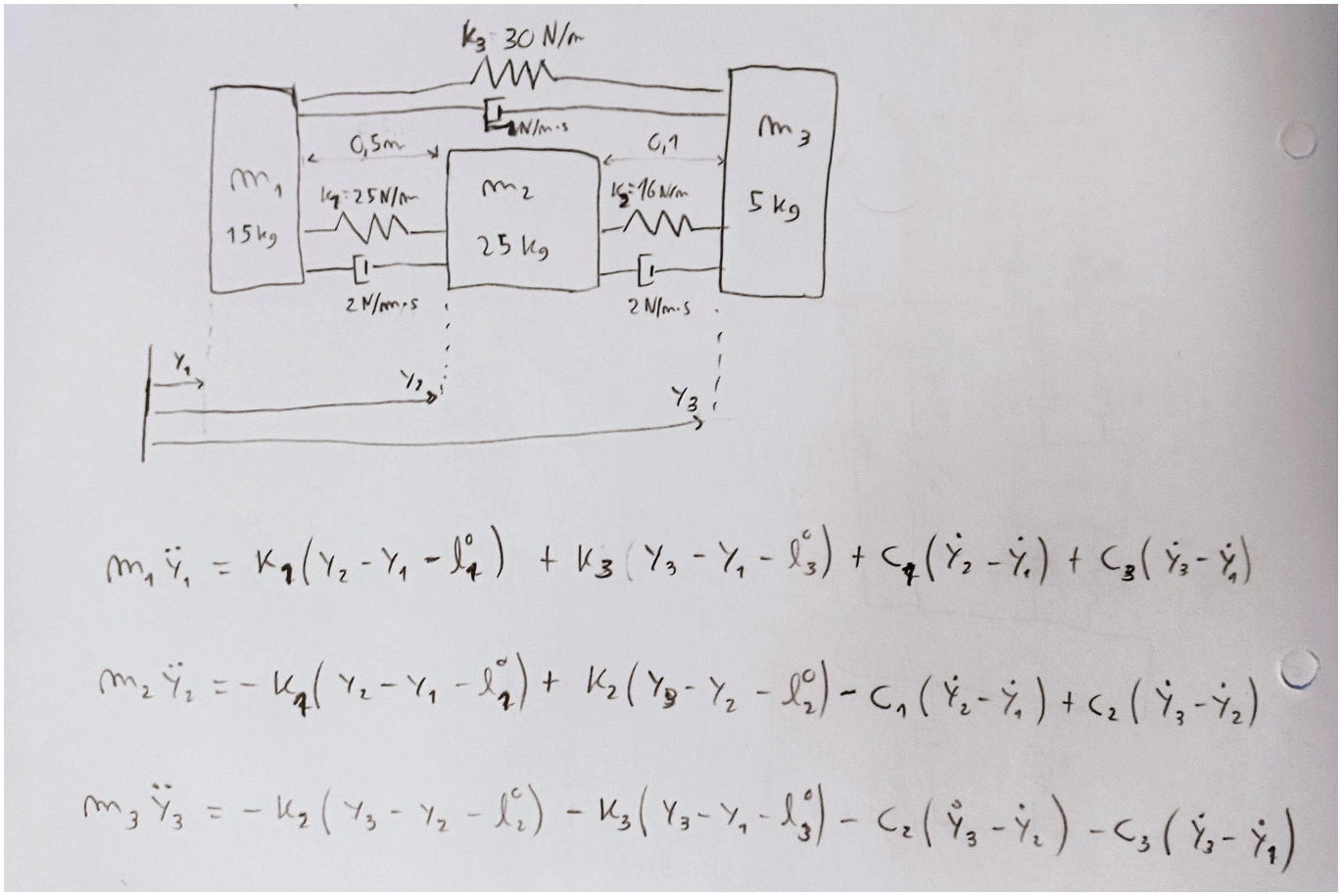

Figure 2: Equations of Motions & Referential.

4 Initial Conditions

The initial conditions considered for the solutions obtained in the following sections were:

- $\dot{y_1} = \dot{y_2} = \dot{y_3} = 0m/s$
- $y_1 = 0m$
- $y_2 = 0.5m$
- $y_3 = 0.1m$

5 Position & Velocities over time

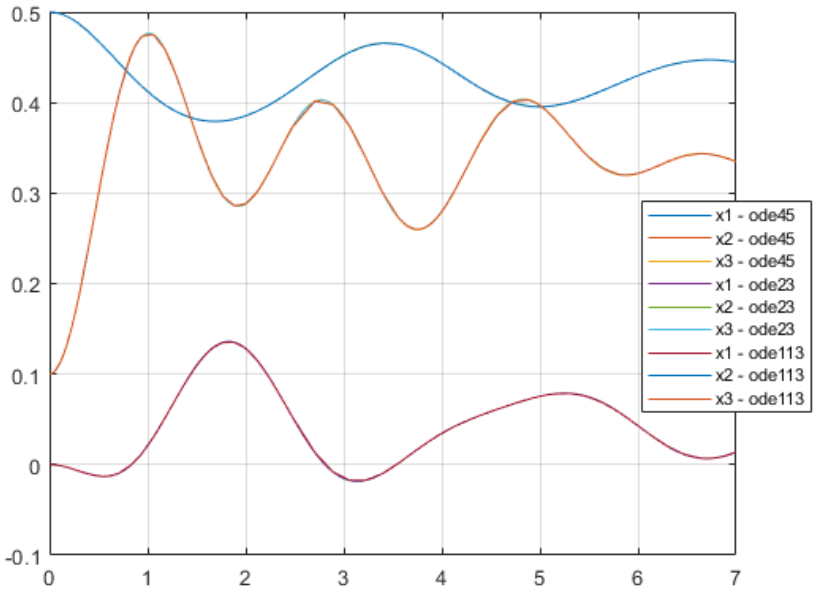

Figure 3: Position of each mass over time.

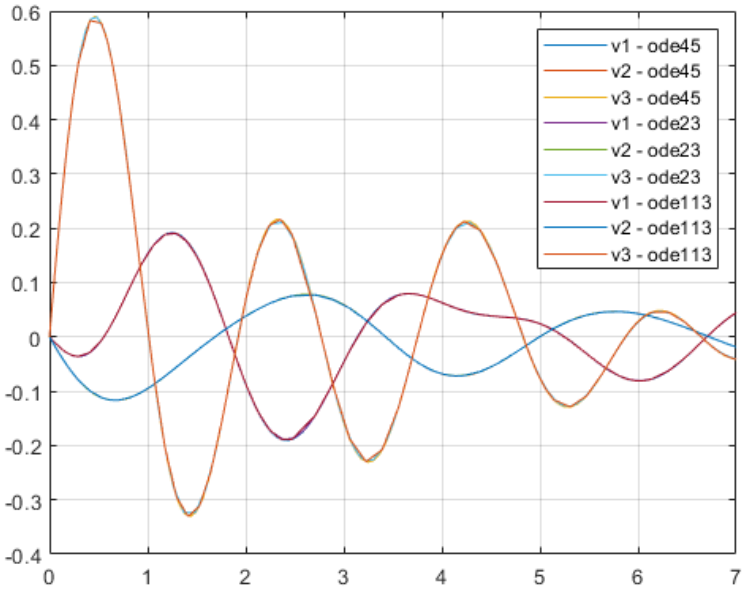

Figure 4: Velocity of each mass over time.

6 Comparing Numerical Integration Methods

The MATLAB function *ode23* is a bit faster when compared to the two others, but error is quite large, specially for the fastest mass, where it's trajectory no longer quite sinusoidal, as we know it should. The user should set the time step and choose and ODE solver which strikes the desired balance between computation time and accuracy.

MATLAB ODE Solver	CPU time (ms)
ode45	62.5
ode113	46.9
ode23	31.2

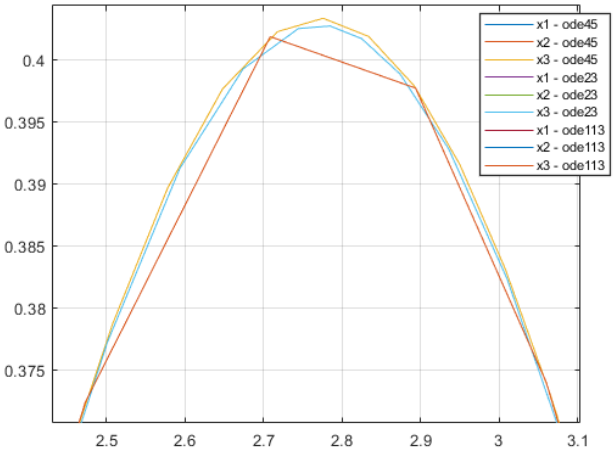

Figure 5: Position of mass number 3 (the fastest mass) obtained with different integration methods.

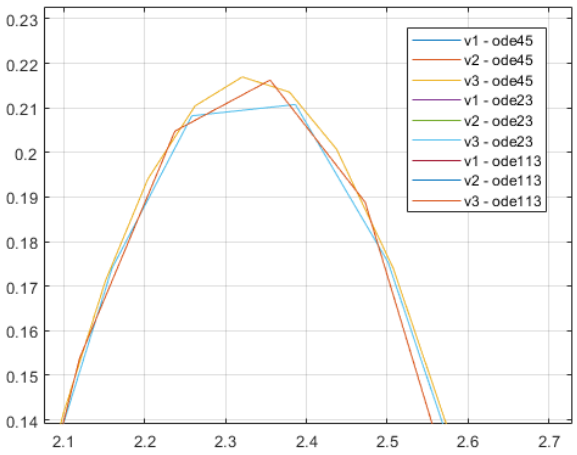

Figure 6: Velocity of mass number 3 (the fastest mass) obtained with different integration methods.

```
%CreateModel.m
clear all
%
SpringDamper(1).K=25;
SpringDamper(1).C=2;
SpringDamper(1).L0=0.5;
%
SpringDamper(2).K=16;
SpringDamper(2).C=2;
SpringDamper(2).L0=0.1;
%
SpringDamper(3).K=30;
SpringDamper(3).C=1;
SpringDamper(3).L0=0.2;
%
%
Body(1).mass=15;
Body(1). x=0.0;
Body(1).xd=0;
%
Body(2).mass=25;
Body(2).x=0.5;
Body(2).xd=0;
%
Body(3).mass=5;
Body(3).x=0.1;
Body(3).xd=0;
%
tstart=0; tstep=0.01; tend=7;
solver='ode45';
FuncEval='FirstDynamicSystem';
%
save('FirstDynamicModel.mat');
%
% End of the script


%%%%%%%%%%%%%%%%%


% PreProcessor.m
function [tspan, y_init]=PreProcessor();
%
% access global memory
global M SpringDamper Body tstart tstep tend solver FuncEval

%Create Mass matrix
M=[Body(1).mass,     0,                              0;
        0,                        Body(2).mass,        0;
        0,                        0,                              Body(3).mass];

% create initial y vector
y_init=[Body(1).x Body(1).xd Body(2).x Body(2).xd Body(3).x Body(3).xd]';

% time analysis profile
tspan=tstart:tstep:tend;
end


% End of the script


%%%%%%%%%%%%%%%%%
```

```
% FirstDynamicSystem.m
function [yd]= FirstDynamicSystem(t,y)
% access global data memory
global M SpringDamper Body tstart tstep tend solver FuncEval

% update local data
Body(1).x=y(1,1);
Body(1).xd=y(2,1);
Body(2).x=y(3,1);
Body(2).xd=y(4,1);
Body(3).x=y(5,1);
Body(3).xd=y(6,1);
%
k1= SpringDamper(1).K;
c1= SpringDamper(1).C;
L10=SpringDamper(1).L0;
x1= Body(1).x;
xd1= Body(1).xd;
%
k2= SpringDamper(2).K;
c2= SpringDamper(2).C;
L20=SpringDamper(2).L0;
x2= Body(2).x;
xd2= Body(2).xd;
%
k3= SpringDamper(3).K;
c3= SpringDamper(3).C;
L30=SpringDamper(3).L0;
x3= Body(3).x;
xd3= Body(3).xd;
% create the force vector from local data
g=[( k1*(x2-x1-L10)  + k3*(x3-x1-L30) + c1*( xd2-xd1)+ c3*(xd3-xd1) );
( -k1*(x2-x1-L10)  + k2*(x3-x2-L20) - c1*( xd2-xd1) + c2*(xd3-xd2) );
( -k2*(x3-x2-L20)  - k3*(x3-x1-L30) - c2*( xd3-xd2) - c3*(xd3-xd1) )];
%
% Evaluate System accelerations
qdd=M\g;
% form the time derivative of y
yd=[Body(1).xd; qdd(1); Body(2).xd; qdd(2);Body(3).xd; qdd(3)];

% End of the script


%%%%%%%%%%%%%%%%


%Solve_Dynamic.m
clear all
close all
clc
%
global SpringDamper Body tstart tstep tend solver FuncEval
%
disp('Need to run first the model to create mat file');
% select the workspace file with model data
[filename, Path]=uigetfile('*.mat','SelectModel Data');
% load model data
load([Path,filename]);
% Pre-process the input data
[tspan,y_init]=PreProcessor();
% integration of the equations of motion

CPUStart = cputime;
[t,y]=ode45(@FirstDynamicSystem,[tstart,tend],y_init);
CPUTime_ode45 = cputime-CPUStart
```

```
%Animation

Nframes=length(y);
colmap = ['b' , 'r' , 'y' , 'k'];

for i=1:Nframes
    for j=1:3
        ybody = y( i , 2*j-1);

        P1=[-0.04 , ybody-0.02];
        P2=[0.04 , ybody-0.02];
        P3=[0.04 , ybody+0.02];
        P4=[-0.04 , ybody+0.02];

        fill([P1(1) P2(1) P3(1) P4(1)],[P1(2) P2(2) P3(2) P4(2)] , colmap(j));
        text(0,ybody,num2str( j));
        hold on;
    end

    axis([-0.35 0.35 -0.1 0.6]);
    pbaspect([1 1 1]);
    hold off;
    pause(0.1);
end
close all

% post-process the results
x1_position=y(:,1);
x1_velocity=y(:,2);
x2_position=y(:,3);
x2_velocity=y(:,4);
x3_position=y(:,5);
x3_velocity=y(:,6);
%
figure(1);
xlabel('t'); ylabel('x_i(t)');
plot(t,x1_position,t,x2_position,t,x3_position);
%legend('x1 - ode45','x2 - ode45','x3 - ode45');
hold on;
grid on

figure(2);
xlabel('t'); ylabel('v_i(t)');
plot(t,x1_velocity,t,x2_velocity,t,x3_velocity);
%legend('v1 - ode45','v2 - ode45','v3 - ode45');
hold on;
grid on


CPUStart = cputime;
[t,y]=ode23(@FirstDynamicSystem,[tstart,tend],y_init);
CPUTime_ode23 = cputime-CPUStart

% post-process the results
x1_position=y(:,1);
x1_velocity=y(:,2);
x2_position=y(:,3);
x2_velocity=y(:,4);
x3_position=y(:,5);
x3_velocity=y(:,6);


%
figure(1);
hold on;
plot(t,x1_position,t,x2_position,t,x3_position);
```

```
%legend('x1 - ode23','x2 - ode23','x3 - ode23');
hold on;
grid on

figure(2);
hold on;
plot(t,x1_velocity,t,x2_velocity,t,x3_velocity);
%legend('v1 - ode23','v2 - ode23','v3 - ode23');
hold on;
grid on


CPUStart = cputime;
[t,y]=ode113(@FirstDynamicSystem,[tstart,tend],y_init);
CPUTime_ode113 = cputime-CPUStart


% post-process the results
x1_position=y(:,1);
x1_velocity=y(:,2);
x2_position=y(:,3);
x2_velocity=y(:,4);
x3_position=y(:,5);
x3_velocity=y(:,6);
%
figure(1);
hold on;
plot(t,x1_position,t,x2_position,t,x3_position);
legend('x1 - ode45','x2 - ode45','x3 - ode45','x1 - ode23','x2 - ode23','x3 - ode23','x1 -
    ode113','x2 - ode113','x3 - ode113');
hold on;
grid on

figure(2);
hold on;
plot(t,x1_velocity,t,x2_velocity,t,x3_velocity);
legend('v1 - ode45','v2 - ode45','v3 - ode45','v1 - ode23','v2 - ode23','v3 - ode23','v1 -
    ode113','v2 - ode113','v3 - ode113');
hold on;
grid on

% End of the script
```