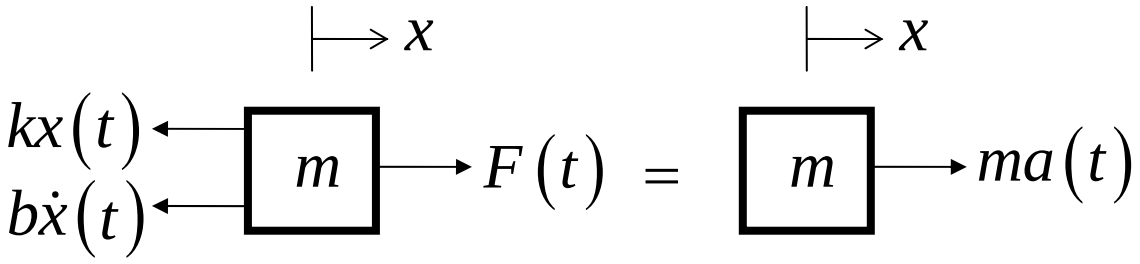
Massachusetts Institute of Technology
Department of Mechanical Engineering

2.003J/1.053J Dynamics & Control I

Fall 2007

Homework 6 Solution

Problem 6.1 : Dynamics of mass-spring-damper system



From the above diagram,

$$F(t) - kx(t) - b\dot{x}(t) = ma(t) \tag{1}$$

$$F(t) - kx(t) - b\frac{dx(t)}{dt} = m\frac{d^2x(t)}{dt^2}$$

$$\rightarrow \quad m\frac{d^2x(t)}{dt^2} + b\frac{dx(t)}{dt} + kx(t) = F(t), \quad x(t_i) = x_0 \quad \& \quad \left.\frac{dx(t)}{dt}\right|_{t=t_i} = v_0 \tag{2}$$

Problem 6.2 : Solver for mass-spring-damper system with Euler method

First, 2nd order differential equation is split into two 1st order differential equations as did in Homework #3.

$$x_1 = x \quad \& \quad x_2 = \dot{x} = \frac{dx}{dt}$$

$$\frac{dx_1}{dt} = \dot{x} = x_2 \tag{3}$$

$$\frac{dx_2}{dt} = \ddot{x} = \frac{1}{m}\left(F_0 \sin\omega t - b\dot{x} - kx\right) = \frac{1}{m}\left(F_0 \sin\omega t - bx_2 - kx_1\right)$$

With Eq.(3), we also obtain the discrete version of above differential equations.

$$
\begin{aligned}
& x(n,1) \leftarrow x_1(n\Delta t) \quad \& \quad x(n,2) \leftarrow x_2(n\Delta t) \\
& x(n+1,1) = x(n,1) + x(n,2) \times \Delta t \\
& x(n+1,2) = x(n,2) + a \times \Delta t \\
& \textit{where} \quad a = \frac{1}{m}\left(F_0 \sin\left(\omega \times (n\Delta t)\right) - b \times x(n,2) - k \times x(n,1)\right)
\end{aligned}
\tag{4}
$$

The solver of mass-spring-damper system with Euler method is implemented as below. Explanation of each command line is included in the following codes.

```
function O=MSDSE(m,b,k,F0,w,x0,v0)
%
% Solver for Mass-Sprring-Damper System with Euler Method
% ----- Input argument -----
% m: mass for particle
% b: damping coefficient
% k: spring constant
% F0: amplitude of external force
% w: angular freuency of external force
% x0: initial condition for the position x(0)
% v0: initial condition for the velocity v(0)
% ----- Output argument -----
% t: time series with given time step from ti to tf.
% x: state variable matrix for corresponding time t matrix

% Define time step in Euler method
dt=0.1;
% Make time series with given time step
t=[0:dt:50]';
% Initialize output matrix
% the 1st column: the position of the particle
% the 2nd column: the veleocity of the particle
x=zeros(length(t),2);
% the 1st row has initial conditions
x(1,:)=[x0 v0];
% Start the simulation with Euler method
for i=1:length(t)-1
```

```
    % Apply Euler method
    % x(n+1)=x(n)*v(n)*dt
    x(i+1,1,1)=x(i,1)+x(i,2)*dt;
    % v(n+1)=v(n)*a(n)*dt
    %where a(n)=1/m*(F0sin(wt)-bv(n)-kx(n))*dt
    x(i+1,2)=x(i,2)+(1/m)*(F0*sin(w*t(i))-b*x(i,2)-k*x(i,1))*dt;
end
% Extract only particle position trajectory
O=[t,x(:,1)];
end
```

Problem 6.3 : Solver for mass-spring-damper system with Runge-Kutta method

Unlike Euler method, you don't need to solve differential equation itself in MATLAB. (You can also make your own code for Runge-Kutta algorithm for yourself.) However, the subfunction should be needed to implement Runge-Kutta algorithm. (Single m-file can have several functions. The first one is the primary function, and others are subfunction.) Subfunction includes several 1st order differential equations by splitting higher order differential equation. In our case, two 1st order equations are used as described in problem 6.2. With this subfunction, the solver (either `'ode23'` or `'ode45'`) solves differential equations actually. The handler of function which has dynamic equation description, time span which the solver calculates between, and initial condition at the simulation starting time are at least specified as the input arguments when the solver runs. More detailed syntax description of either `'ode23'` or `'ode45'` is shown in MATLAB help. The solver of mass-spring-damper system with Runge-Kutta method is implemented as below. Explanation of each command line is included in the following codes.

```
function O=MSDSRK(m,b,k,F0,w,x0,v0)
%
% Solver for Mass-Sprring-Damper System with Runge-Kutta Method
% ----- Input argument -----
% m: mass for particle
% b: damping coefficient
% k: spring constant
% F0: amplitude of external force
% w: angular freuency of external force
```

```
% x0: initial condition for the position x(0)
% v0: initial condition for the velocity v(0)
% ----- Output argument -----
% t: time series with given time step from ti to tf.
% x: state variable matrix for corresponding time t matrix

% define time steps for solver and display
dt=0.1;
% set both initial time step size and maximum step size
% for Runge-Kutta solver
options=odeset('InitialStep',dt,'MaxStep',dt);
% set time span to be generated from Runge-Kutta solver
% from 0 sec to 50 sec with 0.1 sec time step
td=[0:dt:50];
% Solve differential equation with Runge-Kutta solver
[t,x]=ode45(@(t,X)MSD(t,X,m,b,k,F0,w),td,[x0;v0],options);
% Extract only particle position trajectory
O=[t x(:,1)];
end

function dX=MSD(t,X,m,b,k,F0,w)
%
% With two 1st order diffeential equations,
% obtain the derivative of each variables
% (position and velocity of the particle)
%
% t: current time
% X: matrix for state variables
%   The first column : the particle position
%   The second column : the particle velocity
% m,b,k,F0,w: parameters for the system
%

% Apply two 1st order differential equations
% dx(n+1)/dt<-v(n)
```

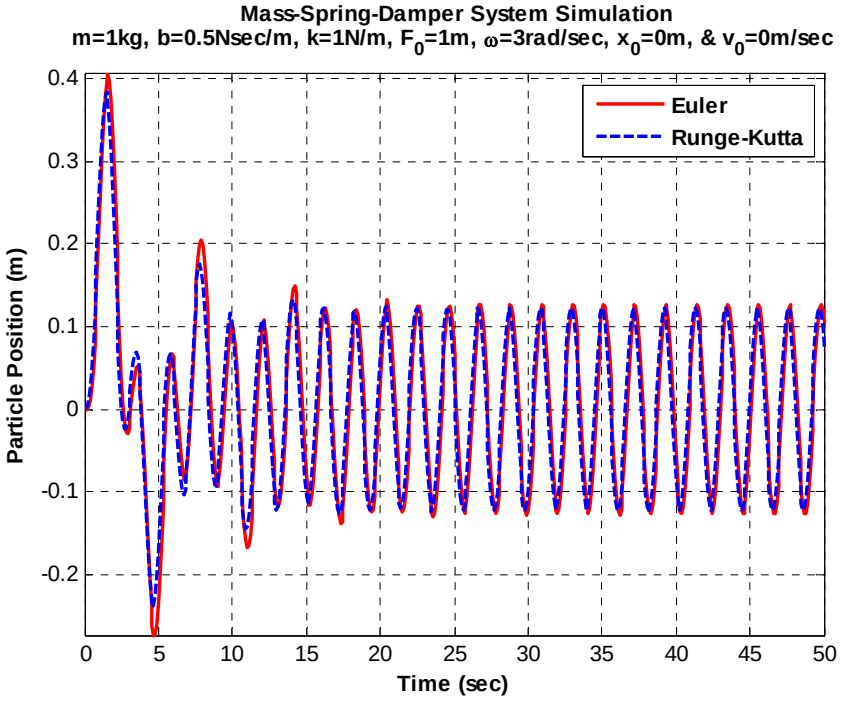
```
dX(1,1)=X(2,1);
% dv(n+1)/dt<-1/m*(F0sin(wt)-bv(n)-kx(n))
dX(2,1)=(1/m)*(F0*sin(w*t)-b*X(2,1)-k*X(1,1));
end
```

Problem 6.4 : Trajectory of mss-spring-damper system with different parameters and initial conditions

i) Following code is used to generate below plot.

```
>> Oe=MSDSE(1,0.5,1,1,3,0,0);
>> Or=MSDSRK(1,0.5,1,1,3,0,0);
>>  plot(Oe(:,1),Oe(:,2),'r',Or(:,1),Or(:,2),'b--','LineWidth',2);  axis
tight; grid on;
>> xlabel('\bfTime (sec)'); ylabel('\bfParticle Position (m)');
>>   title({'\bf   Mass-Spring-Damper   System   Simulation';   'm=1kg,
b=0.5Nsec/m, k=1N/m, F_0=1m, \omega=3rad/sec, x_0=0m, & v_0=0m/sec'})
>> legend('\bfEuler','\bfRunge-Kutta','Location','NorthEast')
```

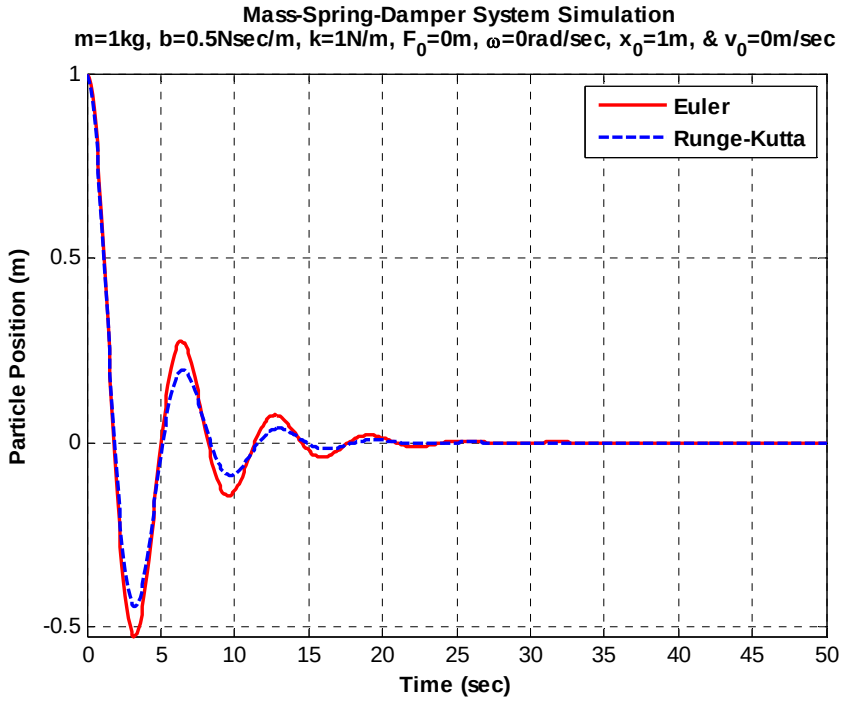
It's the response of the sinusoidal excitation for the particle when it is stationary, and no spring is compressed or stretched. Comparing two results from Euler method and Runge-Kutta method, they produce similar simulation results. The plot is shown as below:



ii) Following code is use to generate below plot.

```
>> Oe=MSDSE(1,0.5,1,0,0,1,0);
>> Or=MSDSRK(1,0.5,1,0,0,1,0);
>>  plot(Oe(:,1),Oe(:,2),'r',Or(:,1),Or(:,2),'b--','LineWidth',2);  axis
tight; grid on;
>> xlabel('\bfTime (sec)'); ylabel('\bfParticle Position (m)');
>>   title({'\bf   Mass-Spring-Damper   System   Simulation';   'm=1kg,
b=0.5Nsec/m, k=1N/m, F_0=0m, \omega=0rad/sec, x_0=1m, & v_0=0m/sec'})
>> legend('\bfEuler','\bfRunge-Kutta','Location','NorthEast')
```

It's the response with no external force and no damping when the spring is initially stretched. Comparing two results from Euler method and Runge-Kutta method, Runge-Kutta method is more accurate than Euler method, based on the analytical solution of this system. The plot is shown as below:



iii) Following code is use to generate below plot.

```
>> Oe=MSDSE(1,0,1,1,1,0,0);
>> Or=MSDSRK(1,0,1,1,1,0,0);
>>  plot(Oe(:,1),Oe(:,2),'r',Or(:,1),Or(:,2),'b--','LineWidth',2);  axis
tight; grid on;
```

```
>> xlabel('\bfTime (sec)'); ylabel('\bfParticle Position (m)');
>>   title({'\bf   Mass-Spring-Damper   System   Simulation';   'm=1kg,
b=0Nsec/m, k=1N/m, F_0=1m, \omega=1rad/sec, x_0=0m, & v_0=0m/sec'})
>> legend('\bfEuler','\bfRunge-Kutta','Location','NorthWest')
```

It's the response of external force with the system resonance frequency when the spring is initially stretched. Resonance happens very quickly, which means the particle position oscillates from $-\infty$ to $+\infty$. However, numerical simulation has limitation to express the infinity. Therefore, as time goes on, oscillations for both methods become larger, but Euler method one has smaller oscillation, compared with Runge-Kutta one. So, Euler method is not powerful near the singularity. The plot is shown as below: