Massachusetts Institute of Technology
Department of Mechanical Engineering

2.003J/1.053J Dynamics & Control I

Fall 2007

Homework 3 Solution

Problem 3.1 : Calculate the trajectory of a ball dropping and bouncing without no drag force

First, the function can be defined as below. No input argument and two output arguments (time 't' and trajectory 'h')

```
function [t,h]=ball3
%
% Problem 3.1 Trajectory of the ball dropping and bouncing
%           until the ball hits 3 times on the ground
%
```

Second, you should define some constants used in simulation: gravity, time step size, coefficients of restitution, and whatever you need.

```
% Define constants
g=9.81;     % gravity (m/sec^2)
dt=0.01;  % time step[temporal resolution] (sec)
H=1;       % initial height when ball start dropping
COR=0.8;    % coefficient of restitution
```

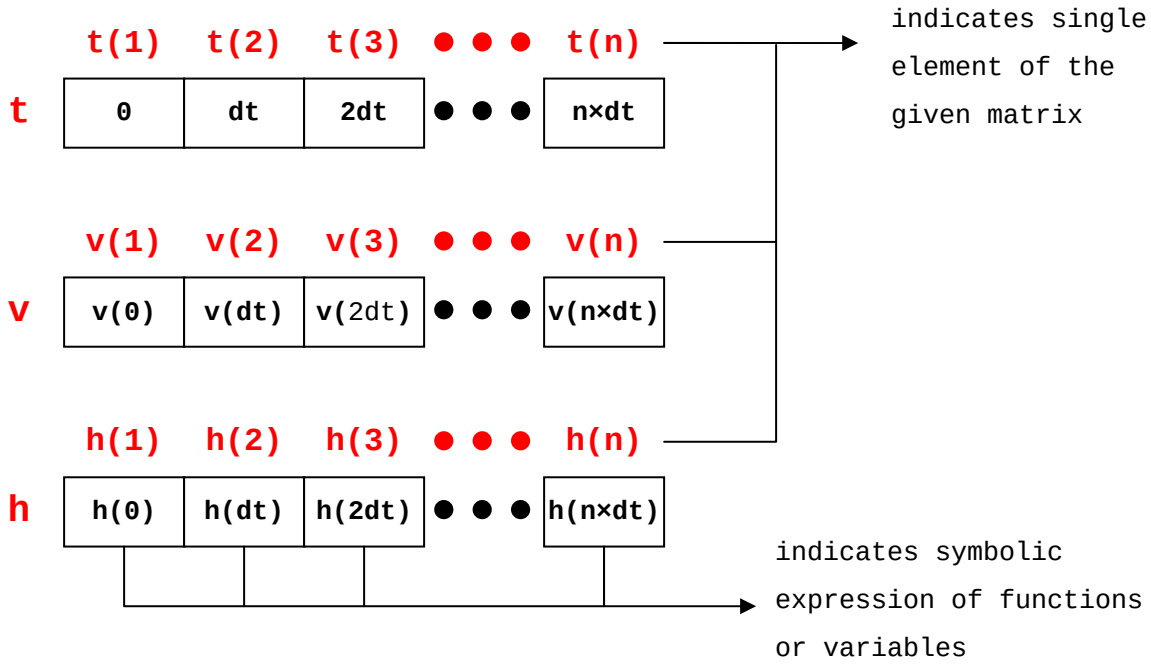
Third, variables you used in the simulation are declared such as the height, velocity and so on, but it is not really required. (Variable declarations in MATLAB are more flexible than any other languages.)

```
% Initialize numerical simulation variables (It is not required)
t=[];      % time (sec)
h=[];      % height of the ball (m)
v=[];      % velocity of the ball (m/sec)
```

Fourth, the initial conditions for solving differential equations of the ball dropping and bouncing should be assigned to the initialized variables above. For example, the first element in the matrix which stores height information has initial height when the ball starts dropping. (like `h(1)=1;`)

```
% Assign initial conditions
t(1)=0;     % t0=0
h(1)=H;     % h(t0=0)=H=1m
v(1)=0;     % v(t0=0)=0 (no initial velocity)
a=-g;       % a=-g, and always constant
```

Following diagram gives better understanding for relationship between the matrix defined in MATLAB, and the function or the variable symbolized in mathematics. Note that index of matrices should be always a positive integer. Therefore, how to translate symbolic expression into MATLAB language will be explained later.



Fifth, we define the index variable to indicate current position in the matrices. It is used for indicating the relative position to current position, or for moving current position to calculate next height of the ball bouncing.

```
% set index number i to 1 (initial condition for i=1)
 i=1;
```

Sixth, we should run the MATLAB codes for solving differential equations until ball hits on the ground. In each bounce, new initial conditions are given to MATLAB, and it then runs same codes again for solving differential equation. So, we repeat the MATLAB codes up to 3th bounce, which means MATLAB codes for solving differential equations and providing new initial conditions are repeated three times. Therefore, we first use '`for`' loop since we already know how many loops we need to calculate ball trajectory up to 3th bounce.

```
% repeat calculating velocity and height of the ball
% until the ball hits 3 times on the ground
for nbounce=1:3
```

Seventh, in the '`for`' loop, the routine to solve differential equation numerically should be inserted. Only one second derivative of the motion equation associated with the gravity is given in the problem 3.1. It is a little complicated to solve the second order differential equation directly, but one second order differential equation can be separated into two first order differential equations as below:

$$\frac{d^2h}{dt^2} = -g \quad \rightarrow \quad \frac{dh}{dt} = v \quad \& \quad \frac{dv}{dt} = -g \tag{1}$$

Based on Euler method, the first differential equations can be converted to the discrete difference equation as follow:

$$\begin{aligned} v(t) &= \frac{dh}{dt} \underset{dt \to \Delta t}{\cong} \frac{h(t+\Delta t) - h(t)}{(t+\Delta t) - t} = \frac{h(t+\Delta t) - h(t)}{\Delta t} \\ v(t)\Delta t &= h(t+\Delta t) - h(t) \\ h(t+\Delta t) &= h(t) + v(t)\Delta t \end{aligned} \tag{2}$$

Note that we assume Δt is small enough to accept numerical errors which happen during the numerical simulation. In the same way, the second differential equation can be also translated.

$$-g = \frac{dv}{dt} \underset{dt\to\Delta t}{\cong} \frac{v(t+\Delta t)-v(t)}{(t+\Delta t)-t} = \frac{v(t+\Delta t)-v(t)}{\Delta t}$$

$$-g\Delta t = v(t+\Delta t)-v(t) \tag{3}$$

$$v(t+\Delta t) = v(t)-g\Delta t$$

Now, we should describe MATLAB version of difference equations, based on the difference equations and relationship between symbolic expression and MATLAB language.

$$\begin{cases} h(t+\Delta t) = h(t)+v(t)\Delta t \\ v(t+\Delta t) = v(t)-g\Delta t \end{cases} \Rightarrow \begin{cases} \mathrm{h(n+1)=h(n)+v(n)*dt} \\ \mathrm{v(n+1)=v(n)-g*dt} \end{cases} \tag{4}$$

$$\Delta t \to \mathrm{dt}, \quad t \to \mathrm{n*dt}, \quad t+\Delta t \to \mathrm{(n+1)*dt}$$

$$\text{Where} \quad h(t) \to \mathrm{h(n)}, \quad h(t+\Delta t) \to \mathrm{h(n+1)}$$

$$v(t) \to \mathrm{v(n)}, \quad v(t+\Delta t) \to \mathrm{v(n+1)}$$

By increasing index number (`i=i+1`), we can calculate the height of the ball bouncing until the ball hits on the ground. However, how many loops we need before ball bouncing is not known. (Even though we calculate it mathematically, computer cannot do.) Therefore, `'while'` is used to calculate every heights of the ball with the given time step as long as the ball hits on the ground, or the calculated height of the ball is still either positive or zero. If the height of the ball we calculated is negative number, `'while'` loop is terminated since negative height of the ball doesn't make sense physically.

```
% if current height h(i) has negative number, terminate 'while' loop
% (it does not physically make sense.)
while h(i)>=0
    % calculate time for given index number
    t(i+1)=t(i)+dt;
    % calculate velocity of the ball at given time t(i+1)
    v(i+1)=v(i)+a*dt;
    % calculate height of the ball at given time t(i+1)
    h(i+1)=h(i)+v(i)*dt;
    % index 'i' increases by 1 to calculate next height
    i=i+1;
end
```

Eighth, we are supposed to give new initial conditions when the ball bounces. First, we eliminate

the last height we obtained since it has negative value. That's why the index number should go one step back (`i=i-1`). Then, we assume that this current position is approximately where the ball is bounced. We observe how much the new ball's velocity is changed, compared with the ball's velocity before the ball bouncing. The coefficient of restitution (COR) is given in the homework, and the definition of the COR is as below:

$$C_R = \frac{v_{2a} - v_{1a}}{v_{1b} - v_{2b}} \tag{5}$$

where

v_{1b} : Velocity of the first object before impact

v_{1a} : Velocity of the first object after impact

v_{2b} : Velocity of the second object before impact

v_{2a} : Velocity of the second object after impact

In the case of ball bouncing, $v_{2b} = v_{2a} = 0$ (for the ground), and $v_{1a} = -ev_{1b}, \quad e > 0$ (for the ball), since the second object (ground) is not moving and the direction of the first object (ball) velocity is opposite after ball bouncing. The relation between the velocities of the ball both before and after the impact is

$$C_R = \frac{v_{2a} - v_{1a}}{v_{1b} - v_{2b}} = \frac{0 - (-ev_{1b})}{v_{1b} - 0} = 0.8 \rightarrow ev_{1b} = C_R v_{1b} \quad \rightarrow e = C_R = 0.8 \tag{6}$$

Therefore, new ball's velocity has opposite sign of old ball's velocity, and is reduced by the ratio of COR. Last `'end'` command makes program go back to `'for'` loop, and run 3 times.

```
% delete current height related values and
% go back to the last height by decreasing i by 1
% (The last position where the ball is above the ground)
t(i)=[]; v(i)=[]; h(i)=[];
i=i-1;
% Assume that the ball bounce slightly above the ground,
% the initial velocity of the ball after bouncing is calculated with
% the final velocity of the ball before bouncing
% and coefficient of restitution
v(i)=-COR*v(i);
```

```
    % index 'i' increases by 1 to calculate next height
    % with new condition after the ball bouncing.
end
```

Trajectory plot for the ball bouncing is shown in the end of Problem 3.3 solution.

Problem 3.2 : Calculate the trajectory of a ball dropping and bouncing with Stokes' drag

In this problem, all the codes you developed in the previous problem are used here except for the acceleration part. In problem 3.1, the acceleration is only gravity, and constant everywhere. However, the new drag called Stoke's drag is introduced in this problem, and it should be plugged into acceleration calculation.

$$a(t) = -g - bv(t) \quad \rightarrow \quad a(n\Delta t) = -g - bv(n\Delta t) \quad \Rightarrow \quad \mathrm{a(n)=-g-b*v(n)} \qquad (7)$$

```
    % if current height h(i) has negative number, terminate 'while' loop
    % (it does not physically make sense.)
    while h(i)>=0
        % calculate time for given index number
        t(i+1)=t(i)+dt;
        % calculate acceleration of the ball at given time t(i+1)
        a(i+1)=-g-b*v(i);
        % calculate velocity of the ball at given time t(i+1)
        v(i+1)=v(i)+a(i)*dt;
        % calculate height of the ball at given time t(i+1)
        h(i+1)=h(i)+v(i)*dt;
        % index 'i' increases by 1 to calculate next height
        i=i+1;
    end
```

In addition, initial condition for the acceleration should be also considered.

$$a(0) = -g - bv(0) \quad \rightarrow \quad \mathrm{a(1)=-g-b*v(1)} \qquad (8)$$

```
% Assign initial conditions
t(1)=0;         % t0=0
h(1)=H;         % h(t0=0)=H=1m
```

```
v(1)=0;        % v(t0=0)=0 (no initial velocity)
a(1)=-g-b*v(1);  % a(t0=0)=-g-bv(t0=0)
```

In the section of defining constants, Stokes' drag coefficient is also defined as below.

```
% Define constants
b=0.8;     % constant for acceleration induced by stokes' drag (1/sec)
```

Trajectory plot for the ball bouncing is shown in the end of Problem 3.3 solution.

Problem 3.3 : Calculate the trajectory of a ball dropping and bouncing with quadratic drag

In the same way, the ball bouncing with quadratic drag can be also calculated by modifying the codes for calculating the acceleration and adding the initial condition of the ball bouncing in problem 3.2

$$a(t) = -g - v(t)\left|v(t)\right| \quad \rightarrow a(n\Delta t) = -g - v(n\Delta t)\left|v(n\Delta t)\right| \quad \Rightarrow \quad \mathrm{a(n)=-g-v(n)*abs(v(n))} \tag{9}$$

```
% if current height h(i) has negative number, terminate 'while' loop
% (it does not physically make sense.)
while h(i)>=0
    % calculate time for given index number
    t(i+1)=t(i)+dt;
    % calculate acceleration of the ball at given time t(i+1)
    a(i+1)=-g-c*v(i)*abs(v(i));
    % calculate velocity of the ball at given time t(i+1)
    v(i+1)=v(i)+a(i)*dt;
    % calculate height of the ball at given time t(i+1)
    h(i+1)=h(i)+v(i)*dt;
    % index 'i' increases by 1 to calculate next height
    i=i+1;
end
```

$$a(0) = -g - cv(0)\left|v(0)\right| \quad \rightarrow \quad \mathrm{a(1)=-g-c*v(1)*abs(v(1))} \tag{10}$$

```
% Assign initial conditions
```

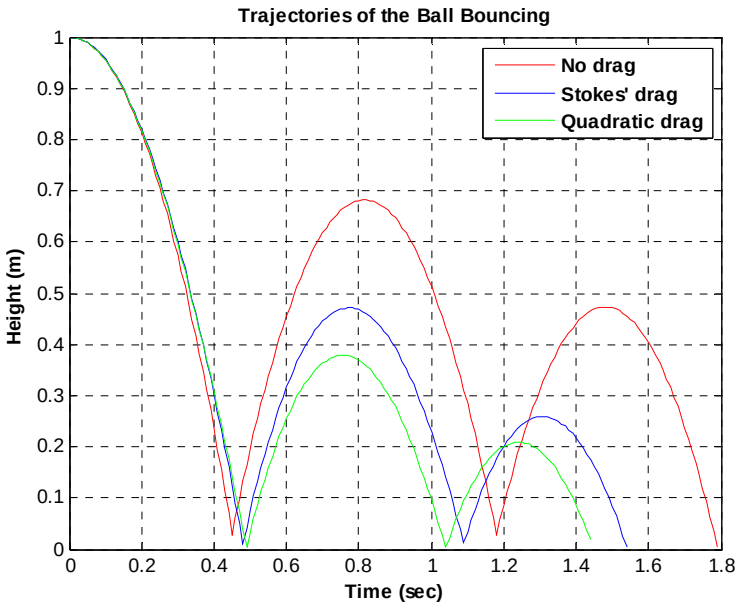
```
t(1)=0;                    % t0=0
h(1)=H;                    % h(t0=0)=H=1m
v(1)=0;                    % v(t0=0)=0 (no initial velocity)
a(1)=-g-c*v(1)*abs(v(1));   % a(t0=0)=-g-cv(t0=0)|(v(t0=0)|
```

In the section of defining constants, quadratic drag coefficient is also defined as below.

```
% Define constants
c=0.5;     % constant for acceleration induced by quadratic drag (1/m)
```

All the plots for the ball bouncing in problem 3.1-3.3 are generated. Following codes are for plotting three trajectories of the ball bouncing with different drag conditions.

```
>> [t1,h1]=ball3;
>> [t2,h2]=ball3stokes;
>> [t3,h3]=ball3quadratic;
>> plot(t1,h1,'r',t2,h2,'b',t3,h3,'g');
>> grid on;
>> legend('\bfNo drag','\bfStokes'' drag','\bfQuadratic drag');
>> xlabel('\bfTime (sec)');
>> ylabel('\bfHeight (m)');
>> title('\bfTrajectories of the Ball Bouncing');
```



Problem 3.4 : Calculate more accurate bouncing time of the ball with given spatial resolution

Go back to the solution in problem 3.1. What we are interested in is the part of determining the bouncing time within the given tolerance (or spatial resolution) by changing step size. Following algorithm describes how to find the step size to satisfy the spatial resolution criterion.

1. Current index number is set to the last one. (`i=i-1`)
2. Define new variable of the step size as `'dtt'`, and set it to the same value of `'dt'`
3. Check if the next height is positive and greater than the given time resolution, the next time is set to the ball bouncing time, and the next velocity becomes the final velocity before the ball hits on the ground.
4. If the next height is negative, the current step size decreases by the factor of 2 (`0.5dtt`). Otherwise, the current step size is increases by the factor of 1.5 (`1.5dtt`).

$$dtt = dtt * \left(1 + 0.5 \frac{|h(t)|}{h(t)} \right) \tag{11}$$

5. Calculate the next height with new time step `'dtt'` .
6. Go back to Step 3.

The above algorithm is implemented by MATLAB, and shown as below:

```
% go back to the last height by decreasing i by 1
% (The last position where the ball is above the ground)
i=i-1;
% define new time step (adaptive time step)
dtt=dt;
% if h(t) satisfies spatial resolution condition,
% terminate 'while' loop.
while h(i+1)>0.001 || h(i+1)<0
    dtt=dtt+dtt/2*sign(h(i+1));
    % calculate time for given couter number
    t(i+1)=t(i)+dtt;
    % recalculate velocity of the ball
    v(i+1)=v(i)+a*dtt;
```

```
        % recalculate the height of ball
        h(i+1)=h(i)+v(i)*dtt;
    end
    % index 'i' increases by 1 with accepting new height
    % with given spatial resolution
    i=i+1;
    % Assume that the ball bounces on the ground,
    % the initial velocity of the ball after bouncing is calculated with
    % the final velocity of the ball before bouncing
    % and coefficient of resititution
    v(i)=-COR*v(i);
    % go back to calculate next height
```

The following codes and plot is for generating trajectory of the ball bouncing with the enough spatial resolution.

```
>> [t,h]=ball3spatial; plot(t,h);
>> grid on;
>> xlabel('\bfTime (sec)');
>> ylabel('\bfHeight (m)');
>> title( ...
      '\bfTrajectories of the Ball Bouncing with Given Spatial Resolution');
```