Solving Ordinary Differential Equations (ODEs)

Note: I confirm that I did not use codes from the web or from past years' assignments and that the work I submit is my own and my own only.

1 Introduction

The goal of this exercise is to simulate the motion of a disk interacting with the walls of a closed container. To solve for the evolution of the position of the center of mass, an ordinary differential equation must be solved using Euler's scheme along with Newton's 2^{nd} Law of Motion.

The two forces acting on the disk are the weight ($f_w = mg$) and the drag ($f_d = -c_d||\mathbf{v}||\mathbf{v}$). Where $\mathbf{v}$=(u,v) is the velocity vector, m is the mass, c_d is the drag coefficient, and $\mathbf{g}$ is the gravity vector.

The equation of motion is given by the ODE:

$$\begin{aligned} \frac{d\mathbf{v}}{dt} &= \mathbf{g} - \frac{c_d}{m}||\mathbf{v}||\mathbf{v} \\ \mathbf{v}(t=0) &= \mathbf{v}^0 \end{aligned} \tag{1}$$

where $\mathbf{v}^0$ is the given initial velocity. Damping and friction forces apply when the ball interacts with the container's wall. The normal velocity is damped by a factor α and the tangential velocity is damped by a factor β.

The given initial parameters are Δt = .03 with the size of the closed container to be [0,1]x[0,1] m. An initial velocity of $\mathbf{v}^0$=(.3,0) $\frac{m}{s}$, disk's radius r = .05 m, initial location of the disk at (x,y) = (.5,1-r) m, α=.8, β=0.99, c_d=0.25 $\frac{kg}{s}$, m=1 kg, and g=.0981 $\frac{m}{s^2}$. Also, assume the time span is from 0 to 60 seconds.

2 Derivation of Equations

Newton's 2^{nd} law:

$$\begin{aligned} F = m\vec{a} = \sum \vec{F} &= \vec{F}_{gravity} + \vec{F}_{drag} \\ m\vec{a} &= m\vec{g} + c_d\vec{v}^2 \\ \frac{d\vec{v}}{dt} &= \vec{g} - \frac{c_d}{m}||\vec{v}||\vec{v} \end{aligned} \tag{2}$$

A general version of the Euler's scheme (or step) is shown below:

$$\begin{aligned} v^{n+1} &= v^n + \Delta t\frac{dv}{dt}|_{t=t^n} \\ v^{n+1} &= v^n + \Delta t(\vec{g} - \frac{c_d}{m}(v^n)^2) \end{aligned} \tag{3}$$

The exact solution to the Euler scheme equation shown above is as follows:

$$v = \sqrt{\frac{gm}{c_d}} tanh(\sqrt{\frac{gc_d}{m}} t) \tag{4}$$

When the disk is not touching the container's wall, the following position and velocity equations are used to describe the disk's center of mass:

$$\begin{aligned} unp1 &= un + dt * a_x \\ vnp1 &= vn + dt * a_y \\ xnp1 &= xn + dt * unp1 \\ ynp1 &= yn + dt * vnp1 \end{aligned} \tag{5}$$

Due to drag, the acceleration in the x and y directions vary according to the following equations:

$$\begin{aligned} a_x = \frac{du}{dt} &= 0 - \frac{c_d}{m}\sqrt{un^2 + vn^2} un^2 \\ a_y = \frac{dv}{dt} &= -0.0981 - \frac{c_d}{m}\sqrt{un^2 + vn^2} vn^2 \end{aligned} \tag{6}$$

When the disk touches a side of the container, the velocity and position equations change due to losses from damping. Using the right wall as an example:

$$\begin{aligned} xnp1 &= 1 - r \\ dtnew &= \frac{xnp1 - xn}{un} \\ ynp1 &= yn + dtnew * vn \\ unp1 &= -\alpha * un \\ vnp1 &= \beta * vn \end{aligned} \tag{7}$$

3 Algorithms

The MATLAB algorithm used to simulate a bouncing circular disk consists of accounting for the change in position and velocity of the center of mass of the disk during the time span of 60 second.

To model this, you must use a function given on gauchospace called *DrawDisk* in order to depict a disk on the plot. The next step is to evaluate the position and velocity of the disk's center of mass at every time step interval, given to be 0.03s. Since the Euler's scheme is used, there will be a different value at each discrete time step. Each value of position and velocity are dependant on the previously calculated value over this discretized time span. In this case, we considered the damping effects of air α and energy loss due to collision β which will eventually stop the disk's motion.

Once the equations for position, velocity, and acceleration are derived it is important to know what will happen to our disk once it reaches one of the borders (or boundaries) of the plot. In order to address how the disk will bounce off the wall, we need to figure out two things: collision detection and how the disk will react to the wall. When dealing with the "container", we would like the disk to stay inside the boundary. This will require several if-statements to ensure the disk will not go outside the specified axis. As shown in the code below, at each boundary there will be an updated position and velocity value depending on which side of the container the disk will collide with. The new velocity and position values derived under the assumption there are losses due to damping, which also will affect acceleration.

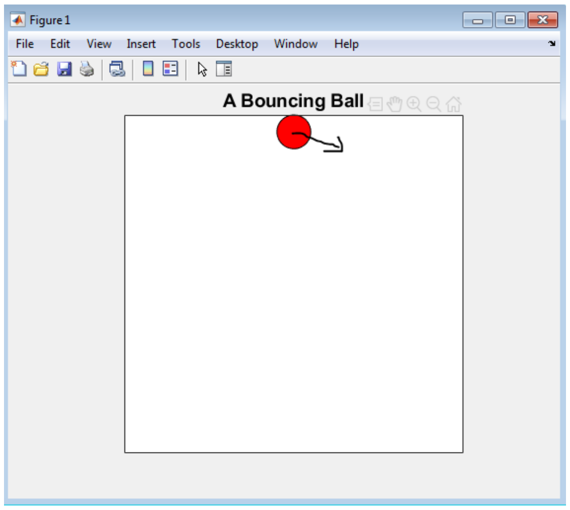
4 Implementation in Matlab and Results

The Matlab implementation, with comments, is given here:

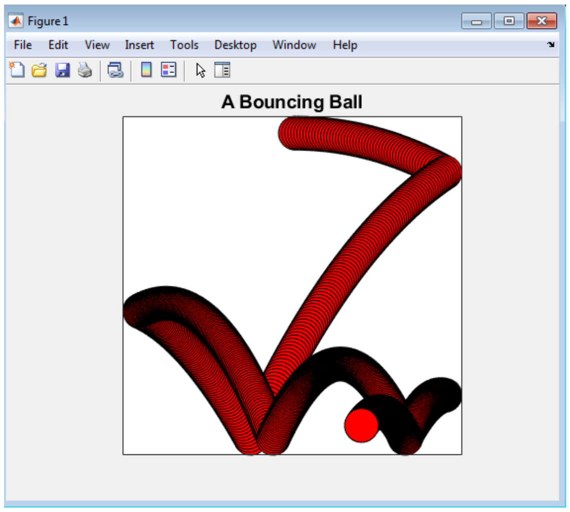
```
%ME17: Solving Ordinary Differential Equations - Alex Nguyen

%Note: I confirm that I did not use codes from the web or from past years'
%assignments and that the work that I submit is my own and my own only.

clc
clf
clear all

m = 1; %mass [kg]
cd = 0.25; %drag coefficient [kg/s]
alpha = 0.8; %normal velocity damping coefficient
beta = 0.99; %tangential velocity damping factor
r = 0.05; %disk radius [m]
un = 0.3; vn = 0; %initial velocity [m/s]
xn = 0.5; yn = 1 - r; %initial position [m]
ax = 0;ay = -0.0981; %initial acceleration [m/s^2]
dt=0.03; %time step [s]
t = 0; %time starts at zero [s]
tfinal=60; %final time value [s]

while t<tfinal
    if t+dt>tfinal
        dt=tfinal-t;
    end
```

```
unp1 = un + dt*ax; %updated x-velocity
vnp1 = vn + dt*ay; %updated y-velocity

xnp1 = xn + dt*un; %updated x-position
ynp1 = yn + dt*vn; %updated y-position

%Right Wall Detection
if xnp1+r>1
    xnp1 = 1-r;
    dtnew = (xnp1 - xn)/un; %new time
    ynp1 = yn + dtnew*vn;
    unp1 = -alpha*un;
    vnp1 = beta*vn;
end

%Bottom Wall Detection
if ynp1<r
    ynp1 = r;
    dtnew = (ynp1 - yn)/vn; %new time
    xnp1 = xn + dtnew*un;
    unp1 = beta*un;
    vnp1 = -alpha*vn;
end

%Left Wall Detection
if xnp1<r
    xnp1 = r;
    dtnew = (xnp1 - xn)/un; %new time
    ynp1 = yn + dtnew*yn;
    unp1 = -alpha*un;
    vnp1 = beta*vn;
end

%Top Wall Detection
if ynp1>1-r
    ynp1 = 1-r;
    dtnew = (ynp1 - yn)/vn; %new time
    xnp1 = xn + dtnew*xn;
    unp1 = beta*un;
```

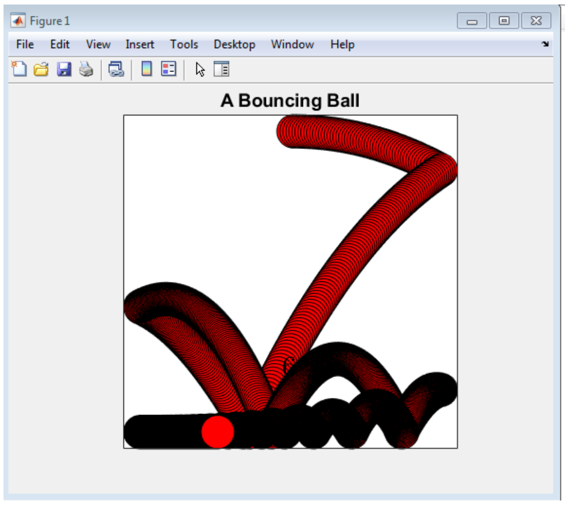
```
        vnp1 = -alpha*vn;
    end

    %Draw Disk
    Draw_Disk(xnp1,ynp1,r);
    axis equal;
    hold on %shows disk path
    axis([0 1 0 1]);
    pause(0.01*dt);

    %Update Variables in loop
    t = t + dt; %updated time
    xn = xnp1; yn = ynp1; %position
    un = unp1; vn = vnp1; %velocity
    ax = 0-(cd/m)*sqrt(un^2+vn^2)*un; %x-acceleration
    ay = -0.0981-(cd/m)*sqrt(un^2+vn^2)*vn; %y-acceleration
end
```



(a) Initial Position



(b) Disk Position at t = 20s



(c) Disk Position at t = 60s

Figure 1: Varying Disk Position Over Time Span