Homework 5

Ryan Branagan

February 19, 2019

Problem 1

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw5_p1.py
#2/5/19

import numpy as np
import pylab as p

def Error(Exp,Exact):
    return np.abs(Exp-Exact)/np.abs(Exact)
#%%
#1 - Discretized Equations

#yi1 = yi + vi*dt
#vi1 = vi + (-g-bvi)*dt
#%%
#2 - Use Discretized Equations to solve for a object's motion

def objfall(y0,v0,t0,tf,points,B):
    g = 9.81
    #Times
    t = np.linspace(t0,tf,points)
    dt = t[1]-t[0]

    #Using Equations
    yt = np.zeros_like(t)
    vt = np.zeros_like(t)
    yt[0] = y0
    vt[0] = v0
    for i in range(points-1):
```

```
        vt[i+1] = vt[i]+(-g-B*vt[i])*dt
        yt[i+1] = yt[i]+vt[i]*dt
    return yt,vt,t
#%%
#3 - Plot Part 2

#Initial Conditions
y0 = 12.
v0 = 35.
b = 0
t0 = 0
tf = 8
points = 1000

#Get Points
yt,vt,t = objfall(y0,v0,t0,tf,points,b)

#Plotting
fig1, pos = p.subplots(1,1)
pos.plot(t,yt,'o')
pos.set_xlabel('Time [s]')
pos.set_ylabel('Height[m]')
pos.set_title('Height of an Object Over Time')
pos.axhline(0,color='k',linestyle='--')
p.show()
#%%
#4 - Analytical Equations

#Initial Conditions
y0 = 12.
v0 = 35.
b = 10**-6
t0 = 0
tf = 8
points = 1000

def AnV(v0,B,t0,tf,points):
    g = 9.81
    #Times
    t = np.linspace(t0,tf,points)
    #Velocities
    v = (v0+(g/B))*np.exp(-B*t)-(g/B)
    return t,v
```

```
def AnY(y0,v0,B,t0,tf,points):
    g = 9.81
    #Times
    t = np.linspace(t0,tf,points)
    #Y Positions
    y = (-(v0/B)-(g/B**2))*np.exp(-B*t)-(g*t/B)+(v0/B)+(g/B**2)+y0
    return t,y

def NoDrag(y0,v0,t0,tf,points):
    g = 9.81
    #Times
    t = np.linspace(t0,tf,points)
    #Y Positions without Drag
    NoDrag = y0+v0*t-0.5*g*(t**2)
    return t,NoDrag

t,ND = NoDrag(y0,v0,t0,tf,points)
print(t[points-1])
print(ND[points-1])
y = AnY(y0,v0,b,t0,tf,points)[1]
print(y[points-1])
#%%
#5 - Relative Error

#Initial Conditions
y0 = 12.
v0 = 35.
b = 10**-6
t0 = 0
tf = 8

#Resolutions
R = np.array([10,20,1000,10000])

#Finding height at 8s
yE = np.zeros_like(R)
yK = np.zeros_like(R)
for i,r in enumerate(R):
    yE[i] = objfall(y0,v0,t0,tf,r,0)[0][r-1]
    yK[i] = NoDrag(y0,v0,t0,tf,r)[1][r-1]

Err = Error(yE,yK)
print(Err[0]/2)
print(Err[1])
```

```
#%%
#6 - Non-Zero Friction

#Initial Conditions
y0 = 12.
v0 = 35.
t0 = 0
tf = 8
points = 10000

#Betas
Bs = np.array([0,0.3,0.6,0.9])

#Getting Points
ya = NoDrag(y0,v0,t0,tf,points)[1]

ys = [np.zeros(points),np.zeros(points),np.zeros(points),np.zeros(points)]
vs = [np.zeros(points),np.zeros(points),np.zeros(points),np.zeros(points)]
for i,b in enumerate(Bs):
    ys[i],vs[i],ts = objfall(y0,v0,t0,tf,points,b)

va = v0-(9.81)*ts

#Plotting
fig1,pos = p.subplots(1,1)
for i,b in enumerate(Bs):
    pos.plot(ts,ys[i],'o',label=r"$\beta=$"+str(b))
pos.plot(ts,ya,'k:')
pos.axhline(0,color='k',linestyle='--')
pos.legend()
pos.set_title('Position Over Time for Various Beta Values')
pos.set_xlabel('Time [s]')
pos.set_ylabel('Height [m]')

fig2,vel = p.subplots(1,1)
for i,b in enumerate(Bs):
    vel.plot(ts,vs[i],'o',label=r"$\beta=$"+str(b))
vel.plot(ts,va,'k:')
vel.axhline(0,color='k',linestyle='--')
vel.legend()
vel.set_title('Velocity Over Time for Various Beta Values')
vel.set_xlabel('Time [s]')
vel.set_ylabel('Velocity [m/s]')
```

The discretized equations are:

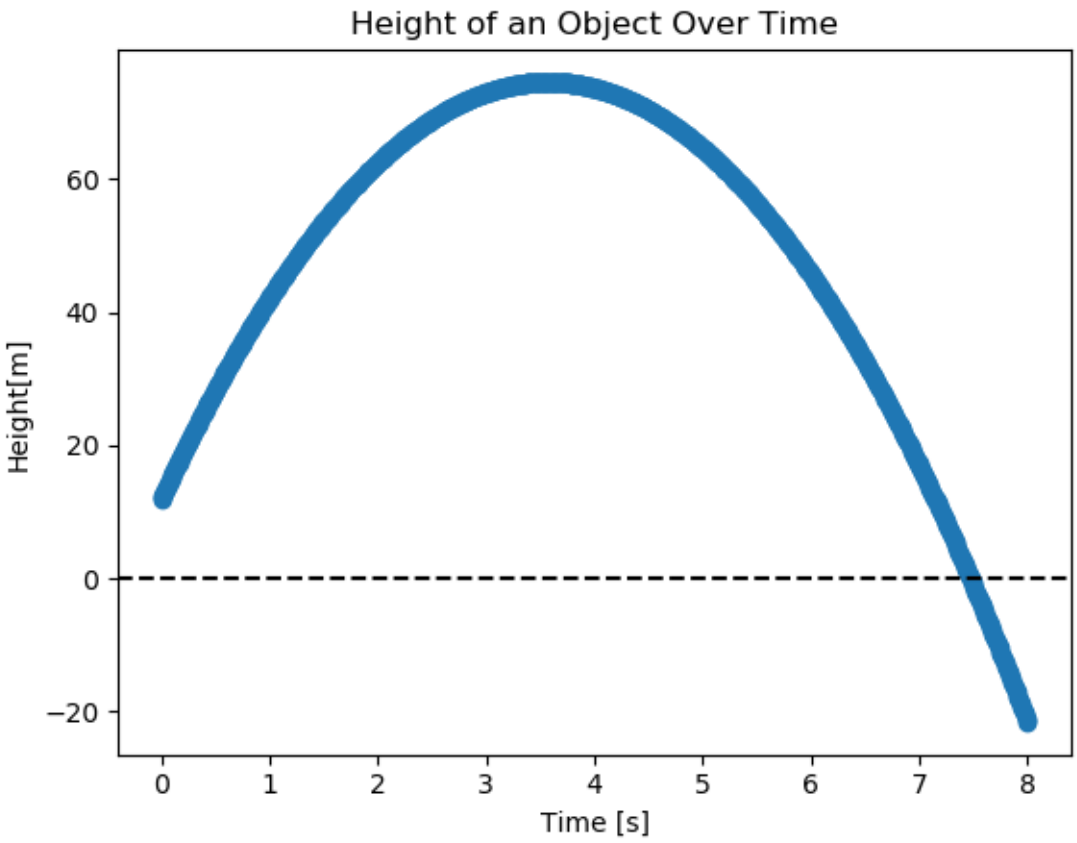
$$y_{i+1} = y_i + v_i * dt \tag{1}$$

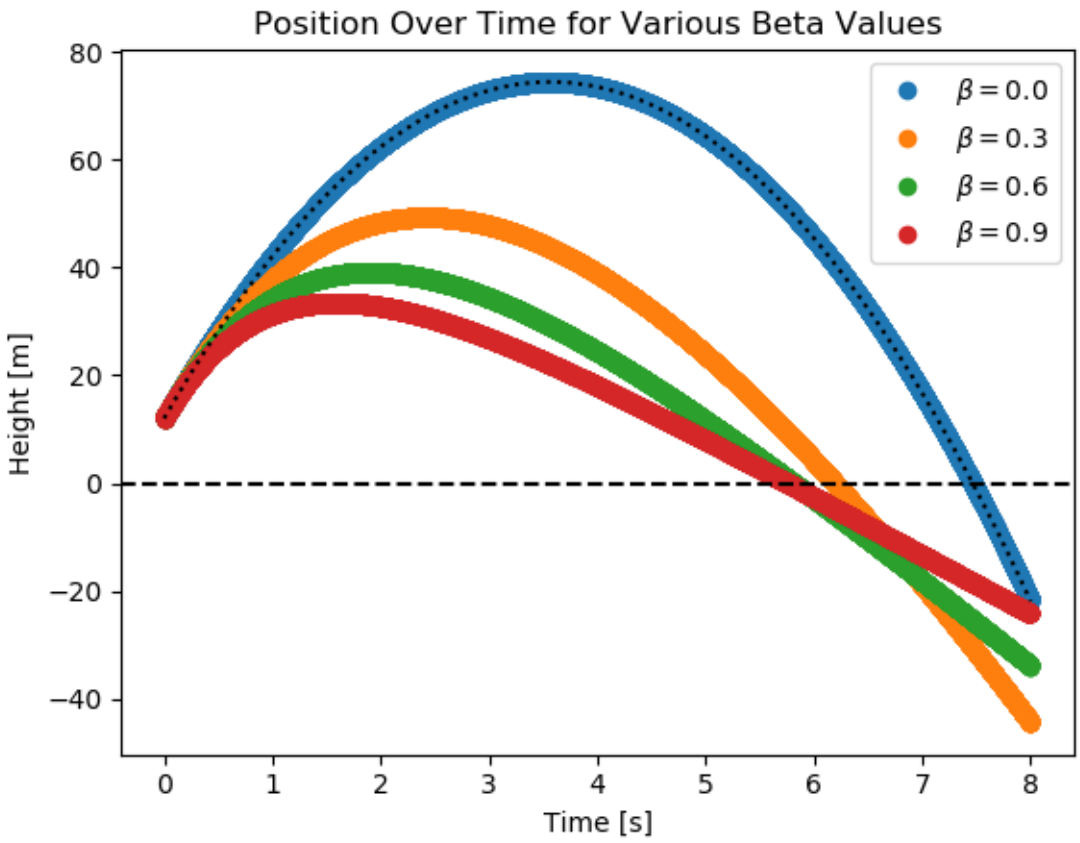
$$v_{i+1} = v_i + (-g - bv_i) * dt \tag{2}$$

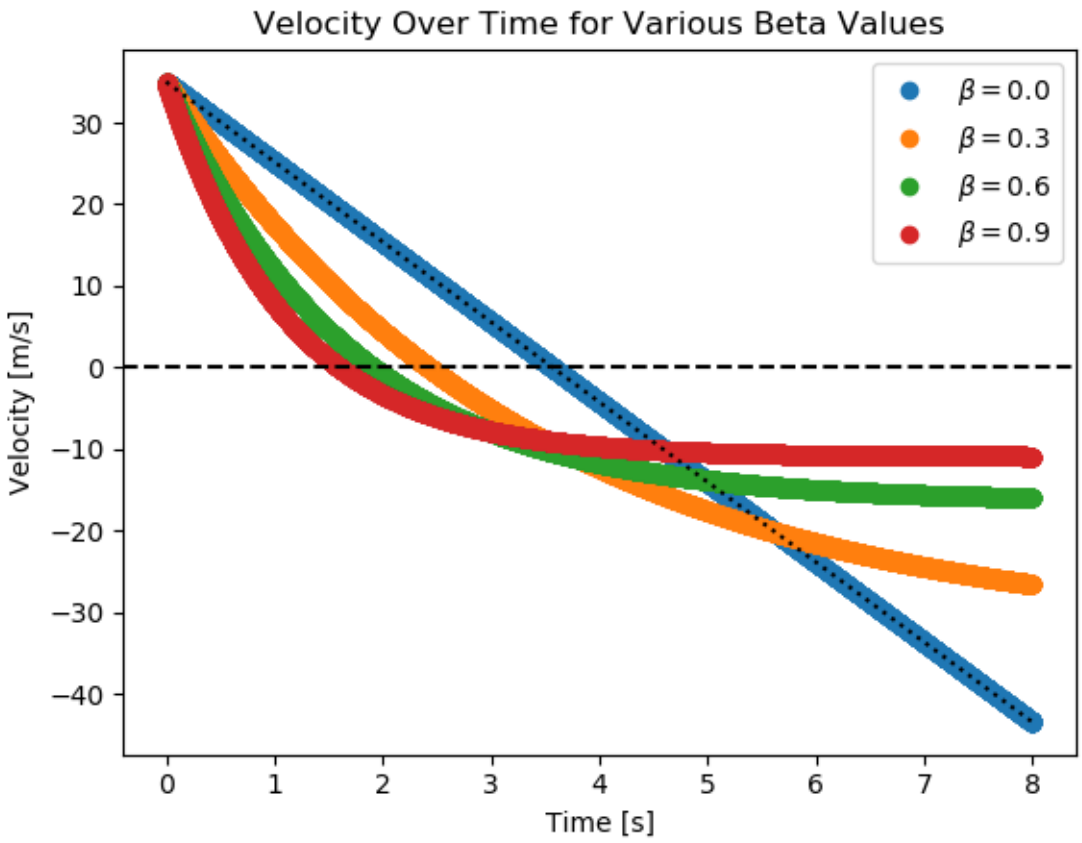
The error between the discretized equations and the analytical equation is approximately cut in half when the resolution is doubled. The question didn't specify a plot and it is only at a certain time of 8 seconds so I manually showed this. My initial resolution was 10 and my second resolution was 20. Dividing the error of 10 by two and printing the error of 20 shows this sufficiently. Half of the first error is 0.786 and the second error is 0.762. For high resolutions this relationship breaks down a little and is very approximate.

For calculating the position and velocity with drag, I chose a resolution of 10,000. I chose this resolution because it would be much more accurate and the program still executed very quickly. Even though for low resolutions Euler's Method can be very off from analytical values, at high resolutions it looks just like the analytical values.

Looking at the plots of position and velocity with drag, they make sense. Looking at the velocity graphs as time goes on they converge to a single value. I believe that this is the terminal velocity which is accurate to what happens in real life when falling. This is also reflected in the position graph because the graph turns from parabolic to looking linear when approaching the terminal velocity.


Height of an Object Over Time
Height[m]
60
40
20
0
−20
0
1
2
3
4
5
6
7
8
Time [s]


Position Over Time for Various Beta Values
Height [m]
Time [s]
80
60
40
20
0
−20
−40
0
1
2
3
4
5
6
7
8
$\beta = 0.0$
$\beta = 0.3$
$\beta = 0.6$
$\beta = 0.9$


Velocity Over Time for Various Beta Values
$\beta = 0.0$
$\beta = 0.3$
$\beta = 0.6$
$\beta = 0.9$
Velocity [m/s]
30
20
10
0
−10
−20
−30
−40
0
1
2
3
4
5
6
7
8
Time [s]