Lab 3: Projectile Motion

23/03/2022
Kevin Grainger
20333346

Contents:

Introduction:

The aim of this lab is to analyze the motion of a projectile under air resistance. Air resistance is force proportional to the velocity of the object (ie. $F(x') = kx' = bx' - c(x')^2$). This leaves us with a differential equation describing the motion of the projectile. The dynamics of which are considerably more complex than the approximation of zero air resistance, as often used. By investigating the scaling of this force with velocity and the effect of air resistance on each component of velocity, we can plot the trajectory of a projectile under differing resistive forces. Lastly I altered the above procedure to plot an air resistance that is related quadratically to velocity, meaning that each component is dependent on the other ($V_x \propto V_y$). This is a more true to life scenario.

Methodology:

Overview:

The procedure follows as such:

Exercise 1:

Investigating the scaling of the separate terms of: $F_{AR}(V) = bV - c(V)^2$ (Air Resistance) against velocity graphically. From this I can define a range of V for which each term contributes or is negligible. I use this to graph the velocity and path of a falling baseball..

Exercise 2:

From the approximation made in Ex1, we can formulate an equation of motion (differential) that describes the vertical velocity over a time step.
From here I can compare the numerical solution to the equation of motion to the analytical.

Lastly I graphed V_y against time, and time to reach the ground (T_g) against mass.

Exercise 3:

Using the approximation that air resistance grows linearly, the Velocity components are decoupled. From this I graph the X and Y position of the projectile under approximated air resistance. This is compared graphically to a projectile fired in a vacuum. The optimum firing angle is then found.

Exercise 4:

The code from Ex 3 can be altered to reflect a coupled relationship between velocity components in the quadratic term of F_{AR}.

The new trajectory is graphed in comparison.

Code Execution:

Exercise 1:

I defined the air resistance function in python as f(D,V).

$F_{AR}(V) = bV - c(V)^2$

We are given values for the constants.

$b = 1.6 \cdot 10^{-4}$

$c = 0.25$

```
B=1.6*(10**-4)
C=0.25
# b = B D and c = C D2
#s f(V) = bV + c V2
#DV=k
k=np.arange(0,10,0.2)
def f(D,V):
    return B*k+C(k**2)
```

I graphed the values of bV and $c(V)^2$ against velocity separately, to see how the resistive force of air scales with velocity in certain ranges.

```
plt.plot(k,B*k,'red',label="bV")
plt.plot(k,C*(k)**2,'green',label="cV^2")
plt.xlabel('DV')
plt.ylabel('bV & cV^2')
plt.title('(bV & cV^2) vs DV')#
plt.legend(loc="upper left")
#B*k can be neglected for all values of DV
#Ck^2 can be ignored from 0-2
```

(ii)To identify the range of 'DV' diameter that makes each of these negligible

We can do this graphically by finding where $bV >> cV^2$.

Exercise 2

(i)

We want to simplify our air resistance, to do this I used the ranges for which $bV >> cV^2$,in order to drop the quadratic term.

Given the spherical piece of dust taken for this exercise is of diameter 10^{-4} and and mass density $2*10^3$, the terminal velocity can be calculated: (where V is the terminal velocity)

$mg = B(DV) - C(DV)^2$

Giving V=0.002

Thus $DV=0.2*10^{-6}$

Which is well inside the ranges defined in Exercise 1

(ii)

To find the progression of our Vertical Velocity:

Given $\Delta V_y = -gm - \frac{b}{m} V_y \Delta t$

Define the constants (note our object is released from rest)

```
#vertical velocity
dt=0.01
g=9.8
tmax=10
t=0
Vy=0
m=0.1047197
```

Create a while loop to iterate our velocity and time values:

```
while t<tmax:
    deltaVy=-(g*dt)-((b/m)*Vy*dt)
    Vy=Vy+deltaVy
    t=t+dt
    print (t)
    print (Vy)
    plt.scatter(t,Vy, color='red')
#plt.plot(Vy,t)
```

This approach ended up not working as well as I thought, for certain arrangements I couldn' t get Vy to iterate properly. So I switched my approach to one that uses lists.

```
def VertVelo(v,b,m,dt):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our velocities
    Vy=IntialVy #to give us the starting value
    while t<tmax :
        Velolist.append(Vy)
        Timelist.append(t)
        Vy=Vy-(g*dt+(b/m)*Vy*dt)
        t=t+dt
    plt.plot(Timelist,Velolist)
    plt.ylabel('Vertical Velocity')
    plt.xlabel('Time')
#VertVelo(0,1.6e-08,1,dt)
```

(iii)

By varying the mass value we can see how this changes the slopes of the Vy progression. This is done using a while loop to cycle through the different masses.

```

#to allow us to vary the mass
i=1 #It is easier to use whole numbers
while i<5:
    VertVelo(0, 0.4, i, dt) #a larger b value makes t
    # of air resistance clearer
    i=i+1
plt.show()
```

(iv)

Given the analytical solution is:

$V_y = \frac{mg}{b}(e^{-bt/m} - 1)$

We can compare this to the Numerical solution by creating an error function, showing the difference (Δ) between the two solutions at each point.

I achieved this by using the same while loop as in (iii) but our function is now:

Vy(Numerical)-Vy(analytical)

```
46
47  def errorfunc(Vy,b,m,dt):
48      t=0
49      TList=[] #change list name to avoid clashes
50      Delta=[] #The difference between methods
51      InitialVy=Vy
52      while t<tmax :
53          Delta.append(((m*g/b)*(np.e**(-b*t/m)-1))-(g*dt+(b/m)*Vy*dt)
54          TList.append(t)
55          t=t+dt
56
57      plt.plot(TList,Delta,color='red')
58      plt.xlabel('Time')
59      plt.ylabel('Error')
60  errorfunc(IntialVy,1,1,dt)
61  plt.show()
62
```

(v)

We are given a system to simulate:

A grain of diameter 10^{-4}, mass 0.1047197kg, dropped from a height of H=5m.

By changing the function to find the Y position given the vertical velocity. This is done with time steps that edit the y-position continuously given the velocity at each instant.

Using this function, the time to reach the ground can be calculated.

This is graphed against mass, to show the relationship between mass and time to reach the ground.

I used the same while loop as before but set its condition to h>0. By using the len() function on our time array we can calculate the time taken to reach the ground.

```
25  def VertPosition(v,b,m,dt,height):
26      t=0 #always starts from 0
27      Timelist=[] #a range of time t<tmax
28      Velolist=[] #same but for our velocities
29      Height=[]
30      Vy=IntialVy #to give us the starting value
31      height=h
32      while height>0 :
33          Velolist.append(Vy)
34          Timelist.append(t)
35          Height.append(height)
36          Vy=Vy-(g*dt+(b/m)*Vy*dt)
37          height=height+(Vy*dt)
38          t=t+dt
39          #print(height)
40
41
42      print(len(Timelist))
43      #plt.plot(Timelist,Velolist)
44      #plt.ylabel('Vertical Velocity')
45      #plt.xlabel('Time')
46      return len(Timelist)
47  for i in range(1,10,1):
48      VertPosition(0, 1, i, dt, h)
49  #VertVelo(0,1.6e-08,1,dt)
```

I then graphed these results against the respective masses to derive a relationship.

Exercise 3

By computing both Vx and Vy we can then plot the trajectory of a projectile fired at angle theta. This is done simply by using the exact same mechanism as in Ex 2 (iv) but for both displacements, x and y.

$$\frac{V_y}{dt} = -g - \frac{b}{m}V_y$$

$$\frac{V_x}{dt} = -\frac{b}{m}V_x$$

Using the exact same mechanism as in the last part of Ex.2 . I just made two different functions and graphed their returned quantities (X -Position vs. Y-Position).
I limited the y-axis instead of creating a h>0 condition as it was easier.

```
25
26  def VertPosition(v,b,m,dt,height):
27      t=0 #always starts from 0
28      Timelist=[] #a range of time t<tmax
29      Velolist=[] #same but for our velocities
30      Height=[]
31      Vy=IntialVy #to give us the starting value
32      height=h
33      while t<tmax :
34          Velolist.append(Vy)
35          Timelist.append(t)
36          Height.append(height)
37          Vy=Vy-(g*dt+(b/m)*Vy*dt)
38          height=height+(Vy*dt)
39          t=t+dt
40      return Height
41
42  def XPosition(v,b,m,dt,xpos):
43      t=0
44      xpos=IntialXpos
45      TList=[]
46      VeloList=[]
47      XposList=[]
48      Vx=IntialVx
49      while t<tmax:
50          VeloList.append(Vx)
51          TList.append(t)
52          Vx=Vx-((b/m)*Vx*dt)
53          XposList.append(xpos)
54          xpos=xpos+(Vx*dt)
55          t=t+dt
56      return XposList
57
```

(ii)

To find the optimum launch angle I created a new function that ran on the condition that Height>0. This function returns the X-position when the projectile hits the ground.

```
29  def XPosition(v,b,m,dt,xpos,theta):
30      t=0
31      xpos=IntialXpos
32      TList=[]
33      VeloList=[]
34      XposList=[]
35      Vx=10*(np.sin(theta))
36      Vy=10*(np.cos(theta))
37      height=h
38      while t<tmax and height>0:
39          VeloList.append(Vx)
40          TList.append(t)
41          Vx=Vx-((b/m)*Vx*dt)
42          Vy=Vy-(g*dt+(b/m)*Vy*dt)
43          XposList.append(xpos)
44          height=height+(Vy*dt)
45          xpos=xpos+(Vx*dt)
46          t=t+dt
47      return xpos
```

By making this function a function of theta, I can vary theta's value using a for loop. Then create a list of these values and graph them against theta.

```
48
49  Ranges=[]
50  for i in theta:
51      Ranges.append(XPosition(10,0.5,1,dt,0,i))
52  plt.plot(theta,Ranges)
53  plt.xlabel('Theta')
54  plt.ylabel('Range')
55  plt.title('Range Vs. Launch Angle')
56
```

Exercise 4:

In reality the differential equations describing the projectile motion are not decoupled. This means in actuality we equations that show the velocity components to be quadratically dependent on each other.

By only altering the Vy and Vx expressions to the one shown in the lab manual we can graph the true trajectory of a projectile under air resistance.

```
44  def XPosition(v,b,m,dt,xpos,k):
45      t=0
46      xpos=IntialXpos
47      TList=[]
48      VeloList=[]
49      XposList=[]
50      Vx=IntialVx
51      Vy=IntialVy
52      while t<tmax:
53          VeloList.append(Vx)
54          TList.append(t)
55          Vx=Vx-k*(c/m*np.sqrt(Vx**2 + Vy**2)*dt*Vx) -b/m*Vx*dt
56          Vy=Vy-g*dt-k*(c/m*np.sqrt(Vy**2 + Vx**2)*Vy*dt) -b/m*Vy*dt
57          XposList.append(xpos)
58          xpos=xpos+(Vx*dt)
59          t=t+dt
60      return XposList
61
```

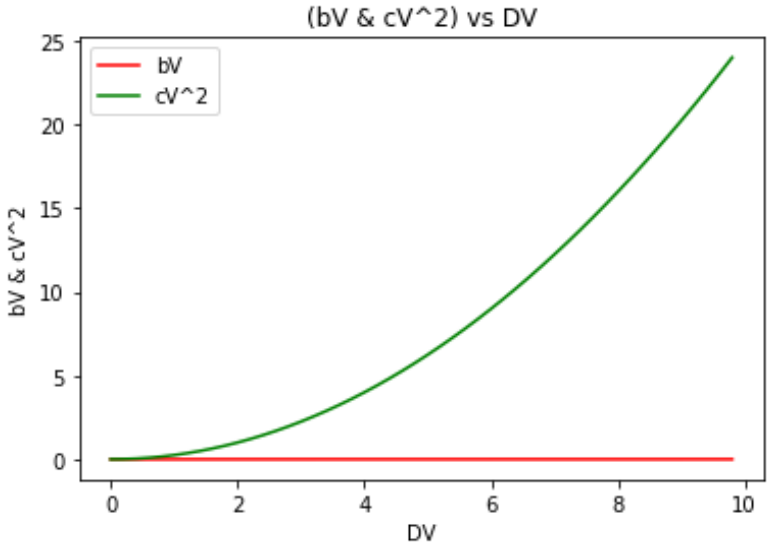
By using a random variable 'k' I can take bits out the functions by making k=0.

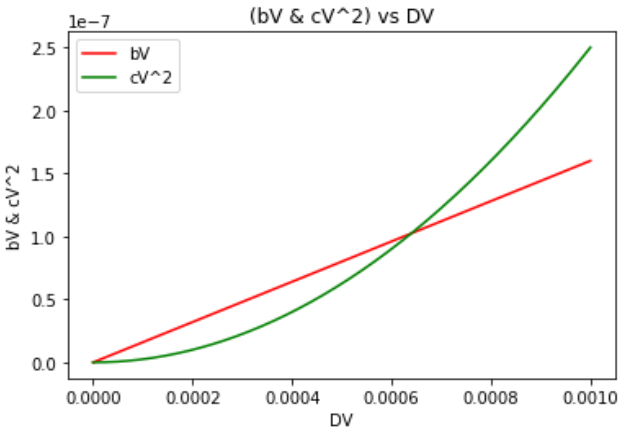
Doing this means I can get a graph of the quadratic dependence and the original dependence on the same graph.

```
98          t=t+dt
99      return XposList
100 plt.plot(XPosition2(0,1,1,dt,0),VertPosition2(0,1,1,dt,100),label="Old Way")
101 plt.plot(XPosition(0,1,1,dt,0,1),VertPosition(0,1,1,dt,100,1),label="With Air Resistance")
102 plt.plot(XPosition(0,0,1,dt,0,1),VertPosition(0,0,1,dt,100,1),label="In a Vacuum")
103 #plt.plot(XPosition2(0,0.5,1,dt,0,1),VertPosition2(0,0.5,1,dt,100,1),label="With Air Resistance")
104 plt.ylim(0,20)
105 plt.xlabel('X-Position')
106 plt.ylabel('Y-Position')
107 plt.legend(loc="upper right")
108 #plt.plot(XPosition2(0,0.5,1,dt,0,1),VertPosition2(0,0.5,1,dt,100,1),label="With Air Resistance")
109
```

Results:

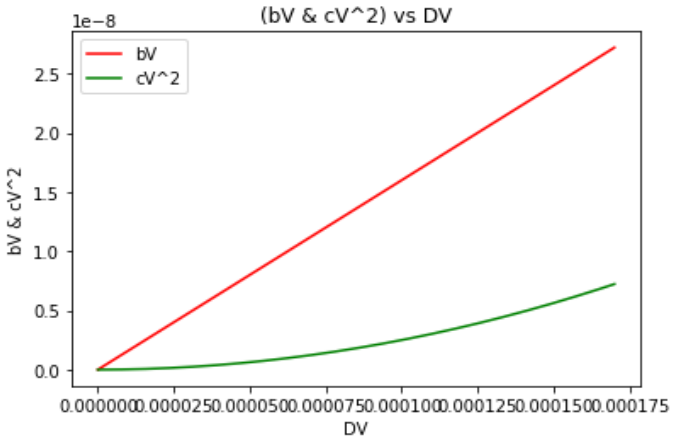
Results: Exercise 1





Rescaling:

$$cV^2 < bV \quad for \quad DV < 0.6 \cdot 10^{-4}$$



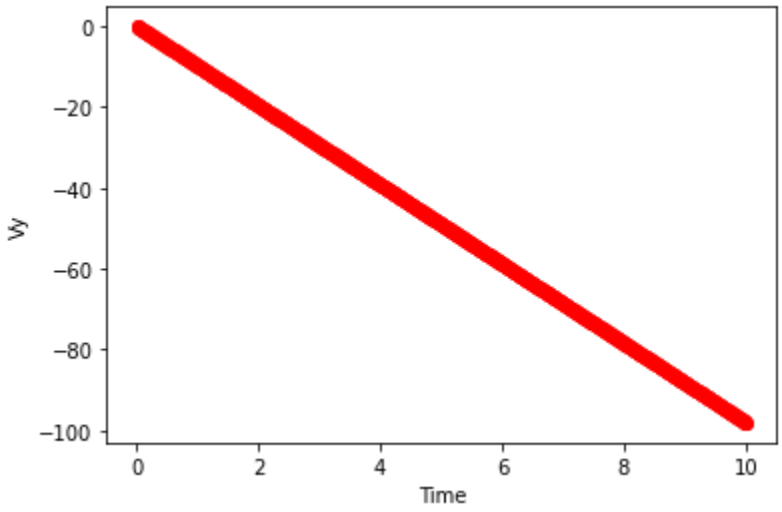
$$cV^2 << bV \quad for \quad DV < 0.175 \cdot 10^{-4}$$

Results: Exercise 2:

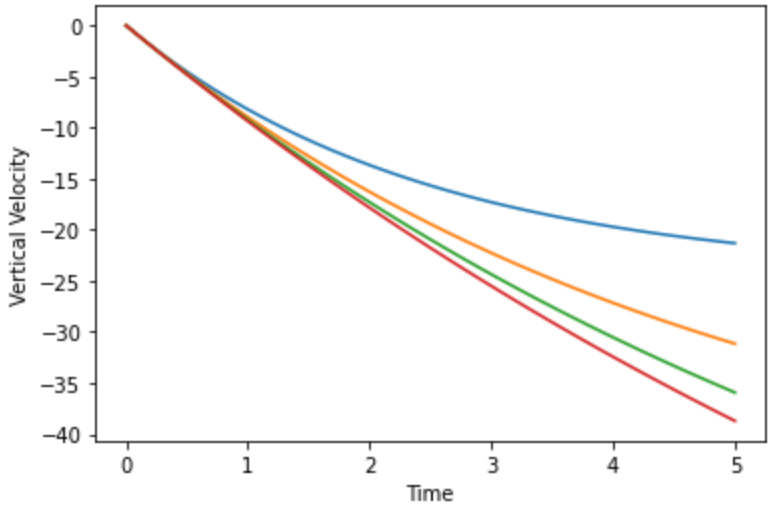
(ii)

This graph shows the progression of Vy against time. It shows a linear decrease from zero.

This is expected as there are no quadratic terms in our equation



We see the Vy for different masses change in slope. An increased mass increases the gravity component of the velocity, this is similar to decreasing the air resistance coefficient. Which we can see by comparing these lines with one with a decreased ‘b’ coefficient.



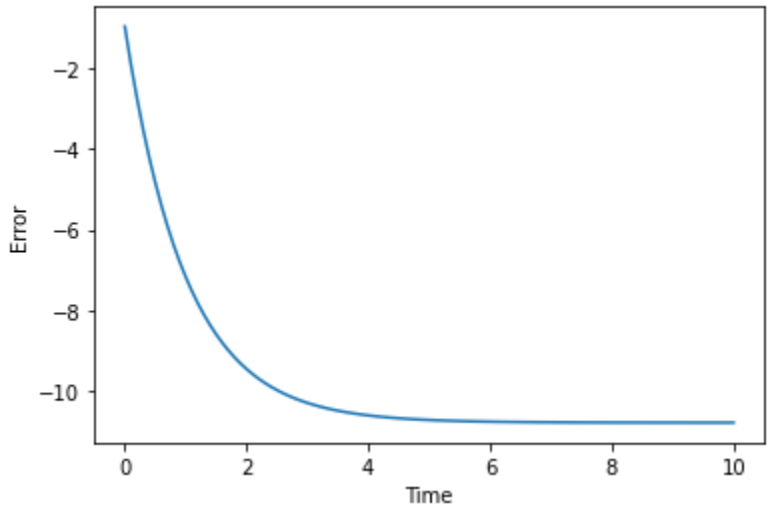
-Varying Masses under air resistance

- Masses progressively get lighter, blue being the lightest. We see the lighter masses will reach their terminal velocities quicker.

(iv)

Error Function: Vy(Numerical)-Vy(analytical)

dt=0.1

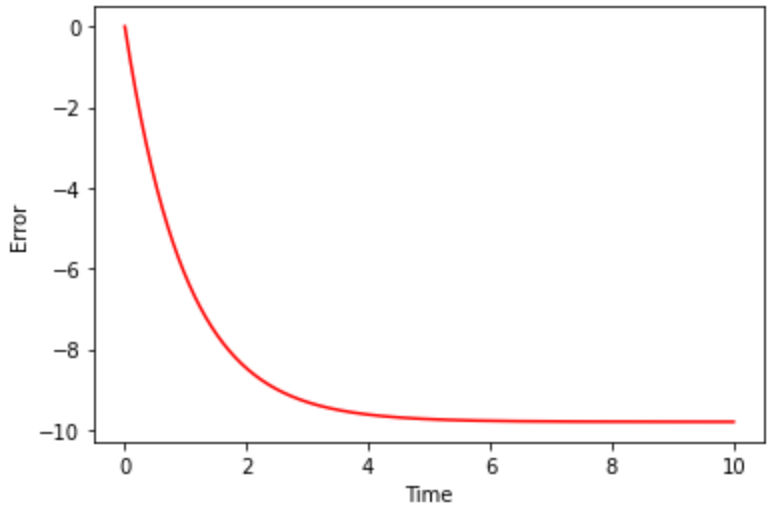


We see that as the time increase the error decreases, ie. Solutions converge.

To increase our accuracy for computation we can firstly increase the number of iterations, and to enter constants to a high degree of accuracy.

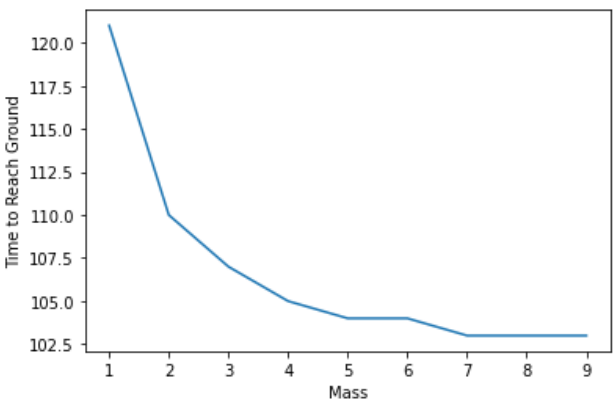
Also by changing the value of the timestep (dt) we see that the lower it becomes the more the methods diverge. This makes sense as there is a set amount of error that occurs with each iteration.

(Below) dt=0.0001



(v)
Time to reach Ground (y=0) vs Mass of Object

Mass of Object	Time Steps needed to reach ground
1	121
2	110
3	107
4	105
5	104
6	103
7	103
8	103
9	103



Mass vs Time to Reach Ground

-We see that the higher the mass the faster it falls. The approximation that all objections fall at the same rate regardless of mass is only accurate for heavier masses. As the mass

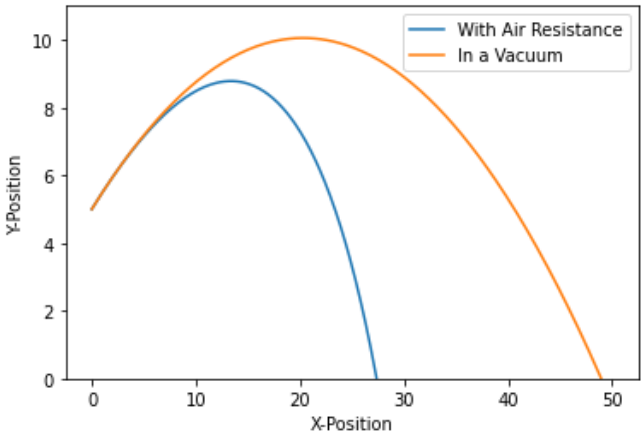
increases the time to reach the ground is more or less constant.

Results: Exercises 3

Trajectory of a Projectile in a Vacuum superimposed on one outside a vacuum:

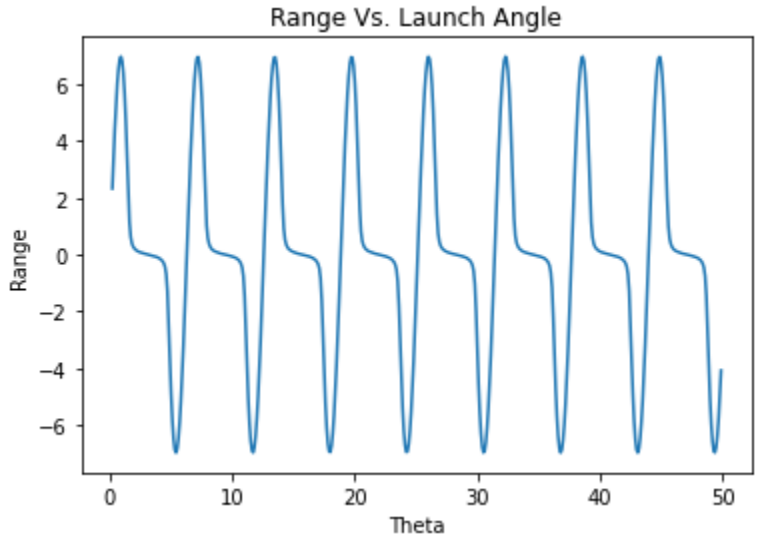
Orange: b=0

Blue: b=0.5



-As expected air resistance stunts the range of the projectile. Meaning the code is running correctly.

(ii)

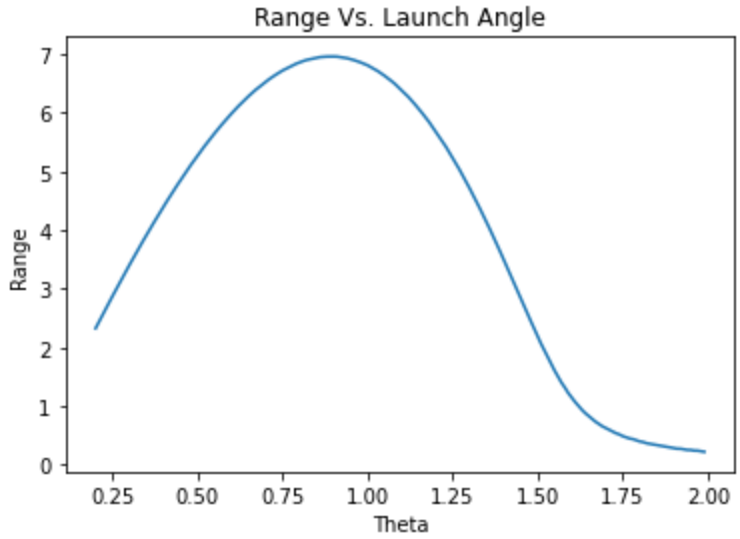


Launch Angle vs. Range

-The relationship is periodic, which makes sense as theta makes multiple fully rotations in this range

-The measurement for theta is in Radians

-By changing the axis limits the optimum angle is more clear.



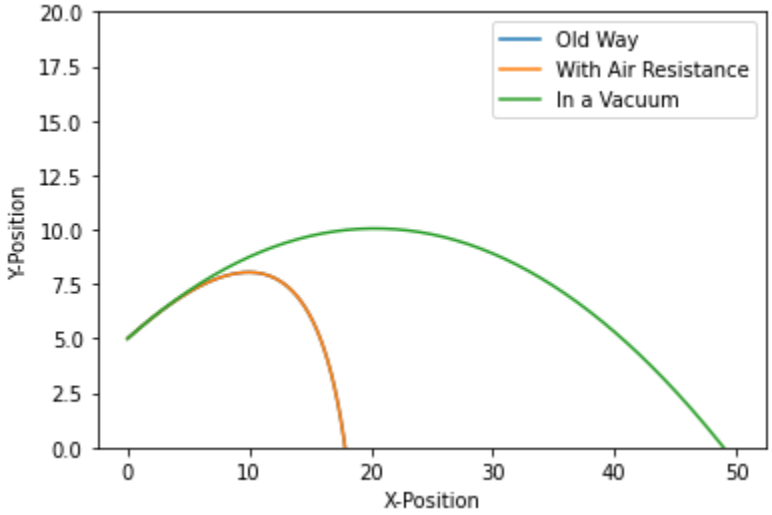
-With new axis we see that optimum launch angle≈ 0.8 ≈ π/4
-This confirms that 45 degrees is the optimum launch angle.

Exercise 4 :

Comparing all three simulations:

-My results for this section didn't work out as expected. The two methods showed no difference in path.

-This was due to an error in my code.



Conclusion

Most of the code ran as it was meant to.

For exercises 1-3, the relationships derived graphically were as expected.

An approximate range for simplification of the air resistance function was found: ($cV^2 << bV \quad for \quad DV < 0.175 \cdot 10^{-4}$). I found the relationship between the mass of the

object and the rate at which it falls in a non-vacuum.It was found that for heavier masses all fall at approx the same rate, yet the lighter the object the slower it falls. A graph comparing the path of a projectile in a vacuum and in an atmosphere was compared, it was observed that the path was shortened by air resistance. Using the graph of theta vs Projectile range, the optimum firing angle was found to be approx. 45 degree. Which is the accepted value. Whereas in Ex 4, I found no difference in the Quadratic relation and the Linear relation for air resistance which was unexpected.
In General the Lab was a success.

Code

Ex1

```
import matplotlib.pylab as plt
import numpy as np

#For a spherical
#projectile in air B = 1.6 x 10-4 N s/m2
and C = 0.25 N s2/m4

B=1.6*(10**-4)
C=0.25
# b = B D and c = C D2
#s f(V) = bV + c V2
#DV=k
k=np.arange(0,0.17*(10**-3),0.0000001)
def f(D,V):
    return B*k+C(k**2)


plt.plot(k,B*k,'red',label="bV")
plt.plot(k,C*(k)**2,'green',label="cV^2")
plt.xlabel('DV')
plt.ylabel('bV & cV^2')
plt.title('(bV & cV^2) vs DV')#
plt.legend(loc="upper left")
#B*k can be neglected for all values of DV
#Ck^2 can be ignored from 0-2
```

Ex2

```
import matplotlib.pylab as plt
import numpy as np

D=10**-4        #general constants
B=1.6*(10**-4)
C=0.25

c=(D**2)*C
b=B*D

dt=0.0001
g=9.8
tmax=10
IntialVy=0
print(c) #to make entering the fucntions easier
print(b)
def VertVelo(v,b,m,dt):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our velocities
    Vy=IntialVy #to give us the starting value
    while t<tmax :
        Velolist.append(Vy)
        Timelist.append(t)
        Vy=Vy-(g*dt+(b/m)*Vy*dt)
        t=t+dt
    plt.plot(Timelist,Velolist)
    plt.ylabel('Vertical Velocity')
    plt.xlabel('Time')
#VertVelo(0,1.6e-08,1,dt)

#to allow us to vary the mass
i=1 #It is easier to use whole numbers
while i<5:
    VertVelo(0, 0.4, i, dt) #a larger b value makes the effects
    # of air resistance clearer
    i=i+1
plt.show()

def errorfunc(Vy,b,m,dt):
    t=0
    TList=[] #change list name to avoid clashes
    Delta=[] #The difference between methods
    InitialVy=Vy
    while t<tmax :

Delta.append(((m*g/b)*(np.e**(-b*t/m)-1))-(g*dt+(b/m)*Vy*dt
))
        TList.append(t)
        t=t+dt

    plt.plot(TList,Delta,color='red')
    plt.xlabel('Time')
    plt.ylabel('Error')
errorfunc(IntialVy,1,1,dt)
plt.show()
```

ii)

```
import matplotlib.pylab as plt
import numpy as np

D=10**-4        #general constants
B=1.6*(10**-4)
C=0.25

c=(D**2)*C
b=B*D

dt=0.01
g=9.8
tmax=10
IntialVy=0
h=5
#print(c) #to make entering the fucntions
easier
#print(b)
def VertPosition(v,b,m,dt,height):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our velocities
    Height=[]
    Vy=IntialVy #to give us the starting
value
    height=h
    while height>0 :
        Velolist.append(Vy)
        Timelist.append(t)
        Height.append(height)
        Vy=Vy-(g*dt+(b/m)*Vy*dt)
        height=height+(Vy*dt)
        t=t+dt
        #print(height)


    print(len(Timelist))
    #plt.plot(Timelist,Velolist)
    #plt.ylabel('Vertical Velocity')
    #plt.xlabel('Time')
    return len(Timelist)
for i in range(1,10,1):
    VertPosition(0, 1, i, dt, h)
#VertVelo(0,1.6e-08,1,dt)
nsteps=[121,110,107,105,104,104,103,10
3,103]
Masses=[1,2,3,4,5,6,7,8,9]
plt.plot(Masses,nsteps)
plt.xlabel('Mass')
plt.ylabel('Time to Reach Ground')
plt.show()
```

Ex3

```
import matplotlib.pylab as plt
import numpy as np

D=10**-4        #general constants
B=1.6*(10**-4)
C=0.25

c=(D**2)*C
b=B*D

dt=0.01
g=9.8
tmax=2.5
IntialVy=10
IntialVx=20
IntialXpos=0
```

```
h=5

def VertPosition(v,b,m,dt,height):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our
velocities
    Height=[]
    Vy=IntialVy #to give us the starting
value
    height=h
    while t<tmax :
        Velolist.append(Vy)
        Timelist.append(t)
        Height.append(height)
        Vy=Vy-(g*dt+(b/m)*Vy*dt)
        height=height+(Vy*dt)
        t=t+dt
    return Height

def XPosition(v,b,m,dt,xpos):
    t=0
    xpos=IntialXpos
    TList=[]
    VeloList=[]
    XposList=[]
    Vx=IntialVx
    while t<tmax:
        VeloList.append(Vx)
        TList.append(t)
        Vx=Vx-((b/m)*Vx*dt)
        XposList.append(xpos)
        xpos=xpos+(Vx*dt)
        t=t+dt
    return XposList

plt.plot(XPosition(0,0.5,1,dt,0),VertPo
sition(0,0.5,1,dt,100),label="With Air
Resistance")
plt.plot(XPosition(0,0,1,dt,0),VertPosi
tion(0,0,1,dt,100),label="In a Va")
plt.ylim(0,11)
plt.xlabel('X-Position')
plt.ylabel('Y-Position')
plt.legend(loc="upper right")
```

ii)

```
import matplotlib.pylab as plt
import numpy as np

D=10**-4          #general constants
B=1.6*(10**-4)
C=0.25

c=(D**2)*C
b=B*D

dt=0.001
g=9.8
tmax=2.5
IntialVelo=10
theta=np.arange(0.2,2,0.01) #not
sure what units pyhton uses
IntialVy=10*(np.sin(theta))
IntialVx=10*(np.cos(theta))
IntialXpos=0
h=0.1 #from ground level
#keepinh Veloctiy fixed
#I need the velocity components as
functions of theta and
def XPosition(v,b,m,dt,xpos,theta):
    t=0
    xpos=IntialXpos
    TList=[]
    VeloList=[]
    XposList=[]
    Vx=10*(np.sin(theta))
    Vy=10*(np.cos(theta))
    height=h
    while t<tmax and height>0:
        VeloList.append(Vx)
        TList.append(t)
        Vx=Vx-((b/m)*Vx*dt)
        Vy=Vy-(g*dt+(b/m)*Vy*dt)
        XposList.append(xpos)
        height=height+(Vy*dt)
        xpos=xpos+(Vx*dt)
        t=t+dt
    return xpos

Ranges=[]
for i in theta:

Ranges.append(XPosition(10,0.5,1,d
t,0,i))
plt.plot(theta,Ranges)
plt.xlabel('Theta')
plt.ylabel('Range')
plt.title('Range Vs. Launch Angle')
```

Ex 4)

```
import matplotlib.pylab as plt
import numpy as np

D=10**-4          #general constants
B=1.6*(10**-4)
C=1

c=(D**2)*C
b=B*D

dt=0.01
g=9.8
tmax=2.5
IntialVy=10
IntialVx=20
IntialXpos=0
h=5

def VertPosition(v,b,m,dt,height,k):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our velocities
    Height=[]
    Vy=IntialVy #to give us the starting
value
    Vx=IntialVx
    height=h
    while t<tmax :
        Velolist.append(Vy)
        Timelist.append(t)
        Height.append(height)
        Vy=Vy-g*dt-k*(c/m*np.sqrt(Vy**2 +
Vx**2)*Vy*dt) -b/m*Vy*dt
        Vx=Vx-k*(c/m*np.sqrt(Vx**2 +
Vy**2)*dt*Vx) -b/m*Vx*dt
        height=height+(Vy*dt)
        t=t+dt
    return Height

def XPosition(v,b,m,dt,xpos,k):
    t=0
    xpos=IntialXpos
    TList=[]
    VeloList=[]
    XposList=[]
    Vx=IntialVx
    Vy=IntialVy
    while t<tmax:
        VeloList.append(Vx)
        TList.append(t)
        Vx=Vx-k*(c/m*np.sqrt(Vx**2 +
Vy**2)*dt*Vx) -b/m*Vx*dt
        Vy=Vy-g*dt-k*(c/m*np.sqrt(Vy**2 +
Vx**2)*Vy*dt) -b/m*Vy*dt
        XposList.append(xpos)
        xpos=xpos+(Vx*dt)
        t=t+dt
    return XposList


#######Old Method########


def VertPosition2(v,b,m,dt,height):
    t=0 #always starts from 0
    Timelist=[] #a range of time t<tmax
    Velolist=[] #same but for our velocities
    Height=[]
    Vy=IntialVy #to give us the starting
value
    Vx=IntialVx
    height=h
    while t<tmax :
        Velolist.append(Vy)
        Timelist.append(t)
        Height.append(height)
        Vy=Vy-g*dt-b*Vy*dt/m
        height=height+(Vy*dt)
        t=t+dt
    return Height

def XPosition2(v,b,m,dt,xpos):
    t=0
    xpos=IntialXpos
    TList=[]
    VeloList=[]
    XposList=[]
    Vx=IntialVx
    Vy=IntialVy

plt.plot(XPosition2(0,1,1,dt,0),VertPosition
2(0,1,1,dt,100),label="Old Way")
```

```
plt.plot(XPosition(0,1,1,dt,0,1),VertPosition(0,1,1,dt,100,1),label="With Air Resistance")
plt.plot(XPosition(0,0,1,dt,0,1),VertPosition(0,0,1,dt,100,1),label="In a Vacuum")
#plt.plot(XPosition2(0,0.5,1,dt,0,1),VertPosition2(0,0.5,1,dt,100,1),label="With Air Resistance")
plt.ylim(0,20)
plt.xlabel('X-Position')
plt.ylabel('Y-Position')
plt.legend(loc="upper right")
#plt.plot(XPosition2(0,0.5,1,dt,0,1),VertPosition2(0,0.5,1,dt,100,1),label="With Air Resistance")
```

References:

[1] Tcd Lab Manual: Comp Lab 3 : Projectile Motion

[2] Computational Physics with Python

[3]https://scipython.com/book2/chapter-8-scipy/examples/a-projectile-with-air-resistance/

[4]https://chartio.com/learn