Lab 3: Projectile Motion

Valentina Watanabe

March 2024

1 Introduction

The aim of this Python lab was to simulate projectile motion and the importance of accounting for air resistance. The constructed code analysed the quadratic and linear components of air resistance and their effects on the motion of the particular launched particles.

1.1 Theory

The force of air resistance can be represented using the equation,

$$F = -f(V)u.$$

This force depends on the velocity of the launched object and the u vector, which is the unit vector that follows the velocity direction. The minus sign indicates its opposition to the direction of velocity.

Function f(v) can be approximated as,

$$f(v) = b * B + c * V^2.$$

Constant coefficients b and c depend on the shape and size of the particle, and spherical objects can be written as b = BD and c = C D^2. Coefficients C and D depend on the medium, and for this experiment, we are using B = 1.6 x $10^{-4} N * s/m^2$ and C = 0.25 $N * s^2/m^4$. D is the diameter of the object.

2 Methodology

2.1 Exercise 1

For certain ranges of velocity and diameter, the linear or quadratic term of the air resistance equation can be neglected. To find these range values, a plot was constructed of these terms as functions of D x V. Three functions were constructed: one for each term, and one for the sum of both terms.

$$linearf(DV) = b * DV$$

$$quadraticf(DV) = c * (DV)^2$$

$$force(DV) = linearf(DV) + quadraticf(DV)$$

The ideal form of the force value was found for a baseball of diameter D=7cm with a speed of V=5m/s, a tiny drop of oil with D=1.5 x 10-6m and V= 5 x 10-5 m/s, and a raindrop of diameter D=1mm travelling at a speed of V=1 m/s.

2.2 Exercise 2

The case studied for this exercise was a spherical grain of dust of mass density of 2x103 kg/m^3 and diameter D = 10^{-4} m that is released from rest. In this case, we were able to neglect the quadratic term of the air resistance and use solely the linear one to approximate it. We were allowed then to use Newton's equation

$$\Delta V_y = -g\Delta t - \frac{b}{m}V_y\Delta T.$$

To graph the values of the vertical velocity with time for various masses, a code was created using the mentioned equation. A loop was used that solved for the v and t values using the velocity equation using inputs from the user for the values of b, g, mass, and time interval. Each t and v value were appended to an empty list outside the loop.

The analytical solution is given by the equation,

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

Comparing this with our solution, a graph of error over time was made. To do so, the same code is used but using the appropriate analytical equation.

The final part of this exercise consisted of investigating the variation of the position of the object with time. The grain was released from a height of 5 m and the time it took to reach the ground was calculated. The previous code is modified so that instead of graphing Velocity vs Time it graphs the height of the particle from the ground. In each loop, velocity is calculated for a certain time. The distance is calculated for that time loop, and is accumulated in a variable Y.

2.3 Exercise 3

We study the trajectory when air resistance is no longer neglected. Using Newton's 2nd law, we get horizontal and vertical coordinates to be,

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

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

We reuse the constructed code to separate both coordinates, considering gravity only in the vertical one. They are incorporated similarly as in the previous

exercise, multiplying by time to have distance solutions. The trajectories in a vacuum and with air resistance are plotted and compared.

The objective was then to find the launching angle that leads to maximum horizontal displacement when considering air resistance.

2.4 Exercise 4

Setting b=0, the air resistance turns to depend quadratically on the velocity. In this case, necessary equations turn out to be

$$\frac{dV_x}{dt} = -\frac{c}{m}\sqrt{{V_x}^2 + {V_y}^2}V_x$$

$$\frac{dV_x}{dt} = -g - \frac{c}{m}\sqrt{{V_x}^2 + {V_y}^2}V_y.$$

These were incorporated into the code and then the plot of height with distance is constructed.

3 Results

3.1 Exercise 1

Considering the graphs, taking a 3% incidence to depreciate the term works. The quadratic term can be neglected for D*V lower than 7.1 x $10^{-5}m^2/s$, and the linear term for D*V higher than 5.8 x $10^{-3}m^2/s$. We can observe so in Figure 1. In Figure 2 we can see their relative relationship with DV.

For the case of the baseball, the total air resistance was found to be $0.0307m^2/s$, with the linear function being able to be neglected as its percentage is 0.183%, while the quadratic function takes the 99.9%. The drop of oil had 1.2 x $10^{-14}m^2/s$, with the linear function consisting of the 99.9% and quadratic part of 1.17 x 10^{-5}%. For the raindrop, the total force was 4.1 x $10^{-7}m^2/s$, with being unable to neglect either function as the linear part is 39.02% and the quadratic 0.98%. Figure 3 shows the results.

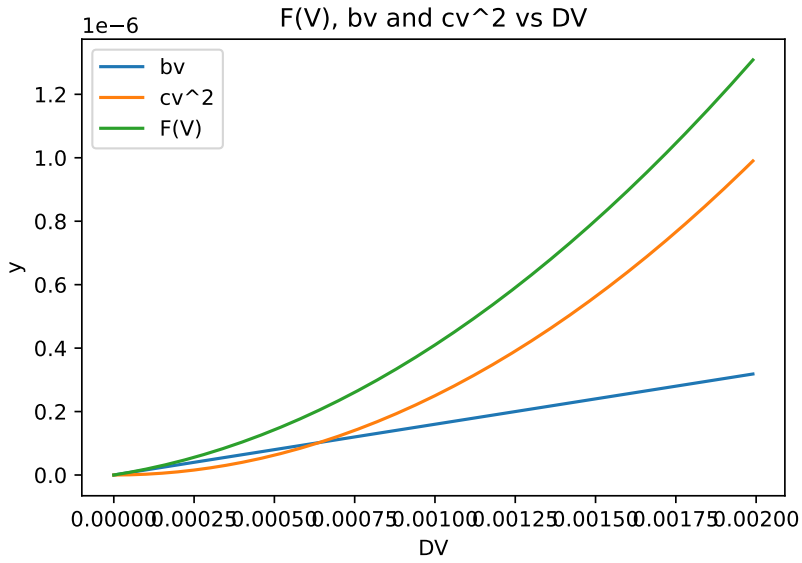

Figure 1: bv and cv^2 vs. DV

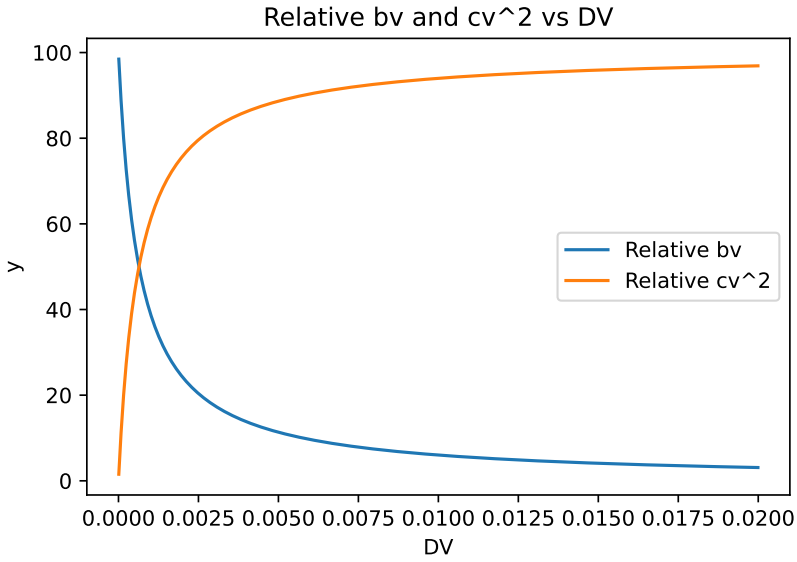

Figure 2: Relative bv and cv^2 vs. DV

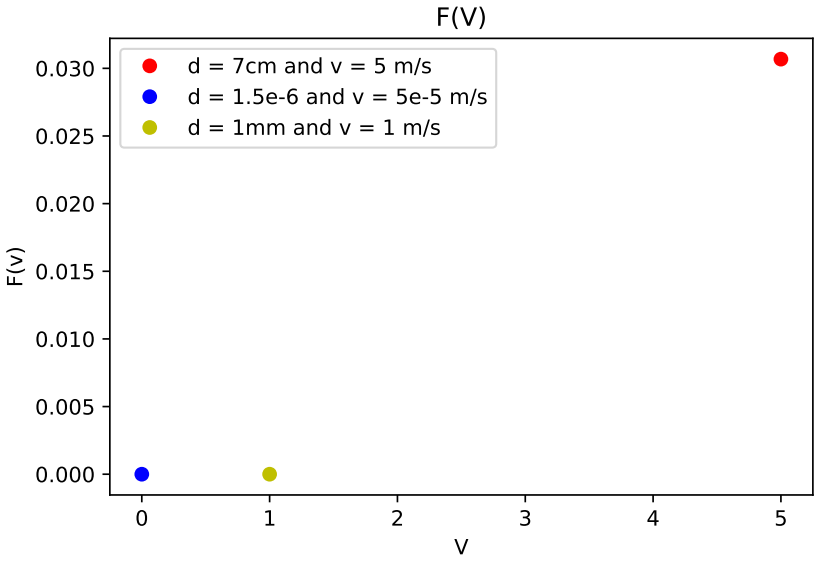

Figure 3: Air resistance for each case.

3.2 Exercise 2

Using the density and diameter, we can calculate the mass to be 1.047 x 10^{-9} kg. The value of the D x V_T product for part A was calculated to be 5.8803421194891694 x $10^{-5} m^3/s$. Using the ranges from previous exercises, we can see that this value is lower than 7.1 x 10^{-5}, which means that we can disregard the quadratic term taking an error to less than 10%.

To investigate how the velocity increases with mass, a code was constructed that obtains the vertical velocity of a spherical object concerning time as it's released from rest. Different masses were used to plot this, as seen in Figure 4. We can see that the larger the mass, the longer it takes to reach its terminal velocity. It also indicates lower air resistance.

Figure 5 shows the difference between the value previously calculated and the analytical one, and Figure 6 shows the error.

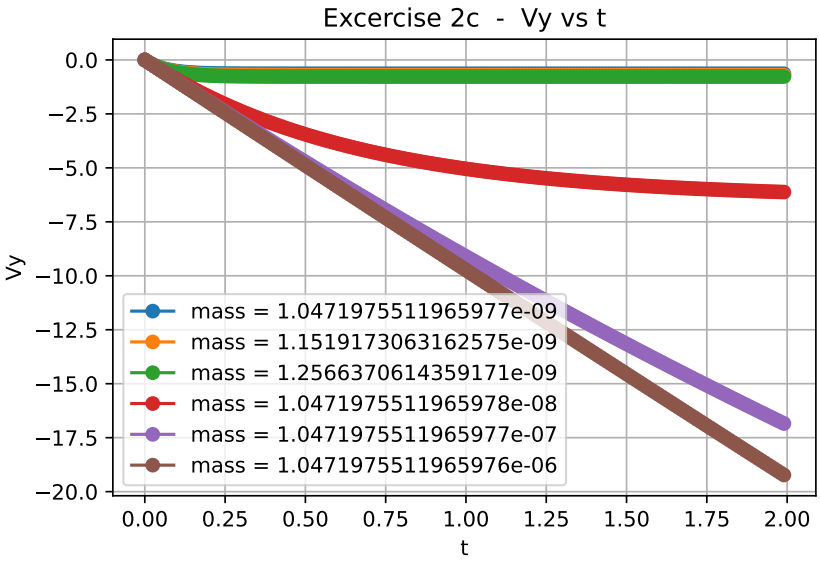

Figure 4: Vy vs. t for various masses

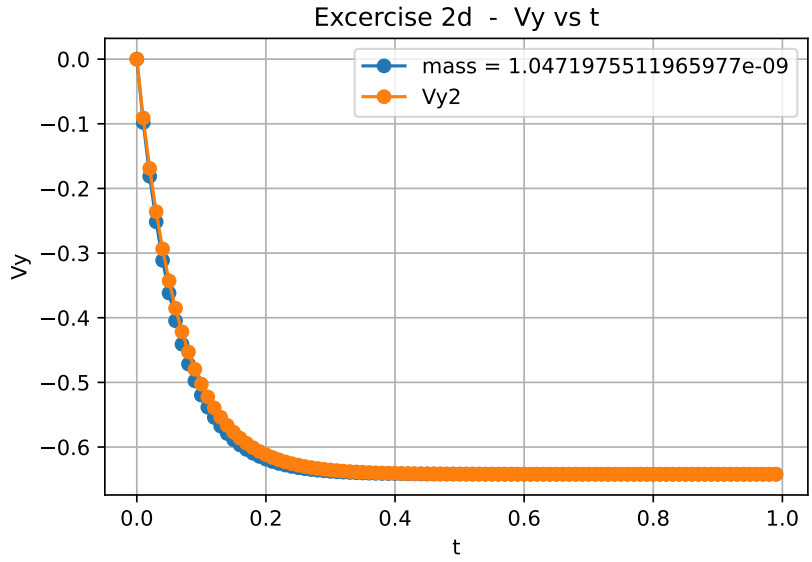

Figure 5: Vy vs. t

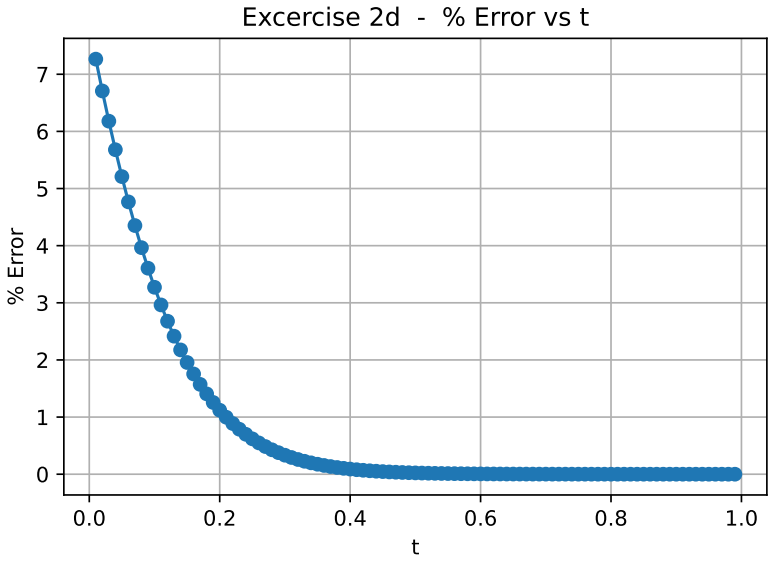

Figure 6: Error Vy vs. t

Figure 7 is a plot that shows the time with a variation of masses. We can see that the lower the mass, the higher the time to reach ground. Figure 8 is height as a function of time, which shows the higher the distance, the less time it takes to reach the ground. We can see that all objects don't fall with the same acceleration as if this were the case, the time would not change.

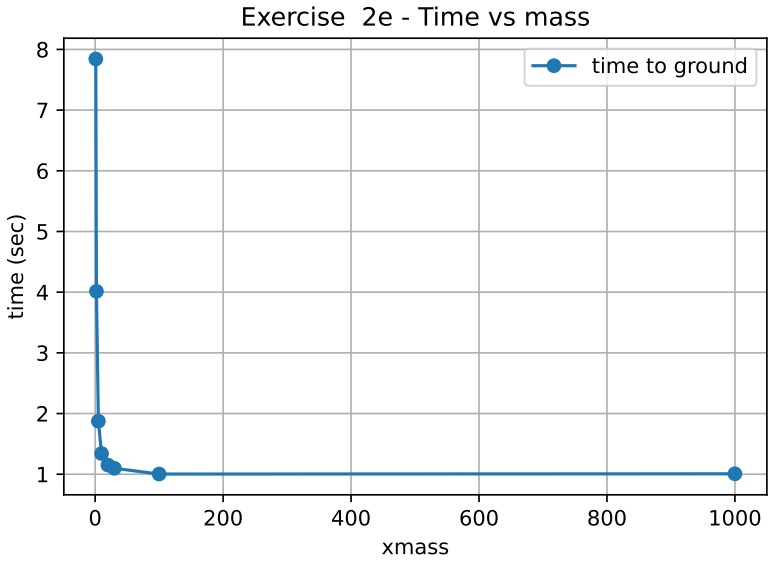

Figure 7: Time vs. xmass

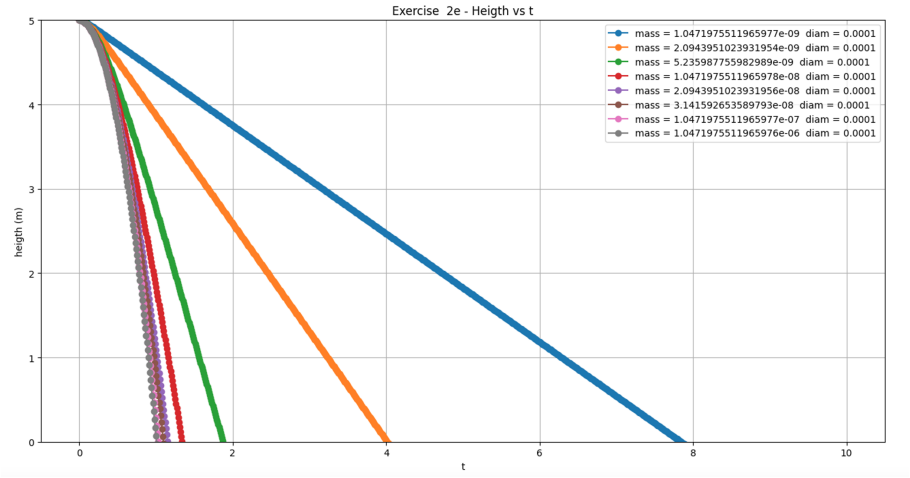

Figure 8: Height vs. t

3.3 Exercise 3

In the following graphs, the orange line shows the no air resistance path and the blue one is the one considering it. The smaller the mass, the greater the difference there is in the projectile motion between with and without air resistance.

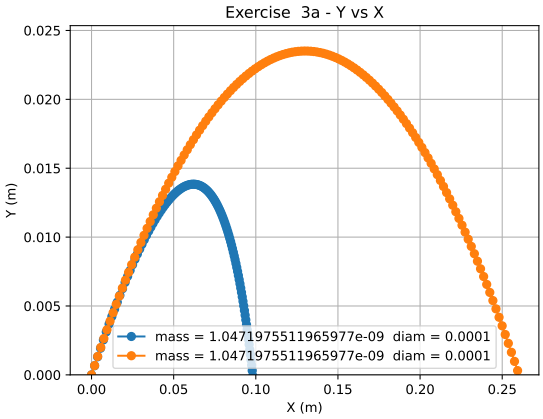

Figure 9: Y vs X

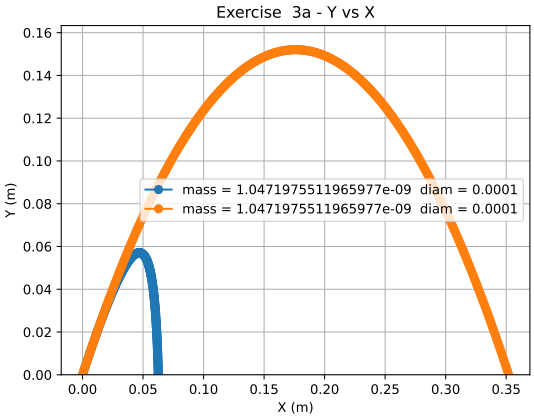

Figure 10: Y vs X

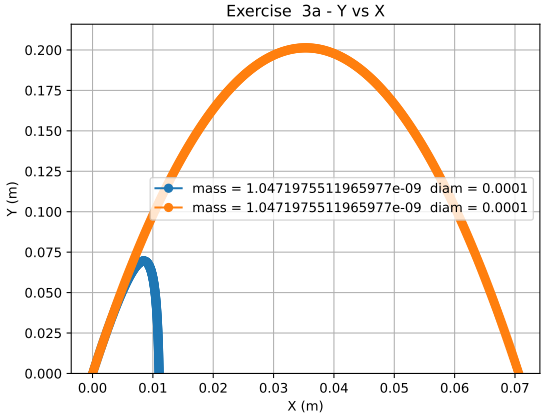

Figure 11: Y vs. X

As is the case without air resistance, the most optimum angle is 45°. But as we can see from the formula, the angle is dependent on mass. Figure 12 shows this relationship. Figure 13 shows the trajectory of each pertinent mass.

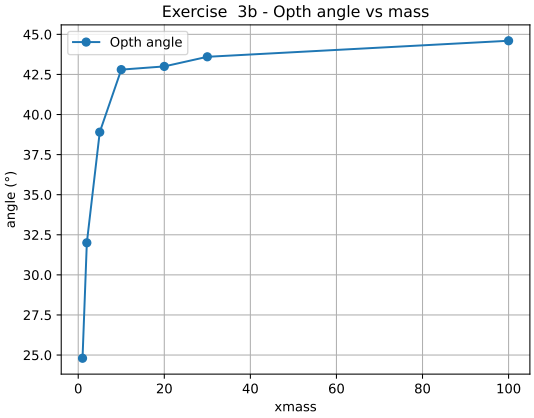

Figure 12: Angle vs. xmass

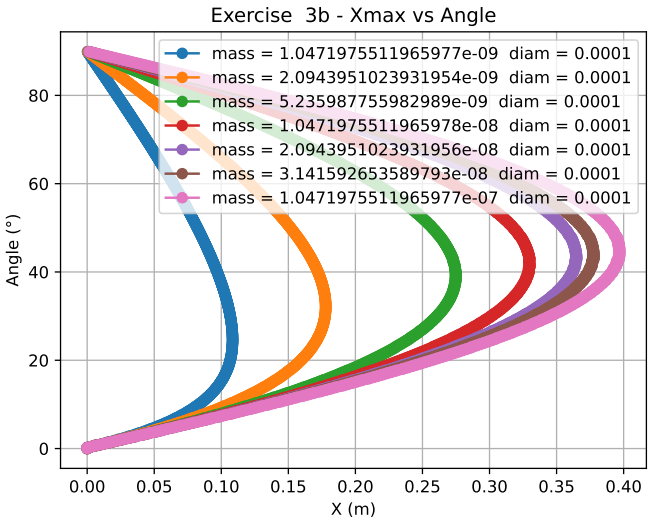

Figure 13: Angle vs. xmass

3.4 Exercise 4

The different trajectories were plotted taking into account the quadratic dependence on air resistance. Halving the launching angle decreases dramatically the distance reached. We can also see that the linear term has a stronger effect than the quadratic term. The plot shows again that air resistance reduces the height reached when compared to a vacuum.

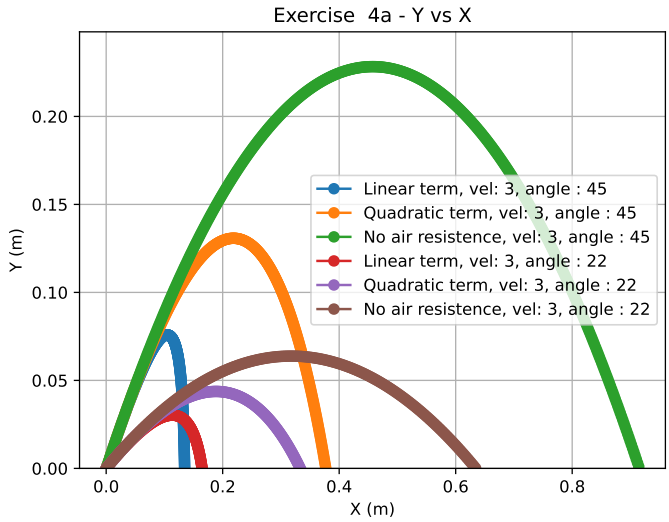

Figure 14: Y vs X

4 Conclusion

We utilized Python simulations to delve into the dynamics of projectile motion, particularly focusing on the influence of air resistance. Both the linear and quadratic components of air resistance were analyzed, with consideration given to their effects on the trajectory of launched particles. The findings revealed that neglecting certain components of air resistance could be done within defined velocity and diameter ranges with high accuracy. Experiments also demonstrated that all objects do not fall with the same acceleration, contrary to the theoretical statement, as seen by the varying time taken to reach the ground for objects of differing masses. The optimal angle was also experimented with various masses, concluding that displacement is greater at the same as motion without air resistance, which is 45°. Finally, the quadratic dependence was studied. It was observed that this term had the greatest effect on motion when air resistance was present. In conclusion, an analysis was made of the effect of air resistance on projectile motion.

5 Appendix

Exercise 1:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import numpy as np
import matplotlib.pyplot as plt

B = 1.6e-4
C = 0.25

DV = np.arange(0,0.02,0.00001)

def linearf(DV):
    bv = B * DV
    return bv

def quadraticf(DV):
    cv2 = C * (DV)**2
    return cv2

plt.plot(DV, linearf(DV), label = 'bv')
plt.plot(DV, quadraticf(DV), label = 'cv^2')
plt.plot(DV, linearf(DV)+quadraticf(DV), label = 'F(V)')
plt.title('F(V), bv and cv^2 vs DV')
plt.xlabel('DV')
plt.ylabel('y')
plt.legend()
plt.savefig('Lab 3 Ex1.1.pdf')
plt.show()

relativlin = 100*(linearf(DV)/(linearf(DV)+quadraticf(DV)))
relativqua = 100*(quadraticf(DV)/(linearf(DV)+quadraticf(DV)))

plt.plot(DV, relativlin, label = 'Relative bv')
plt.plot(DV, relativqua, label = 'Relative cv^2')
plt.title('Relative bv and cv^2 vs DV')
plt.xlabel('DV')
plt.ylabel('y')
plt.legend()
plt.savefig('Lab 3 Ex1.2.pdf')
```

```
plt.show()

def force(DV):
    f = linearf(DV) + quadraticf(DV)
    return f

V1=5
y = force(0.07*5)
print(y)

V2=5e-5
y1=force(1.5e-6*5e-5)
print(y1)

V3=1
y2=force(0.001*1)
print(y2)
plt.plot(V1,y, 'ro', label = 'd = 7cm and v = 5 m/s')
plt.plot(V2,y1, 'bo', label = 'd = 1.5e-6 and v = 5e-5 m/s')
plt.plot(V3,y2, 'yo', label = 'd = 1mm and v = 1 m/s')
plt.title('F(V)')
plt.xlabel('V')
plt.ylabel('F(v)')
plt.legend()
plt.savefig('Lab 3 Ex1.3.pdf')
plt.show()

Exercise 2:
Part A:
"""
Created on Sun Mar 31 18:58:44 2024

@author: vwitch
"""
import math

#Weigth force F = m*g
#constants
B=1.6e-4
C=0.25
g=9.81

#data
dens = 2000   #kg/m3
Diam =  1e-4    #m
```

```
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg

#calculate de weight force
F = mass * 9.81  #Weigth force equals to air resistence at terminal velocity

# At terminal velocity
###   C*(DV)^2  +  B*(DV) = m*g
###   C*(DV)^2  +  B*(DV) - mg = 0
###   a(x)^2  +  b*(x) + c = 0
###   x1, x2 roots  -->     a=C, b=B, c=-mg
###   x1  =   (-b + SQR(b2 - 4ac ) /2a

a=C
b=B
c=-F

DVlimn = (-b + math.sqrt(b**2 - 4*a*c))/(2*a)


print("Limits to depreciate cuadratic term  from Ex.1")
print("Cuadratic term neglected for D*V lower than:   7.1E-5")
print("Linear term neglected for D*V higher than:   5.8E-3")
print()
print("DV lim = ", DVlimn, " lower than 7.1E-5. Cuadratic term neglected.")

Part B:
@author: vwitch
"""
import matplotlib.pyplot as plt

mass = float(input("mass = "))
g = float(input("g = "))
b =float(input("b = "))

DV=0
Vy = 0
t=0
# Lists to plot  Vy and t
vy_values = []
t_values = []

while t <= 2:    #from 0 to 2 seconds
    vy_values.append(Vy)
    t_values.append(t)
    # Steps =  0.01 seconds
```

```
    DT = 0.01
    DV = -g * DT - (b / mass) * Vy * DT
    Vy = Vy + DV
    t = t + DT

# Plot Vy vs t
plt.plot(t_values, vy_values, marker='o', label='mass = ' + str(mass))
plt.xlabel('t')
plt.ylabel('Vy')
plt.title('Excercise 2b  -  Vy vs t')
plt.legend()
plt.grid(True)
plt.show()

Part C:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
Created on Sun Mar 31 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt


def calc_vy(B, Diam, g, mass):
    b = B*Diam

    DT = 0
    DV=0
    Vy = 0
    t=0

    # Lists to plot  Vy and t
    vy_values = []
    t_values = []

    while t <= 2:    #from 0 to 2 seconds
        vy_values.append(Vy)
        t_values.append(t)
        # Steps =  0.01 seconds
        DT = 0.01
        DV = -g * DT - (b / mass) * Vy * DT
        Vy = Vy + DV
```

```
        t = t + DT

    # Plot Vy vs t
    plt.plot(t_values, vy_values, marker='o', label='mass = ' + str(mass) + '  diam = ' + st
    plt.xlabel('t')
    plt.ylabel('Vy')
    plt.title('Excercise 2c  -  Vy vs t')
    plt.legend()
    plt.grid(True)
plt.show()


########################################################################
#g, mass previusly defined
#Quadratic term depreciated

#constants
B=1.6e-4

g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg

calc_vy(B, Diam, g, mass)
calc_vy(B, Diam, g, mass*1.1)
calc_vy(B, Diam, g, mass*1.2)
calc_vy(B, Diam, g, mass*10)
calc_vy(B, Diam, g, mass*100)
calc_vy(B, Diam, g, mass*1000)

Part C:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt

#constants
```

```
B=1.6e-4

g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg
b = B*Diam

DT = 0
DV=0
Vy = 0
t=0

# Lists to plot  Vy and t
vy_values = []
t_values = []
vy2_values = []

while t <= 1:    #from 0 to 2 seconds
    Vy2 = mass*g/b*(math.exp(-b*t/mass)-1)
    vy2_values.append(Vy2)

    vy_values.append(Vy)
    t_values.append(t)
    # Steps =  0.01 seconds
    DT = 0.01
    DV = -g * DT - (b / mass) * Vy * DT
    Vy = Vy + DV
    t = t + DT


Error_values = []
for vy2, vy1 in zip(vy2_values, vy_values):
    if vy1 == 0:
        Error_values.append(float('inf'))  # Handle division by zero
    else:
        Error_values.append((vy1 - vy2) / vy1 * 100)


# Plot Vy vs t
plt.plot(t_values, vy_values, marker='o',
label='mass = ' + str(mass))
plt.plot(t_values, vy2_values, marker='o', label='Vy2')
```

```
plt.xlabel('t')
plt.ylabel('Vy')
plt.title('Excercise 2d  -  Vy vs t')
plt.legend()
plt.grid(True)
plt.savefig('Lab 3 2.4a.pdf')
plt.show()

# Plot Vy vs t
plt.figure()
plt.plot(t_values, Error_values, marker='o', label='% Error')
plt.xlabel('t')
plt.ylabel('% Error')
plt.title('Excercise 2d  -  % Error vs t')
plt.grid(True)
plt.savefig('Lab 3 2.4.pdf')
plt.show()

Part D:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt

#constants
B=1.6e-4

g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg
b = B*Diam

DT = 0
DV=0
Vy = 0
t=0
```

```
# Lists to plot  Vy and t
vy_values = []
t_values = []
vy2_values = []

while t <= 1:    #from 0 to 2 seconds
    Vy2 = mass*g/b*(math.exp(-b*t/mass)-1)
    vy2_values.append(Vy2)

    vy_values.append(Vy)
    t_values.append(t)
    # Steps =  0.01 seconds
    DT = 0.01
    DV = -g * DT - (b / mass) * Vy * DT
    Vy = Vy + DV
    t = t + DT


Error_values = []
for vy2, vy1 in zip(vy2_values, vy_values):
    if vy1 == 0:
        Error_values.append(float('inf'))  # Handle division by zero
    else:
        Error_values.append((vy1 - vy2) / vy1 * 100)


# Plot Vy vs t
plt.plot(t_values, vy_values, marker='o', label='mass = '
+ str(mass) + '  diam = ' + str(Diam))
plt.plot(t_values, vy2_values, marker='o', label='Vy2')

plt.xlabel('t')
plt.ylabel('Vy')
plt.title('Excercise 2d  -  Vy vs t')
plt.legend()
plt.grid(True)
plt.show()

# Plot Vy vs t
plt.figure()
plt.plot(t_values, Error_values, marker='o', label='% Error')
plt.xlabel('t')
plt.ylabel('% Error')
plt.title('Excercise 2d  -  % Error vs t')
plt.grid(True)
```

```
plt.show()

Part E:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt

def calc_H(B, Diam, g, mass):
    b = B*Diam

    H = 5   #5 meters

    DT = 0
    DV=0
    Vy = 0
    t=0
    Hy=H
    Y=0

    # Lists to plot  Vy and t
    vy_values = []
    t_values = []
    Y_values = []
    H_values = []


    while t <= 10:    #from 0 to 2 seconds
        Y_values.append(Y)
        H_values.append(Hy)
        t_values.append(t)
        vy_values.append(Vy)

        # Steps =  0.01 seconds
        DT = 0.01
        DV = -g * DT - (b / mass) * Vy * DT
        Vy = Vy + DV

        DY = Vy * DT
        Y = Y + DY
        Hy = H + Y
```

```
        t = t + DT


    # Plot Vy vs t
    plt.plot(t_values, H_values, marker='o',
    label='mass = ' + str(mass) + '  diam = ' + str(Diam))
    plt.xlabel('t')
    plt.ylabel('heigth (m)')
    plt.title('Exercise  2e - Heigth vs t')
    plt.legend()
    plt.grid(True)
    # Establecer límites del eje y entre 0 y 5
    plt.ylim(0, 5)
    plt.show()


########################################################################
#g, mass previusly defined
#Quadratic term depreciated

#constants
B=1.6e-4

g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg
calc_H(B, Diam, g, mass)
calc_H(B, Diam, g, mass*2)
calc_H(B, Diam, g, mass*5)
calc_H(B, Diam, g, mass*10)
calc_H(B, Diam, g, mass*20)
calc_H(B, Diam, g, mass*30)
calc_H(B, Diam, g, mass*100)
calc_H(B, Diam, g, mass*1000)

##Plot time vs mass
##From the previus plot

time_to_ground =[7.84282, 4.0142, 1.8744, 1.3411, 1.1507, 1.0979, 1.00317, 1.0071]
mass_x =[1, 2, 5, 10, 20, 30, 100, 1000]

plt.figure()
```

```
plt.plot(mass_x, time_to_ground, marker='o', label='time to ground')
plt.xlabel('xmass')
plt.ylabel('time (sec)')
plt.title('Exercise  2e - Time vs mass')
plt.legend()
plt.grid(True)
# Establecer límites del eje y entre 0 y 5
plt.savefig('Lab 3 2e.pdf')
plt.show()

Exercise 3:
Part A:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt

def calc_XYT(B, Diam, g, mass, Vt, angle):

    Vxi = Vt * math.cos (math.radians(angle))
    Vyi = Vt * math.sin (math.radians(angle))

    b = B*Diam

    DT = 0
    t=0

    #For air resistence calculation
    Vy = Vyi   #Initital Speed
    Vx = Vxi   #Initital Speed
    DVy=0
    Y=0
    DVx=0
    X=0

    #Without air resistence calculation
    Vy_noair = Vyi   #Initital Speed
    Vx_noair = Vxi   #Initital Speed
    DVy_noair=0
    Y_noair=0
    DVx_noair=0
```

```
X_noair=0

# Lists to plot Vy, Vx, Y, X and t

#For air resistence calculation
t_values = []
Y_values = []
X_values = []

#Without air resistence calculation
Y_values_noair = []
X_values_noair = []


while Y_noair >= 0:

    t_values.append(t)
    # Steps =  0.01 seconds
    DT = 0.001

    #For air resistence calculation
    Y_values.append(Y)
    X_values.append(X)

    DVy = -g * DT - (b / mass) * Vy * DT
    Vy = Vy + DVy
    DVx = - (b / mass) * Vx * DT
    Vx = Vx + DVx

    DY = Vy * DT
    Y = Y + DY
    DX = Vx * DT
    X = X + DX

    #Without air resistence calculation
    Y_values_noair.append(Y_noair)
    X_values_noair.append(X_noair)

    DVy_noair = -g * DT
    Vy_noair = Vy_noair + DVy_noair
    DVx_noair = 0
    Vx_noair = Vx_noair + DVx_noair

    DY_noair = Vy_noair * DT
    Y_noair = Y_noair + DY_noair
    DX_noair = Vx_noair * DT
```

```
        X_noair = X_noair + DX_noair

        t = t + DT

    # Plot Vy vs t
    plt.figure()
    plt.plot(X_values, Y_values, marker='o', label='mass = ' + str(mass) + '  diam = ' + str
    plt.plot(X_values_noair, Y_values_noair, marker='o', label='mass = ' + str(mass) + '  di
    plt.xlabel('X (m)')
    plt.ylabel('Y (m)')
    plt.title('Exercise  3a - Y vs X')
    plt.legend()
    plt.grid(True)

    # Establecer límites del eje y entre 0 y ymaximo default
    plt.ylim(0)  #establece el minimo en Y en cero y deja el máximo en valor default

    plt.show()



########################################################################
#g, mass previusly defined
#Quadratic term depreciated

#constants
B=1.6e-4
g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg

#Initital Speed, angle
Vt = 2

calc_XYT(B, Diam, g, mass, Vt, 85)

calc_XYT(B, Diam, g, mass, Vt, 60)

calc_XYT(B, Diam, g, mass, Vt, 20)

Part B:
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt


def calc_XYT(B, Diam, g, mass, Vt, angle):

    Vxi = Vt * math.cos (math.radians(angle))
    Vyi = Vt * math.sin (math.radians(angle))

    b = B*Diam

    DT = 0
    t=0

    #For air resistence calculation
    Vy = Vyi   #Initital Speed
    Vx = Vxi   #Initital Speed
    DVy=0
    Y=0
    DVx=0
    X=0

    #Without air resistence calculation
    Vy_noair = Vyi   #Initital Speed
    Vx_noair = Vxi   #Initital Speed
    DVy_noair=0
    Y_noair=0
    DVx_noair=0
    X_noair=0

    # Lists to plot Vy, Vx, Y, X and t

    #For air resistence calculation
    t_values = []
    Y_values = []
    X_values = []

    #Without air resistence calculation
    Y_values_noair = []
```

```
    X_values_noair = []


    while Y_noair >= 0:

        t_values.append(t)
        # Steps =  0.01 seconds
        DT = 0.001

        #For air resistence calculation
        Y_values.append(Y)
        X_values.append(X)

        DVy = -g * DT - (b / mass) * Vy * DT
        Vy = Vy + DVy
        DVx = - (b / mass) * Vx * DT
        Vx = Vx + DVx

        DY = Vy * DT
        Y = Y + DY
        DX = Vx * DT
        X = X + DX

        #Without air resistence calculation
        Y_values_noair.append(Y_noair)
        X_values_noair.append(X_noair)

        DVy_noair = -g * DT
        Vy_noair = Vy_noair + DVy_noair
        DVx_noair = 0
        Vx_noair = Vx_noair + DVx_noair

        DY_noair = Vy_noair * DT
        Y_noair = Y_noair + DY_noair
        DX_noair = Vx_noair * DT
        X_noair = X_noair + DX_noair

        t = t + DT
    return X_values[-1]   #Devuelve el ultimo valor de la lista



def opt_angle(B, Diam, g, mass, Vt):

    Xmaxs_values = []
    angles_values = []
```

```
    for i in range(0,900,1):
        angle= i/10

        Xmax = calc_XYT(B, Diam, g, mass, Vt, angle)
        Xmaxs_values.append(Xmax)
        angles_values.append(angle)


    # Encontrar el máximo valor en Xmaxs_values
    max_Xmaxs = max(Xmaxs_values)

    # Encontrar el índice correspondiente al máximo valor
    indice_max_Xmaxs = Xmaxs_values.index(max_Xmaxs)

    # Obtener el valor correspondiente en angles_values
    max_angle = angles_values[indice_max_Xmaxs]

    # Plot Vy vs t
    plt.plot(Xmaxs_values, angles_values, marker='o', label='mass = ' + str(mass) + '  diam
    plt.xlabel('X (m)')
    plt.ylabel('Angle (°)')
    plt.title('Exercise  3b - Xmax vs Angle')
    plt.legend()
    plt.grid(True)

    print(max_angle)


########################################################################
#g, mass previusly defined
#Quadratic term depreciated

#constants
B=1.6e-4
g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg

#Initital Speed, angle
Vt = 2
```

```
anglemax = opt_angle(B, Diam, g, mass, Vt)

anglemax = opt_angle(B, Diam, g, mass*2, Vt)
anglemax = opt_angle(B, Diam, g, mass*5, Vt)
anglemax = opt_angle(B, Diam, g, mass*10, Vt)
anglemax = opt_angle(B, Diam, g, mass*20, Vt)
anglemax = opt_angle(B, Diam, g, mass*30, Vt)
anglemax = opt_angle(B, Diam, g, mass*100, Vt)


##Plot angle vs mass
##From the previus plot

opth_angle =[24.8, 32, 38.9, 42.8, 43, 43.6, 44.6]
mass_x =[1, 2, 5, 10, 20, 30, 100]

plt.figure()
plt.plot(mass_x, opth_angle, marker='o', label='Opth angle')
plt.xlabel('xmass')
plt.ylabel('angle (°)')
plt.title('Exercise  3b - Opth angle vs mass')
plt.legend()
plt.grid(True)
# Establecer límites del eje y entre 0 y 5
plt.show()

Exercise 4:
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 18:58:44 2024

@author: vwitch
"""
import math
import matplotlib.pyplot as plt

#########################################################################
#####                   Exercise  4a                             #####
#########################################################################
def calc_XYT(B, C, Diam, g, mass, Vt, angle):

    Vxi = Vt * math.cos (math.radians(angle))
    Vyi = Vt * math.sin (math.radians(angle))

    b = B*Diam
```

```
c = C*Diam**2


DT = 0
t=0

#For LINEAR air resistence calculation
Vy_lin = Vyi   #Initital Speed
Vx_lin = Vxi   #Initital Speed
DVy_lin = 0
Y_lin = 0
DVx_lin = 0
X_lin = 0

#For QUADRATIC air resistence calculation
Vy_quad = Vyi   #Initital Speed
Vx_quad = Vxi   #Initital Speed
DVy_quad = 0
Y_quad = 0
DVx_quad = 0
X_quad = 0


#Without air resistence calculation
Vy_noair = Vyi   #Initital Speed
Vx_noair = Vxi   #Initital Speed
DVy_noair=0
Y_noair=0
DVx_noair=0
X_noair=0

# Lists to plot Vy, Vx, Y, X and t

#For LINEAR air resistence calculation
t_values = []
Y_values_lin = []
X_values_lin = []

#For QUAD air resistence calculation
Y_values_quad = []
X_values_quad = []

#Without air resistence calculation
Y_values_noair = []
X_values_noair = []
```

```
while Y_noair >= 0:     #executes while Y for the no air force calculation is higher tha

    t_values.append(t)
    # Steps =  0.01 seconds
    DT = 0.001

    #For LINEAR air resistence calculation
    Y_values_lin.append(Y_lin)
    X_values_lin.append(X_lin)

    DVy_lin = -g * DT - (b / mass) * Vy_lin * DT
    Vy_lin = Vy_lin + DVy_lin
    DVx_lin = - (b / mass) * Vx_lin * DT
    Vx_lin = Vx_lin + DVx_lin

    DY_lin = Vy_lin * DT
    Y_lin = Y_lin + DY_lin
    DX_lin = Vx_lin * DT
    X_lin = X_lin + DX_lin


    #For QUADRATIC air resistence calculation
    Y_values_quad.append(Y_quad)
    X_values_quad.append(X_quad)

    DVy_quad = -g * DT - (c / mass) * math.sqrt(Vx_quad**2 + Vy_quad**2) * Vy_quad * DT
    DVx_quad = - (c / mass) * math.sqrt(Vx_quad**2 + Vy_quad**2) * Vx_quad * DT

    Vy_quad = Vy_quad + DVy_quad
    Vx_quad = Vx_quad + DVx_quad

    DY_quad = Vy_quad * DT
    Y_quad = Y_quad + DY_quad
    DX_quad = Vx_quad * DT
    X_quad = X_quad + DX_quad

    #Without air resistence calculation
    Y_values_noair.append(Y_noair)
    X_values_noair.append(X_noair)

    DVy_noair = -g * DT
    Vy_noair = Vy_noair + DVy_noair
    DVx_noair = 0
    Vx_noair = Vx_noair + DVx_noair
```

```
        DY_noair = Vy_noair * DT
        Y_noair = Y_noair + DY_noair
        DX_noair = Vx_noair * DT
        X_noair = X_noair + DX_noair

        t = t + DT

    # Plot Vy vs t
    plt.plot(X_values_lin, Y_values_lin, marker='o', label='Linear term, vel: ' + str(Vt) +
    plt.plot(X_values_quad, Y_values_quad, marker='o', label='Quadratic term, vel: ' + str(V
    plt.plot(X_values_noair, Y_values_noair, marker='o', label='No air resistence, vel: ' +
    plt.xlabel('X (m)')
    plt.ylabel('Y (m)')
    plt.title('Exercise  4a - Y vs X')
    plt.legend()
    plt.grid(True)

    # Establecer límites del eje y entre 0 y ymaximo default
    plt.ylim(0)  #establece el minimo en Y en cero y deja el máximo en valor default
    plt.show()


########################################################################
#g, mass previusly defined
#Quadratic term depreciated

#constants
B=1.6e-4
C=0.25
g=9.81

#data for the spherical grain
dens = 2000   #kg/m3
Diam =  1e-4    #m
volume = 4/3*math.pi * (Diam/2)**3   #m3
mass = dens*volume   #kg

#Initital Speed, angle

calc_XYT(B, C, Diam, g, mass, 3, 45)
calc_XYT(B, C, Diam, g, mass, 3, 22)
```