Computational Physics
Fall 2020

Lalit Chaudhary

October 6, 2020

Realistic Projectile Motion

Contents

1 Equations of motion **2**

2 Solution for Low velocity friction **3**
2.1 Validation of Code 3
2.2 The Trajectory 4

3 Solution for High Velocity Friction **5**
3.1 n = 3/2; Medium Velocity Friction 5
3.2 n = 2; High Velocity Friction 6

4 Trajectory for various Initial conditions **7**

5 Analysis of Range and Time of Flight **9**

A
Full Code **10**

List of Figures

1 Code Snippet for modified Euler's Algorithm to solve two ODEs 3
2 Comparison of Euler's Algorithm with Analytical Solution 4
3 Trajectory of Projectile in for low-velocity friction 5
4 Trajectory of Projectile with n = 3/2 6
5 Trajectory of Projectile with n = 2 7
6 Trajectory of Projectile with low initial velocity 7
7 Trajectory of Projectile with medium initial velocity 8
8 Trajectory of Projectile with high initial velocity 8

1 Equations of motion

For the frictional model given by:

$$\begin{aligned}\vec{F_f} &= -km\,|v|^n \cdot \frac{\vec{v}}{|v|} \\ &= -km\,|v|^{n-1}\,(\vec{v_x} + \vec{v_y})\end{aligned} \tag{1}$$

where,

$$\vec{v} = \vec{v_x} + \vec{v_y} \Rightarrow |v| = \sqrt{{v_x}^2 + {v_y}^2}$$

So the net force acting on a projectile is:

$$\begin{aligned}\vec{F} &= \vec{F_f} + m\vec{g} \\ &= -km\,|v|^{n-1}\,(v_x \cdot \hat{\boldsymbol{x}} + v_y \cdot \hat{\boldsymbol{y}}) + mg(0 \cdot \hat{\boldsymbol{x}} - 1 \cdot \hat{\boldsymbol{y}}) \\ &= (-km\,|v|^{n-1}\,v_x)\hat{\boldsymbol{x}} + (-km\,|v|^{n-1}\,v_y - mg)\hat{\boldsymbol{y}}\end{aligned}$$

Then, acceleration, $\vec{a}$ is:

$$\vec{a} = (-k\,|v|^{n-1}\,v_x)\hat{\boldsymbol{x}} + (-k\,|v|^{n-1} - g)\hat{\boldsymbol{y}} \tag{2}$$

Hence, the equations of motion can be written as:

$$\frac{d^2x}{dt^2} = a_x = -k\,|v|^{n-1}\,v_x \tag{3}$$

$$\frac{d^2y}{dt^2} = a_y = -k\,|v|^{n-1}\,v_y - g \tag{4}$$

$$|v| = \sqrt{{v_x}^2 + {v_y}^2}$$

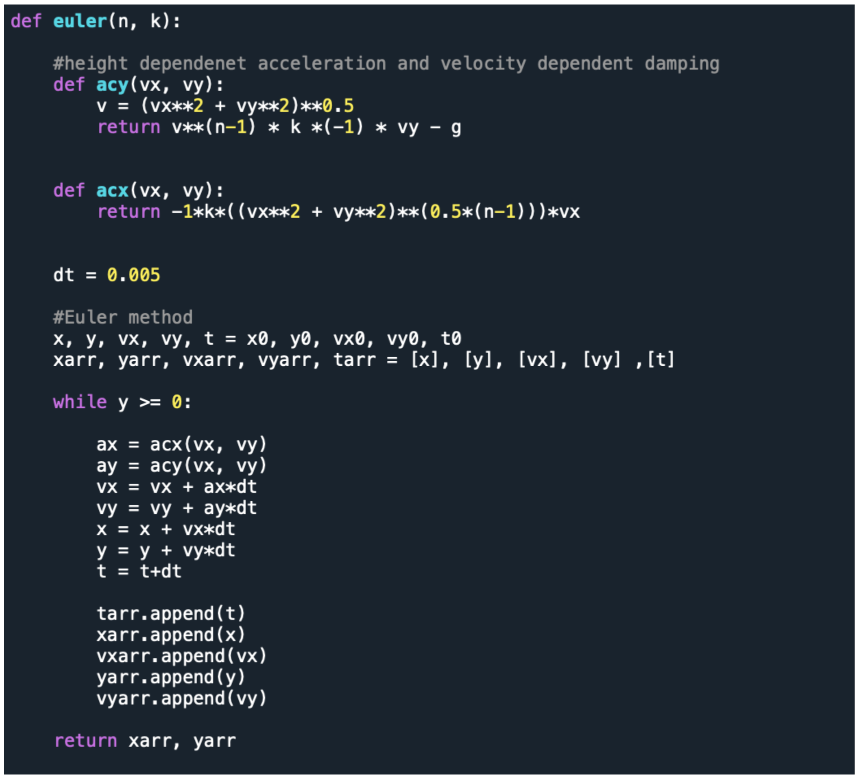
In terms of first order differential equations, equation (3) can be rewritten as:

$$\begin{aligned}\frac{dx}{dt} &= v_x \\ \frac{dv_x}{dt} &= -k\,|v|^{n-1}\,v_x\end{aligned}$$

Similarly, equation (4) can be rewritten as

$$\begin{aligned}\frac{dy}{dt} &= v_y \\ \frac{dv_y}{dt} &= -k\,|v|^{n-1}\,v_y - g\end{aligned}$$

2 Solution for Low velocity friction



```python
def euler(n, k):

    #height dependenet acceleration and velocity dependent damping
    def acy(vx, vy):
        v = (vx**2 + vy**2)**0.5
        return v**(n-1) * k *(-1) * vy - g


    def acx(vx, vy):
        return -1*k*((vx**2 + vy**2)**(0.5*(n-1)))*vx


    dt = 0.005

    #Euler method
    x, y, vx, vy, t = x0, y0, vx0, vy0, t0
    xarr, yarr, vxarr, vyarr, tarr = [x], [y], [vx], [vy] ,[t]

    while y >= 0:

        ax = acx(vx, vy)
        ay = acy(vx, vy)
        vx = vx + ax*dt
        vy = vy + ay*dt
        x = x + vx*dt
        y = y + vy*dt
        t = t+dt

        tarr.append(t)
        xarr.append(x)
        vxarr.append(vx)
        yarr.append(y)
        vyarr.append(vy)

    return xarr, yarr
```

Figure 1: Code Snippet for modified Euler's Algorithm to solve two ODEs

2.1 Validation of Code

To check the validity of code, the solution was plotted against the analytical solution for the case of no $k = 0$ i.e. no friction. The plot is shown in fig.(2).

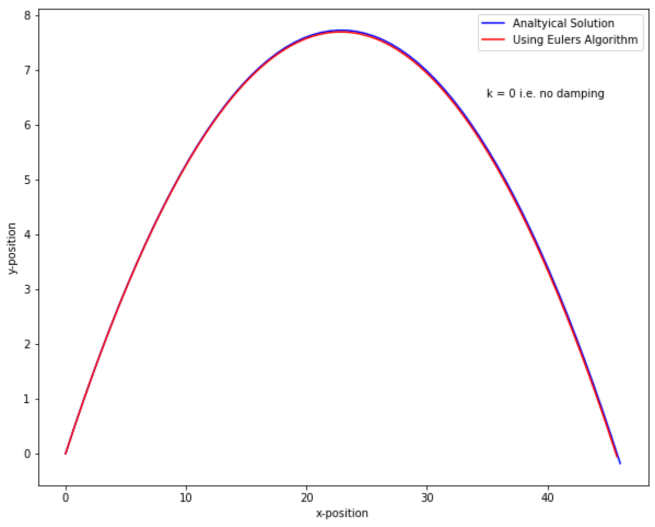


Figure 2: Comparison of Euler's Algorithm with Analytical Solution

2.2 The Trajectory

For the initial condition,

$$\begin{aligned} n &= 1 \\ v_0 &= 22\, m/s \\ \Theta &= 34\, radians \\ g &= 9.81\, m/s^2 \\ k &= 0.8 \end{aligned}$$

the trajectory of the projectile is:

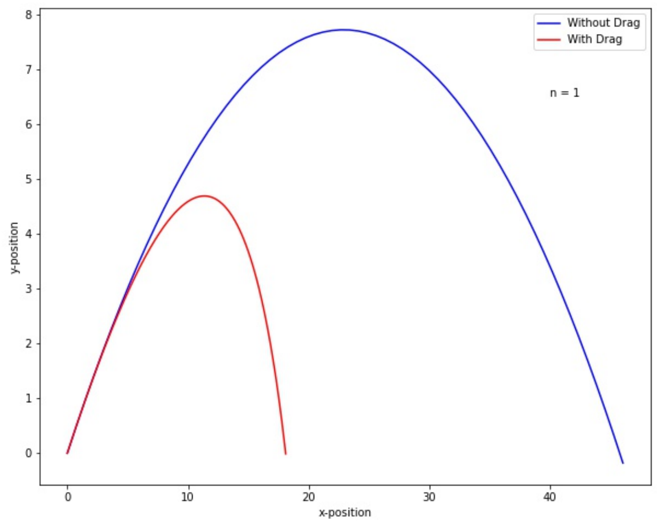


Figure 3: Trajectory of Projectile in for low-velocity friction

3 Solution for High Velocity Friction

3.1 n = 3/2; Medium Velocity Friction

For the initial condition,

$$\begin{aligned} n &= \frac{3}{2} \\ v_0 &= 22\, m/s \\ \Theta &= 34\, radians \\ g &= 9.81\, m/s^2 \\ k &= 0.17 \end{aligned}$$

the trajectory of the projectile is:

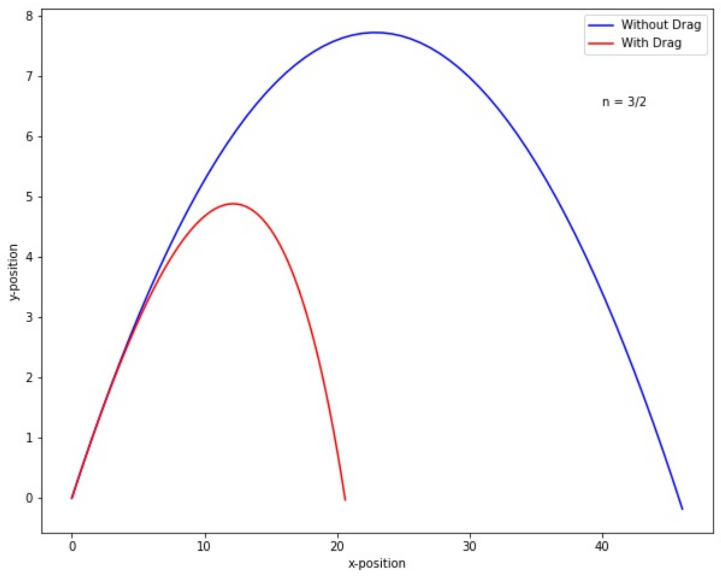


Figure 4: Trajectory of Projectile with n = 3/2

3.2 n = 2; High Velocity Friction

For the initial condition,

$$\begin{aligned} n &= 2 \\ v_0 &= 22\, m/s \\ \Theta &= 34\, radians \\ g &= 9.81\, m/s^2 \\ k &= 0.036 \end{aligned}$$

the trajectory of the projectile is:

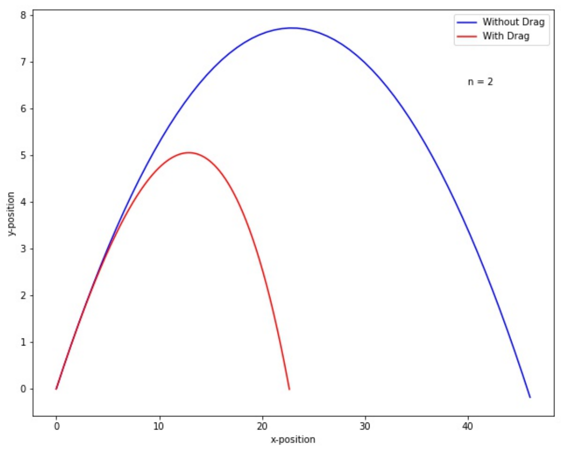


Figure 5: Trajectory of Projectile with n = 2

4 Trajectory for various Initial conditions

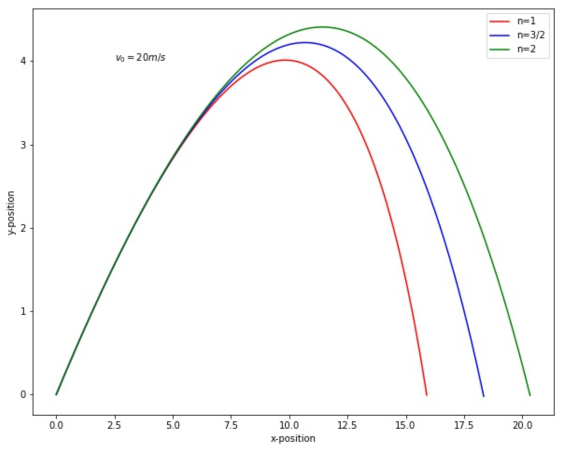


Figure 6: Trajectory of Projectile with low initial velocity

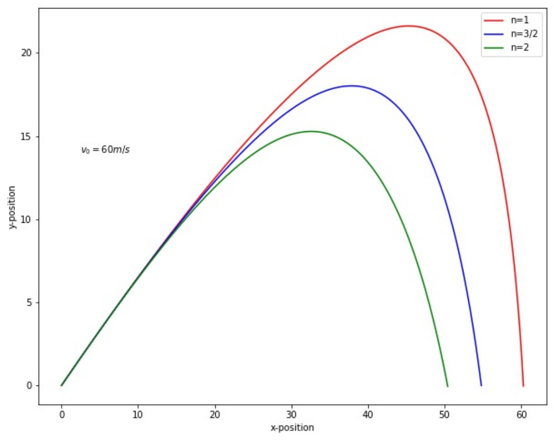


Figure 7: Trajectory of Projectile with medium initial velocity

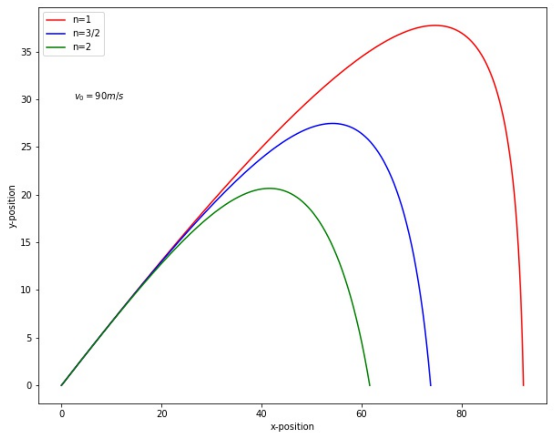


Figure 8: Trajectory of Projectile with high initial velocity

5 Analysis of Range and Time of Flight

As seen in figure (7) and (8), the horizontal range and the maximum height (consequently, the time of flight) decreases with the increase in value of n i.e. friction. The trajectory also explains why the balls appear to fall straight down near the ground at the end of the trajectory in presence velocity-based drag force.

The anomaly seen in figure (6) can be accounted for our assumption of constant k, which is based only on the initial velocity. For better approximation, the value of k and n varies at various points in the trajectory based on the velocity at that point.

A Full Code

The general outline of the code looked like the following although some variables and plots were altered throughout the program to get desired graphs.

```
"""
Computational Physics
Project 2
@Lalit Chaudhary
l.chaudhary@jacobs-university.de
"""

import numpy as np
import matplotlib.pyplot as plt

#Setting up the constants
m = 1
#k = 0.8
g = 9.8
#n = 1
#Initial conditions
t0 = 0
y0 = 0

x0 = 0
v0 = 20

theta0 = 34/180 *np.pi

vx0 = v0 * np.cos(theta0)
vy0 = v0 * np.sin(theta0)

def euler(n, k):

    #height dependenet acceleration and velocity dependent damping
    def acy(vx, vy):
        v = (vx**2 + vy**2)**0.5
        return v**(n-1) * k *(-1) * vy - g


    def acx(vx, vy):
        return -1*k*((vx**2 + vy**2)**(0.5*(n-1)))*vx


    dt = 0.005
    #Euler method
    x, y, vx, vy, t = x0, y0, vx0, vy0, t0
    xarr, yarr, vxarr, vyarr, tarr = [x], [y], [vx], [vy] ,[t]

    while y >= 0:

        ax = acx(vx, vy)
        ay = acy(vx, vy)
        vx = vx + ax*dt
        vy = vy + ay*dt
        x = x + vx*dt
        y = y + vy*dt
        t = t+dt

        tarr.append(t)
        xarr.append(x)
        vxarr.append(vx)
        yarr.append(y)
        vyarr.append(vy)

    return xarr, yarr

#analytical
"""
th = np.linspace(0, 20, 500)
x,y=[],[]
yt = 1
for t in th:
    if yt >= 0:
        xt = x0+vx0 * t
        yt =y0+vy0 *t - 0.5 * g * t**2
        x.append(xt)
        y.append(yt)
"""
x1arr, y1arr = euler(1, 0.8)
x2arr, y2arr = euler(1.5, 0.17)
x3arr, y3arr = euler(2, 0.036)

plt.figure(1, figsize=(10,8))
#plt.plot(tarr, yarr, 'b-')
#plt.plot(x[:], y[:], 'b', label = 'Without Drag')
plt.plot(x1arr[:], y1arr[:], 'r', label = 'n=1')
plt.plot(x2arr[:], y2arr[:], 'b', label = 'n=3/2')
plt.plot(x3arr[:], y3arr[:], 'g', label = 'n=2')
plt.legend()
plt.text(2.5, 4, r'$v_0 = 20 m/s$')
plt.xlabel('x-position')
plt.ylabel('y-position')
plt.savefig('1.jpg')
plt.show()
```