Session 24 Programming Activity AS.171.416/610 Fall 2020

Your task is to create a GitHub repository for a new "universal artillery calculator" software package. Start the codebase for the project with the solution to Exercise 8.7 from the Newman textbook reproduced below and posted on the class Blackboard site. The first step of the project is to make the code work on the surfaces of both the Earth and Mars. Modify the code (committing after each significant change) until it works on both planets. It may be useful to know that on the surface of Mars $g = 3.71$ m s^{-2} and $\rho = 0.20$ kg m^{-3}. You should follow the steps given in Chapters 15 and 16 of the Hull and Scopatz textbook. Include a copy of the output of the "git log" command and a public GitHub link for your project. Then answer the following questions:

(a) Repeat part (c) of Problem 8.7 for both the Earth and Mars. What are the main differences?

(b) How far will a cannonball fired at 100 m s^{-1} and 30° to the horizontal travel on Mars?

(c) Is it possible to launch a cannonball on Mars to clear both the summit (21 km) and horizontal extent (600 km) of Olympus Mons (the tallest planetary mountain in the solar system) to hit a target on the other side? Explain why or why not.

NUMERICAL METHODS FOR PHYSICISTS

ACTIVITY FOR SESSION 24

Exercise 8.7: Trajectory with air resistance

Many elementary mechanics problems deal with the physics of objects moving or flying through the air, but they almost always ignore friction and air resistance to make the equations solvable. If we're using a computer, however, we don't need solvable equations.

Consider, for instance, a spherical cannonball shot from a cannon standing on level ground. The air resistance on a moving sphere is a force in the opposite direction to the motion with magnitude

$$F = \tfrac{1}{2}\pi R^2 \rho C v^2,$$

where R is the sphere's radius, ρ is the density of air, v is the velocity, and C is the so-called *coefficient of drag* (a property of the shape of the moving object, in this case a sphere).

a) Starting from Newton's second law, $F = ma$, show that the equations of motion for the position (x, y) of the cannonball are

$$\ddot{x} = -\frac{\pi R^2 \rho C}{2m}\,\dot{x}\sqrt{\dot{x}^2 + \dot{y}^2}, \qquad \ddot{y} = -g - \frac{\pi R^2 \rho C}{2m}\,\dot{y}\sqrt{\dot{x}^2 + \dot{y}^2},$$

where m is the mass of the cannonball, g is the acceleration due to gravity, and $\dot{x}$ and $\ddot{x}$ are the first and second derivatives of x with respect to time.

b) Change these two second-order equations into four first-order equations using the methods you have learned, then write a program that solves the equations for a cannonball of mass 1 kg and radius 8 cm, shot at 30° to the horizontal with initial velocity $100\,\mathrm{m\,s^{-1}}$. The density of air is $\rho = 1.22\,\mathrm{kg\,m^{-3}}$ and the coefficient of drag for a sphere is $C = 0.47$. Make a plot of the trajectory of the cannonball (i.e., a graph of y as a function of x).

c) When one ignores air resistance, the distance traveled by a projectile does not depend on the mass of the projectile. In real life, however, mass certainly does make a difference. Use your program to estimate the total distance traveled (over horizontal ground) by the cannonball above, and then experiment with the program to determine whether the cannonball travels further if it is heavier or lighter. You could, for instance, plot a series of trajectories for cannonballs of different masses, or you could make a graph of distance traveled as a function of mass. Describe briefly what you discover.

```
# start from Example 8.5 program odesim.py
import numpy as np
import matplotlib.pyplot as plt

# constants
g     = 9.81    # m s^-2
m     = 1.0     # kg
rho   = 1.22    # kg m^-3
C     = 0.47    # unitless
R     = 0.08    # m
h     = 0.001   # seconds
theta = 30.0*(np.pi/180) # radians
v0    = 100.0   # m s^-1
const = (rho*C*np.pi*R**2)/(2.0*m)

# define the equations of motion
def f(r,const):
    x   = r[0]
    y   = r[1]
    vx  = r[2]
    vy  = r[3]
    fx  = vx
    fy  = vy
    fvx = -const*vx*np.sqrt(vx**2+vy**2)
    fvy = -g-const*vy*np.sqrt(vx**2+vy**2)
    return np.array([fx,fy,fvx,fvy],float)

# containers for output
r = np.array([0.0,0.0,v0*np.cos(theta),v0*np.sin(theta)],float)
xpoints = []
ypoints = []

# use fourth-order Runge-Kutta
while r[1]>=0:
    k1 = h*f(r,const)
    k2 = h*f(r+0.5*k1,const)
    k3 = h*f(r+0.5*k2,const)
    k4 = h*f(r+k3,const)
    r += (k1+2*k2+2*k3+k4)/6
    xpoints.append(r[0])
    ypoints.append(r[1])

# make plot for part (b)
p1 = plt.figure(1)
plt.plot(xpoints,ypoints)
plt.xlabel('x [m]')
plt.ylabel('y [m]')
```

```
p1.show()

# try different values of m
p2 = plt.figure(2)
for m in [0.25,0.5,1,2,4]:
    const = (rho*C*np.pi*R**2)/(2.0*m)
    r = np.array([0.0,0.0,v0*np.cos(theta),v0*np.sin(theta)],float)
    xpoints = []
    ypoints = []

    # use fourth-order Runge-Kutta
    while r[1]>=0:
        k1 = h*f(r,const)
        k2 = h*f(r+0.5*k1,const)
        k3 = h*f(r+0.5*k2,const)
        k4 = h*f(r+k3,const)
        r += (k1+2*k2+2*k3+k4)/6
        xpoints.append(r[0])
        ypoints.append(r[1])

    plt.plot(xpoints,ypoints,label='m = '+str(m)+' kg')

plt.xlabel('x [m]')
plt.ylabel('y [m]')
plt.legend()
p2.show()
```