Trajectories of a Streamlined Projectile on Earth, the Moon, Mars, and Venus

Logan Spooner
(Dated: October 13, 2023)

Projectile motion with air resistance can be studied on 4 different terrestrial bodies within our solar system, which we include Earth, Luna (Earth's Moon), Mars, and Venus in particular. Our equations of motion include Newtonian gravity and quadratic drag to model the forces acting on a streamlined projectile in motion on each body. A fourth order Runge-Kutta method is implemented to find the optimal launch angle to minimize kinetic energy on each of the terrestrial bodies considered, which are near 45° (Earth, Mars, & Luna) and is around 18° (Venus). We use similar methods to solve the differential equations for the motion of the projectile and a Newton-Raphson searching method is used to optimize the launch velocity for the given conditions where the target range for the projectile on each body is 20 kilometers. Using the searching method and optimal launch angle, we find the required initial velocities for the probe to reach the target range on Earth, Luna, Mars, and Venus are about 463, 180, 273, and 28500 m/s, respectively. We discuss the density requirements for such a probe and use a model based off of the Apollo Lunar module with a density of around 2000 kgm^{-3}.

I. INTRODUCTION

Projectile motion is one of the most studied and well-known problems in physics. The problem consists of solving a system of differential equations based off Newton's law of gravity and can vary in complexity due to what parameters are considered in the calculation. Because of this varying complexity, projectile motion is studied by a range of physicists from undergraduate physics students to engineers at NASA [1]. The wide application of this topic make it one of the most important subjects in all of physics. With the recent advancement in space flight technology, and prospects for the exploration of our solar system, it has become increasingly more important to be able to model the projectile motion of objects on planets that may be candidates for future probing missions. The modeling of these parameters can be very tricky though due to the wildly different conditions on each planet. This is especially true when modeling a single probe to work on multiple planets.

The main issue that is encountered when modeling these equations is the difference in forces that will be acting on the probe. In the simplest case for bodies such as the Moon or Mercury there is only the force of gravity as there is no atmosphere to create drag. With the other bodies, the calculation becomes more complicated. Despite the added effect of air resistance, both Earth and Mars have very similar projectile trajectories to the Moon since their atmospheric densities are low, especially if a projectile with a low cross-sectional area is considered. The complexity arises when considering bodies such as Venus since the atmosphere is so dense that all but a few sets of conditions will allow a probe to travel any significant distance.

In this paper, we discuss the use of a fourth order Runge-Kutta (rk4) method that can be implemented to model the trajectories of a streamlined probe launched from the surface of four different bodies within our solar system. The methods employed to calculate the trajectories for each launch are discussed. Kinetic energy optimization and fuel-mass conservation are discussed along with the probe density conditions required for successful launch on atmospherically dense bodies. We show plots relating the launch angle and velocity with the final distance to determine the optimal launch angle. We then use this angle to model the trajectory of the probe on each body and determine the required initial velocities. Finally, we connect our findings to current research being done in the field and discuss other areas to be explored.

II. METHODS

A. Equations of Motion

To set up the equations of motion for the projectile we first determine the forces acting on the projectile on each body. For the calculations done here we will be using a quadratic drag force as described by [2]. The equation for quadratic drag force is given by:

$$\vec{F}_{D_i} = -\frac{1}{2}\rho C_D A|v|\vec{v}_i \tag{1}$$

where ρ is the surface air density, C_D is the coefficient for drag for the given projectile, A is the cross-sectional area of the projectile perpendicular to the direction of motion, and $\vec{v}_i$ is the velocity vector for the i component (i.e. v_x is the x-component of velocity). This force will vary depending on what planet and projectile shape we choose. This is not the only force that will be acting on the projectile as the vertical component will also have a force of gravity acting on it calculated as:

$$F_G = -\frac{GmM}{R^2} \tag{2}$$

where m is the mass of the projectile, M is the mass of the body which we are launching from, R is the radius of the body we are launching from, and $G = 6.6743 \times 10^{-11} m^3 kg^{-1} s^{-2}$. The values of these constants for each body are given in Table I. This gives us the set of differential equations for acceleration:

$$\begin{aligned} \frac{d^2x}{dt^2} &= \frac{1}{m}[F_{D_x}] \\ \frac{d^2y}{dt^2} &= \frac{1}{m}[F_G + F_{D_y}] \\ \frac{dx}{dt} &= v_x \\ \frac{dy}{dt} &= v_y \end{aligned} \tag{3}$$

Body	M (kg)	R (m)	ρ (kgm^{-3})
Earth	5.9722×10^{24}	6.371×10^6	1.217
Moon	$0.0123 M_E$	$0.2727 R_E$	0
Mars	$0.107 M_E$	$0.532 R_E$	0.020
Venus	$0.815 M_E$	$0.95 R_E$	65

TABLE I. This table shows the value for the mass, radius, and atmospheric density for Earth, Moon, Mars, and Venus. M_E and R_E represent mass and radius of the Earth.

B. Solving the System

Now that we have our set of differential equations, we can begin solving the system for the state vector consisting of our position and velocity functions: x, v_x, y, v_y. We arbitrarily choose a time range of $T = 200s$ as this will allow plenty of time for most trajectories to go to completion. A fourth order Runge-Kutta (rk4) system is then utilized that will take in our system of differential equations (equation (3)), apply them to the initial values for our state vector, iterate over our time range and return the next set of values for the state vector. This method uses a Taylor series approximation to simplify the computation of the integrals.

This system depends on the initial velocity and launch angle that we choose. The angle that we use is the optimal launch angle to minimize the kinetic energy of the probe and will depend on the specific characteristics of each body's atmosphere. A velocity of 10 m/s is chosen as a starting velocity. This velocity will most likely not be the correct launch velocity to give us the trajectories we want so we must use some method to find the necessary velocity. To do this, we create a function that determines the difference in our target distance of 20,000 m and the final horizontal distance in our rk4 system, which will be a function of the initial velocity, giving:

$$D_{diff} = 20,000 - x_f \tag{4}$$

This equation equals zero when our final position is equal to our target position. We can use a Newton-Raphson root finding method to find this zero. This gives us the necessary initial velocity required for the probe to land 20 km away after launch for each body.

III. RESULTS

A. Projectile Characteristics

The projectile chosen for this model is a streamlined body with a coefficient of drag of $C_d = 0.04$. This low coefficient of drag is chosen to minimize the drag force on the projectile due in particular to Venus' extreme atmospheric density. For this same reason, a small cross-sectional area is chosen as well. Assuming the streamlined body has a circular cross-section, we choose a diameter of $d = 1$ meter, giving us a cross-sectional area of approximately $A = 0.79m^2$. To make the scenario as realistic as possible, we let our projectile have the density of the Apollo Lunar Module with full fuel tanks at around $2250 kg/m^3$. We also assume that over the short period of launch the mass does not change significantly enough to make a difference in the trajectory.

The reason for the specifics on the density of the projectile is due to the drag force on Venus. Since the drag force and thus acceleration scales with area it is important to minimize the area of the projectile. Another issue arises when considering the acceleration also scales with the inverse of mass. Due to these two correlations, there seems to be a cut-off at around densities of $2000 kg/m^3$ where any streamlined projectile with a lower density will not reach the target range no matter the velocity or launch angle. This consideration is also taken assuming the length of the projectile (assuming the shape approximates a cylinder) is no more than twice the diameter. We can assume that the density requirement will scale with the ratio of the length to width of the projectile.

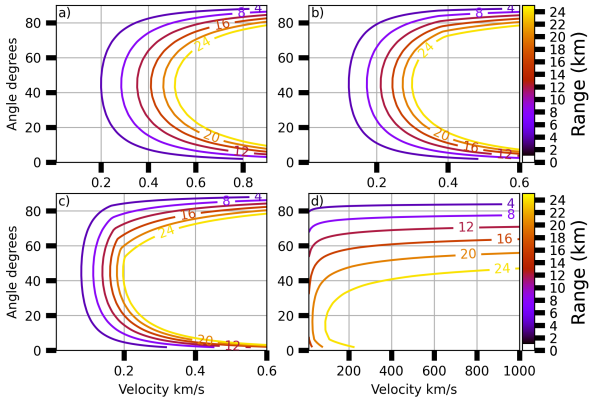

FIG. 1. This figure shows the contour plots for the relation between initial velocity and angle with final position. The order of the bodies represented by the plots are Earth, Mars, Moon, and Venus. There is a noticeable downward shift in the optimal launch angle for Venus compared to the other bodies.

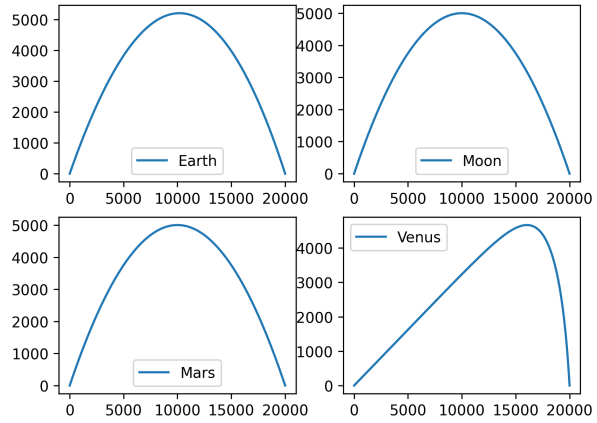

FIG. 2. This figure shows the trajectories for the probe on each body. Due to the low drag force on the probe, the trajectories are nearly parabolic for Earth, Moon, and Mars. The drag force is much more noticeable for Venus due to its much higher atmospheric density.

B. Minimizing Energy

To better ensure the mass of the projectile will not change significantly with time, we can minimize the fuel expenditure by minimizing the required launch velocity. We know for simple projectile motion without air resistance, the launch angle that minimizes the required velocity will simply be 45° [3]. This is not true for a trajectory with air resistance which has been shown analytically [4]. We will create a model that can determine the launch angle of minimum velocity numerically. We do this by creating a contour plot for each of the bodies showing the final position for different launch conditions and finding the set that minimizes the launch velocity.

After doing this, we get a contour plot that shows the maximum distances for different velocity and angle pairs seen in Figure 1. The launch angle of minimum velocity for Earth, Luna, and Mars are all very close to 45°. This is due to the streamlined body having such a low drag force on these bodies (zero for Luna). The angle is much different for Venus at around 18°. This correlates to the much higher drag force from Venus' dense atmosphere.

C. Trajectories

Once we have a realistic probe design and know the optimal launch angle, we can use the procedure outlined in the Methods section for each body given the parameters in Table I. After solving the system of equations and optimizing for our desired final positions, we find the initial speeds referenced in Table II. The probe launched on Venus requires a significantly higher initial velocity due to Venus' high atmospheric density compared to the other bodies. In fact, the launch conditions for Venus are so harsh that all but a few of the initial conditions will actually result in the probe making it the required distance.

Body	θ (degrees)	v_0 (m/s)	x_f (m)	y_f (m)
Earth	45	462.9	20000.5	0.6
Moon	45	180.2	20000.2	0.3
Mars	45	272.7	20000.4	0.4
Venus	18	28548.0	20000.0	0.5

TABLE II. This table shows the launch and landing conditions for the probe on each body. θ is the launch angle, v_0 is the initial launch velocity, x_f is the final horizontal distance, and y_f is the final height of the projectile. Notice that the trajectory is not exact and there is some error in the final positions.

We can then plot the data for the probe's motion on each body to see its trajectory. Figure 2 shows the trajectories for each body. The trajectory for the Moon is parabolic due to its lack of an atmosphere making it simple projectile motion. The trajectories for Earth and Mars are both nearly parabolic because the affect of their atmospheres are so small on the streamlined projectile. We can see a drastic difference in the trajectory on Venus with the x-component of the velocity decreasing over time and the y-component decreasing much faster than in the other cases.

IV. CONCLUSION

The motion of a projectile experiencing a drag force on a body can be modeled by equations based on Newton's law of gravity and a quadratic drag force. A fourth order Runge-Kutta method can be employed to solve for the system of differential equations of motion for the projectile. This method is employed to plot the trajectories of a streamlined body on Earth, Mars, Luna, and Venus with the terrestrial conditions listed in Table I. An optimal launch angle is determined using an rk4 method to create contour plots of the distances for given initial angles and velocities. We find that the optimal launch angle for the Moon is 45° while Earth and Mars are nearly the same. The optimal launch angle for Venus is found to be around 18° due to the large drag force of its extreme atmospheric density.

Table II lists the required launch velocities for the streamlined probe when launched at the optimal angle. Figure 2 shows the trajectory of the projectile with the given launch conditions where Earth, Luna, and Mars all have nearly parabolic trajectories while the probes trajectory on Venus experiences very obvious effects on the velocity components due to an extreme drag force. A high density probe is needed in order to have a successful launch on Venus due to the relations between velocity, drag force, mass, and acceleration outlined in equations (3). The density of the Apollo Lunar Lander at around $2250 kgm^{-3}$ is used as a realistic set up for our simulations giving a successful launch on each body.

There have been many recent studies done on the viability of using a single vehicle for launch and reentry within the atmospheres of different bodies for the purpose of system exploration such as [5] where methods for simulating these vehicles are outlined. There have also been many studies exploring the idea of return missions from the surface of Venus for the collection of samples for study. Studies such as [6] have explored some of these requirements in detail. The launch requirements are not the only thing that must be taken into consideration for certain bodies in our system such as Venus as the surface temperature requires very careful shielding which are explored in [7]. The results found in our study explore the requirements and conditions for just the launching of a streamlined body on the four different bodies outlined within and may be combined with results from some of these other studies to model a probe capable of safe launch and entry on multiple bodies within our solar system.

[1] Jeffrey C Andrews-Hanna, James W Head, Brandon Johnson, James T Keane, Walter S Kiefer, Patrick J McGovern, Gregory A Neumann, Mark A Wieczorek, and Maria T Zuber. Nasa public access. *Icarus*, 310:1–20, 2018.

[2] Vladimir Ivchenko. On projectile motion with quadratic drag force. *European Journal of Physics*, 39(4):045004, apr 2018.

[3] Bons Bušie. A simple solution for maximum range of projectile motion. *The Physics Teacher*, 51(1):52–52, 2013.

[4] Peter Chudinov. An optimal angle of launching a point mass in a medium with quadratic drag force, 2005.

[5] Monika Auweter-Kurtz, Helmut L Kurtz, and Stefan Laure. Plasma generators for re-entry simulation. *Journal of Propulsion and Power*, 12(6):1053–1061, 1996.

[6] Ted Sweetser, Craig Peterson, Erik Nilsen, and Bob Gershman. Venus sample return missions—a range of science, a range of costs. *Acta Astronautica*, 52(2):165–172, 2003. Selected Proceedings of the 4th IAA International conference on L ow Cost Planetary Missions.

[7] David L Peterson and William E Nicolet. Heat shielding for venus entry probes. *Journal of Spacecraft and Rockets*, 11(6):382–387, 1974.

Appendix A: Code

1. Velocity-Angle-Distance Data

```
    import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit,bisect
from scipy.constants import G
import os

path = os.getcwd()+"/Planets"
```

```
#Earth Constants
M_earth=5.9722*10**24 # kg
R_earth=6371*10**3 # m
Rho_earth = 1.217 #kgm^-3

#Moon constants
M_moon = 0.0123*M_earth
R_moon=0.2727*R_earth
Rho_moon = 0

#Mercury constants
M_mars=0.107*M_earth #kg
R_mars=0.532*R_earth#m
Rho_mars=0.020 #kg/m^3

#Venus constants
M_venus = 0.815*M_earth
R_venus = 0.950*R_earth
Rho_venus = 65 #kgm^-3


x_f=20000

def central_diff(func,x,step):
    return (func(x+step/2)-func(x-step/2))/step

def grav_accl(M,R):
    return -G*M/(R**2)

def rk4(y,t,h,derivs,args):
    #function to implement rk4
    #y = [x,v] current state
    #t = current time
    #h = time step
    #derivs = derivative function that defines the problem
    k1,k2,k3,k4 = np.zeros(2),np.zeros(2),np.zeros(2),np.zeros(2)
    k1 = h*derivs(y,t,args)
    y_halfstep = y + k1/2. #Euler half step using k1
    k2 = h*derivs(y_halfstep,t+h/2,args)
    y_halfstep = y + k2/2. #Euler half step using k2
    k3 = h*derivs(y_halfstep,t+h/2,args)
    k4 = h*derivs(y + k3,t+h,args) #full step using k3
    y_next = y + (k1+2*k2+2*k3+k4)/6.
    return y_next

def Newt_Raph(func,x_0,h,tol,N):
    '''func takes in function to be evaluated
    x_0 is first guess
    h is accuracy for derivative
    tol dertermines how close to the exact solution we want
    N determines iteration limit'''
    while abs(func(x_0)) > tol:
        for i in range(N):
            dx = -(func(x_0))/(central_diff(func,x_0,h))
            x_0 += dx
```

```
            print(func(x_0))
        x_0 =x_0 +dx/2
        return x_0

def proj_mot(z,t,args):
    #z=[x_t,vx_t,y_t,vy_t]
    #F=-mg-Fd
    Rho, A, Cd, m, g = args
    zp=np.zeros(4)
    vmag = np.sqrt(z[1]**2 + z[3]**2)
    zp[0]=z[1]
    zp[2]=z[3]
    zp[1]=-0.5*(Rho/m)*A*Cd*z[1]*vmag
    zp[3]=g-0.5*(Rho/m)*A*Cd*z[3]*vmag
    return zp

planets=np.array([[M_earth,R_earth,Rho_earth],[M_moon,R_moon,Rho_moon],[M_mars,R_mars,Rho_mars],[M_venus
def motion(pl,v_0,theta):
    Cd=0.04 #Streamlined booger
    d = 1
    A= np.pi*d**2/4
    m = 2250*A*(2*d) #kg
    M=pl[0]
    R=pl[1]
    Rho=pl[2]
    g = grav_accl(M,R)
    args = [Rho, A, Cd, m, g]

    N=100000 #number of steps
    T=200 #Total time
    h=T/(float(N-1)) #step size
    time=np.arange(0,T+h,h)

    #Initial conditions
    y_0=0
    x_0=0

    vx_0=v_0*np.cos(np.radians(theta))
    vy_0=v_0*np.sin(np.radians(theta))

    states=np.zeros((N,4))
    #print(type(x_0),type(vx_0),type(y_0),type(vy_0))
    z=[x_0,vx_0,y_0,vy_0]
    states[0,:]=z

    for j in range(0,N-1):
        states[j+1,:]=rk4(states[j,:],time[j],h,proj_mot,args)
        if states[j+1,2] < 0:
            states = states[:j,:]
            #print(states[-1,:])
            break
    return states

def plot_trajec(planet,v1,v2,name,pos):
```

```
    def xfunc(veloc):
        return x_f - max(motion(pl,veloc,th)[:,0])
    theta=np.linspace(1,90,18)
    velocity=np.linspace(v1,v2,100)

    file = os.path.join(path,name+'.txt')
    out=open(file,'w')
    for th in theta:
        for veloc in velocity:
            States = motion(planet,veloc,th)
            xmax=States[-1,0]
            ymax= max(States[:,2])
            stuff = '%s\t%s\t%s\t%s\n' % (veloc,th,xmax,ymax)
            out.write(stuff)
    out.close()

#plot_trajec(planets[0],10,900,'Earth',1)import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit,bisect
from scipy.constants import G

#Earth Constants
M_earth=5.9722*10**24 # kg
R_earth=6371*10**3 # m
Rho_earth = 1.217 #kgmˆ-3

#Moon constants
M_moon = 0.0123*M_earth
R_moon=0.2727*R_earth
Rho_moon = 0

#Mercury constants
M_mars=0.107*M_earth #kg
R_mars=0.532*R_earth#m
Rho_mars=0.020 #kg/mˆ3

#Venus constants
M_venus = 0.815*M_earth
R_venus = 0.950*R_earth
Rho_venus = 65 #kgmˆ-3

x_f=20000

def central_diff(func,x,step):
    return (func(x+step/2)-func(x-step/2))/step

def grav_accl(M,R):
    return -G*M/(R**2)

def rk4(y,t,h,derivs,args):
    #function to implement rk4
    #y = [x,v] current state
    #t = current time
    #h = time step
    #derivs = derivative function that defines the problem
    k1,k2,k3,k4 = np.zeros(2),np.zeros(2),np.zeros(2),np.zeros(2)
```

```
    k1 = h*derivs(y,t,args)
    y_halfstep = y + k1/2. #Euler half step using k1
    k2 = h*derivs(y_halfstep,t+h/2,args)
    y_halfstep = y + k2/2. #Euler half step using k2
    k3 = h*derivs(y_halfstep,t+h/2,args)
    k4 = h*derivs(y + k3,t+h,args) #full step using k3
    y_next = y + (k1+2*k2+2*k3+k4)/6.
    return y_next

def Newt_Raph(func,x_0,h,tol,N):
    '''func takes in function to be evaluated
    x_0 is first guess
    h is accuracy for derivative
    tol dertermines how close to the exact solution we want
    N determines iteration limit'''
    while abs(func(x_0)) > tol:
        for i in range(N):
            dx = -(func(x_0))/(central_diff(func,x_0,h))
            x_0 += dx
            print(func(x_0))
        x_0 =x_0 +dx/2
        return x_0

def proj_mot(z,t,args):
    #z=[x_t,vx_t,y_t,vy_t]
    #F=-mg-Fd
    Rho, A, Cd, m, g = args
    zp=np.zeros(4)
    vmag = np.sqrt(z[1]**2 + z[3]**2)
    zp[0]=z[1]
    zp[2]=z[3]
    zp[1]=-0.5*(Rho/m)*A*Cd*z[1]*vmag
    zp[3]=g-0.5*(Rho/m)*A*Cd*z[3]*vmag
    return zp

planets=np.array([[M_earth,R_earth,Rho_earth],[M_moon,R_moon,Rho_moon],[M_mars,R_mars,Rho_mars],[M_venus
def motion(pl,v_0,theta):
    Cd=0.04 #Streamlined booger
    d = 1
    A= np.pi*d**2/4
    m = 2250*A*(2*d) #kg
    M=pl[0]
    R=pl[1]
    Rho=pl[2]
    g = grav_accl(M,R)
    args = [Rho, A, Cd, m, g]

    N=100000 #number of steps
    T=200 #Total time
    h=T/(float(N-1)) #step size
    time=np.arange(0,T+h,h)

    #Initial conditions
    y_0=0
    x_0=0
```

```
    vx_0=v_0*np.cos(np.radians(theta))
    vy_0=v_0*np.sin(np.radians(theta))

    states=np.zeros((N,4))
    #print(type(x_0),type(vx_0),type(y_0),type(vy_0))
    z=[x_0,vx_0,y_0,vy_0]
    states[0,:]=z

    for j in range(0,N-1):
        states[j+1,:]=rk4(states[j,:],time[j],h,proj_mot,args)
        if states[j+1,2] < 0:
            states = states[:j,:]
            #print(states[-1,:])
            break
    return states

def plot_trajec(planet,v1,v2,th,name,pos):
    y_f=1000
    xmax=0
    pl=planet

    def xfunc(veloc):
        return x_f - max(motion(pl,veloc,th)[:,0])

    Vel2=Newt_Raph(xfunc,v1,1,1e-4,15)
    States2 = motion(planet,Vel2,th)
    xmax2=States2[-1,0]
    y_f2 = States2[-1,2]
    print(name,'2 :',xmax2,y_f2,Vel2)

    ax=fig.add_subplot(2,2,pos)
    x,y=States2[:,0],States2[:,2]
    ax.plot(x,y,label=name)
    ax.legend()

fig=plt.figure()
plot_trajec(planets[0],450,500,35,'Earth',1)
plot_trajec(planets[1],160,210,35,'Moon',2)
plot_trajec(planets[2],255,305,35,'Mars',3)
plot_trajec(planets[3],53350,53550,35,'Venus',4)
plot_trajec(planets[1],10,600,'Moon',2)
plot_trajec(planets[2],10,600,'Mars',3)
plot_trajec(planets[3],12000,1000000,'Venus',4)
```

2. Contour Plots

3. Projectile Trajectories

```
    import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit,bisect
```

```
from scipy.constants import G

#Earth Constants
M_earth=5.9722*10**24 # kg
R_earth=6371*10**3 # m
Rho_earth = 1.217 #kgmˆ-3

#Moon constants
M_moon = 0.0123*M_earth
R_moon=0.2727*R_earth
Rho_moon = 0

#Mercury constants
M_mars=0.107*M_earth #kg
R_mars=0.532*R_earth#m
Rho_mars=0.020 #kg/mˆ3

#Venus constants
M_venus = 0.815*M_earth
R_venus = 0.950*R_earth
Rho_venus = 65 #kgmˆ-3

x_f=20000

def central_diff(func,x,step):
    return (func(x+step/2)-func(x-step/2))/step

def grav_accl(M,R):
    return -G*M/(R**2)

def rk4(y,t,h,derivs,args):
    #function to implement rk4
    #y = [x,v] current state
    #t = current time
    #h = time step
    #derivs = derivative function that defines the problem
    k1,k2,k3,k4 = np.zeros(2),np.zeros(2),np.zeros(2),np.zeros(2)
    k1 = h*derivs(y,t,args)
    y_halfstep = y + k1/2. #Euler half step using k1
    k2 = h*derivs(y_halfstep,t+h/2,args)
    y_halfstep = y + k2/2. #Euler half step using k2
    k3 = h*derivs(y_halfstep,t+h/2,args)
    k4 = h*derivs(y + k3,t+h,args) #full step using k3
    y_next = y + (k1+2*k2+2*k3+k4)/6.
    return y_next

def Newt_Raph(func,x_0,h,tol,N):
    '''func takes in function to be evaluated
    x_0 is first guess
    h is accuracy for derivative
    tol dertermines how close to the exact solution we want
    N determines iteration limit'''
    while abs(func(x_0)) > tol:
        for i in range(N):
            dx = -(func(x_0))/(central_diff(func,x_0,h))
            x_0 += dx
```

```
            print(func(x_0))
        x_0 =x_0 +dx/2
        return x_0

def proj_mot(z,t,args):
    #z=[x_t,vx_t,y_t,vy_t]
    #F=-mg-Fd
    Rho, A, Cd, m, g = args
    zp=np.zeros(4)
    vmag = np.sqrt(z[1]**2 + z[3]**2)
    zp[0]=z[1]
    zp[2]=z[3]
    zp[1]=-0.5*(Rho/m)*A*Cd*z[1]*vmag
    zp[3]=g-0.5*(Rho/m)*A*Cd*z[3]*vmag
    return zp

planets=np.array([[M_earth,R_earth,Rho_earth],[M_moon,R_moon,Rho_moon],[M_mars,R_mars,Rho_mars],[M_venus
def motion(pl,v_0,theta):
    Cd=0.04 #Streamlined booger
    d = 1
    A= np.pi*d**2/4
    m = 2250*A*(2*d) #kg
    M=pl[0]
    R=pl[1]
    Rho=pl[2]
    g = grav_accl(M,R)
    args = [Rho, A, Cd, m, g]

    N=100000 #number of steps
    T=200 #Total time
    h=T/(float(N-1)) #step size
    time=np.arange(0,T+h,h)

    #Initial conditions
    y_0=0
    x_0=0

    vx_0=v_0*np.cos(np.radians(theta))
    vy_0=v_0*np.sin(np.radians(theta))

    states=np.zeros((N,4))
    #print(type(x_0),type(vx_0),type(y_0),type(vy_0))
    z=[x_0,vx_0,y_0,vy_0]
    states[0,:]=z

    for j in range(0,N-1):
        states[j+1,:]=rk4(states[j,:],time[j],h,proj_mot,args)
        if states[j+1,2] < 0:
            states = states[:j,:]
            #print(states[-1,:])
            break
    return states

def plot_trajec(planet,v1,v2,th,name,pos):
    y_f=1000
    xmax=0
```

```
    pl=planet

    def xfunc(veloc):
        return x_f - max(motion(pl,veloc,th)[:,0])

    Vel2=Newt_Raph(xfunc,v1,1,1e-4,15)
    States2 = motion(planet,Vel2,th)
    xmax2=States2[-1,0]
    y_f2 = States2[-1,2]
    print(name,'2 :',xmax2,y_f2,Vel2)

    ax=fig.add_subplot(2,2,pos)
    x,y=States2[:,0],States2[:,2]
    ax.plot(x,y,label=name)
    ax.legend()

fig=plt.figure()
plot_trajec(planets[0],450,500,35,'Earth',1)
plot_trajec(planets[1],160,210,35,'Moon',2)
plot_trajec(planets[2],255,305,35,'Mars',3)
plot_trajec(planets[3],53350,53550,35,'Venus',4)
```