assignment1

April 26, 2023

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import sympy
from pylab import *
import matplotlib.animation as animation
from IPython import display
import math
import pandas as pd
from scipy.spatial.transform import Rotation as R
```

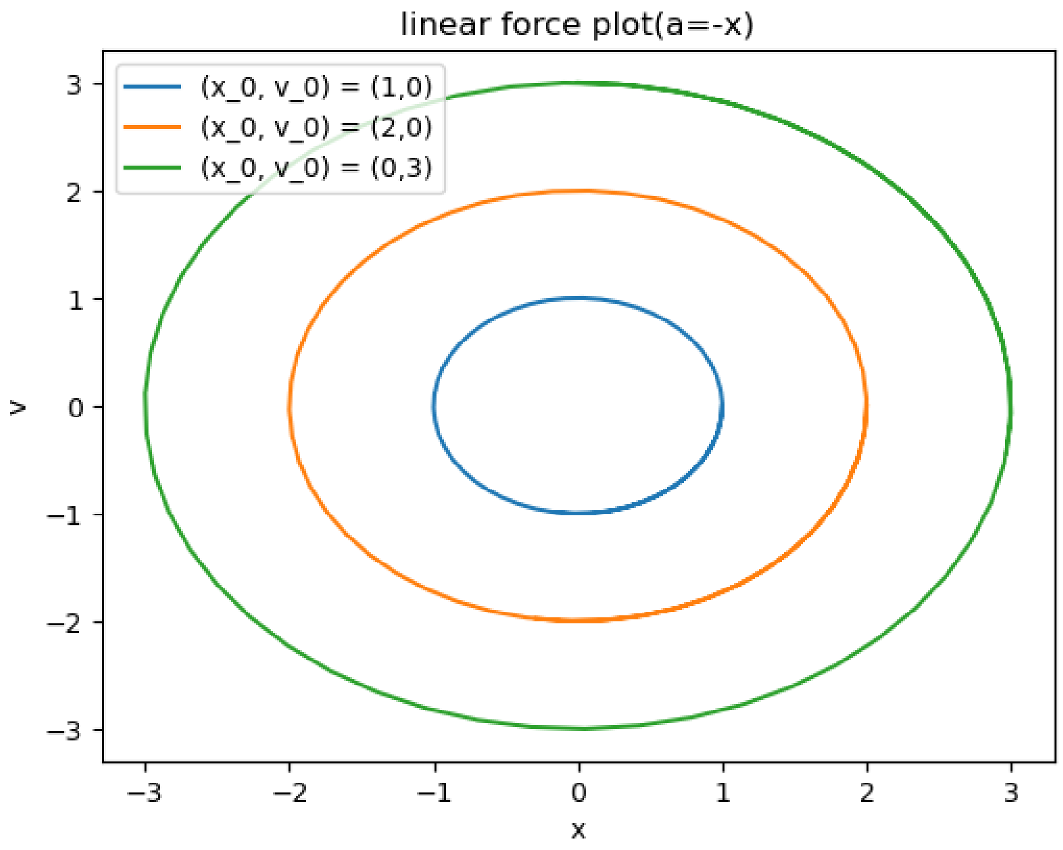
0.0.1 1a

plotting (x,v)

[ ]:
```
# process information from file
def file_process(s):
    f = open(s, "r")
    lines = f.readlines()
    f.close()
    convert = []
    result = []

    for line in lines:
        # process each line
        for element in line.split():
            convert.append(float(element))
        result.append(convert)
        convert = []

    # convert to numpy
    # first column is tnow, third column is position, fourth column is velocity
    result = np.array(result)
    return result
```

[ ]:
```
# reading from files
result1_l = file_process("result1_l.txt")
result2_l = file_process("result2_l.txt")
```

```
result3_l = file_process("result3_l.txt")


# getting data and plotting
time = result1_l[:,0]
x = result1_l[:,2]
v = result1_l[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (1,0)")

time = result2_l[:,0]
x = result2_l[:,2]
v = result2_l[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (2,0)")

time = result3_l[:,0]
x = result3_l[:,2]
v = result3_l[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (0,3)")

# set label
plt.xlabel("x")
plt.ylabel("v")
plt.title("linear force plot(a=-x)")

plt.legend()
plt.savefig("1_a_linear.png")
```

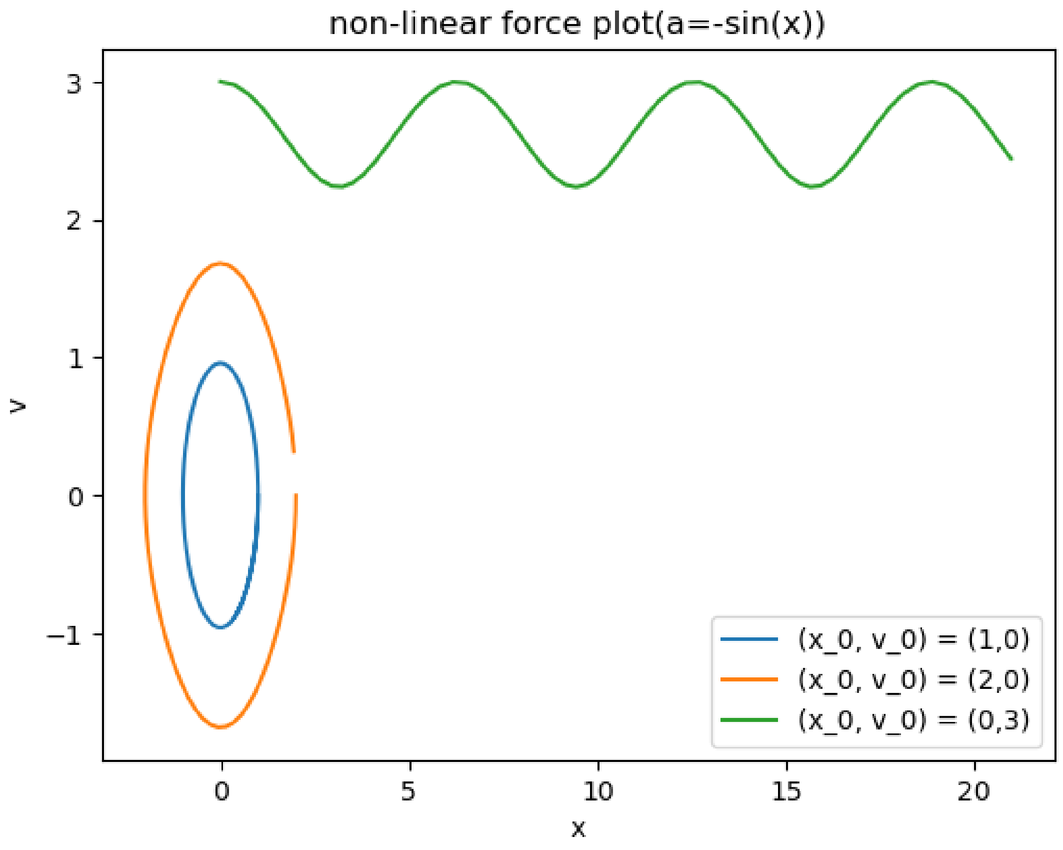


0.0.2 1b

non linear

```
[ ]: # same logic as 1a
result1 = file_process("result1.txt")
result2 = file_process("result2.txt")
result3 = file_process("result3.txt")

time = result1[:,0]
x = result1[:,2]
v = result1[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (1,0)")

time = result2[:,0]
x = result2[:,2]
v = result2[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (2,0)")

time = result3[:,0]
```

```
x = result3[:,2]
v = result3[:,3]
plt.plot(x, v, label = "(x_0, v_0) = (0,3)")
plt.xlabel("x")
plt.ylabel("v")
plt.title("non-linear force plot(a=-sin(x))")

plt.legend()
plt.savefig("1_b_nonlinear.png")
```



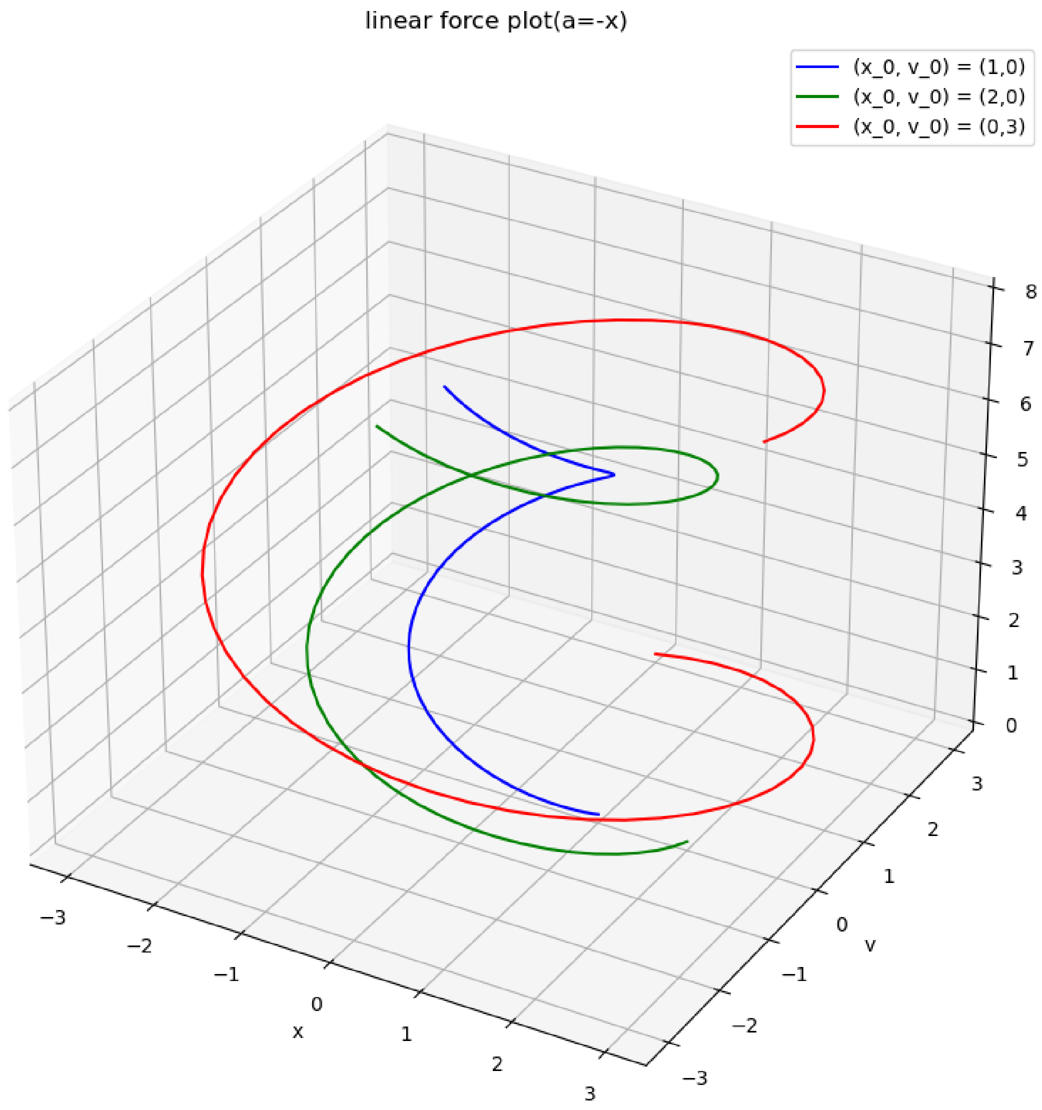
1 1c

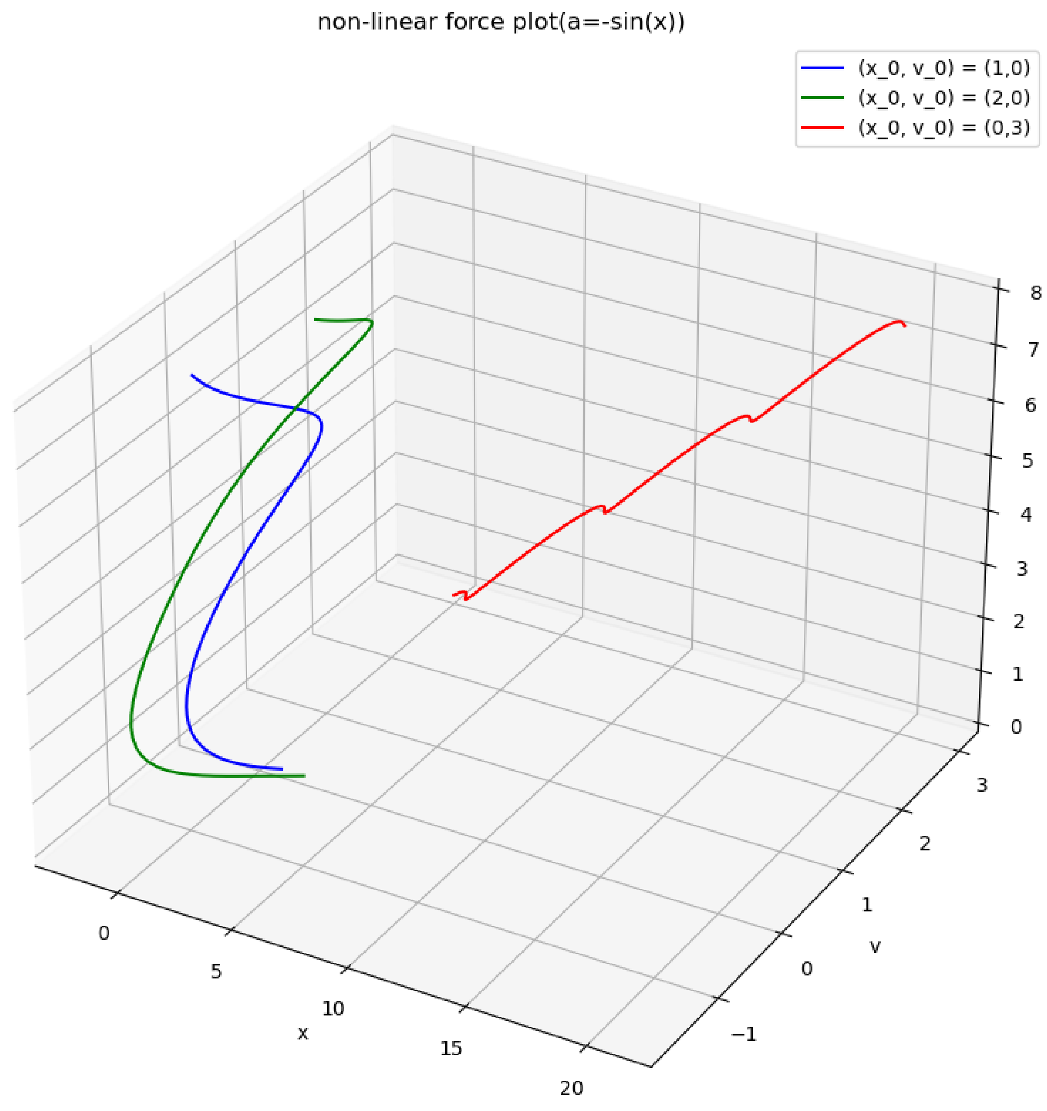
in 1c.py

2 1d

[ ]:
```
# plotting for linear force
time = result1_l[:,0]
x = result1_l[:,2]
v = result1_l[:,3]
fig = plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'blue', label = "(x_0, v_0) = (1,0)")

time = result2_l[:,0]
x = result2_l[:,2]
v = result2_l[:,3]
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'green', label = "(x_0, v_0) = (2,0)")

time = result3_l[:,0]
x = result3_l[:,2]
v = result3_l[:,3]
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'red', label = "(x_0, v_0) = (0,3)")
ax.set_xlabel("x")
ax.set_ylabel("v")
ax.set_zlabel("t")
ax.set_title("linear force plot(a=-x)")
ax.legend()
plt.savefig("1_d_linear.png")


# plotting for non-linear force
time = result1[:,0]
x = result1[:,2]
v = result1[:,3]
fig = plt.figure(figsize=(12,10))
ax = plt.axes(projection='3d')
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'blue', label = "(x_0, v_0) = (1,0)")

time = result2[:,0]
x = result2[:,2]
v = result2[:,3]
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'green', label = "(x_0, v_0) = (2,0)")

time = result3[:,0]
x = result3[:,2]
```

```
v = result3[:,3]
# Data for a three-dimensional line
ax.plot3D(x, v, time, 'red', label = "(x_0, v_0) = (0,3)")
ax.set_xlabel("x")
ax.set_ylabel("v")
ax.set_zlabel("t")
ax.set_title("non-linear force plot(a=-sin(x))")
ax.legend()
plt.savefig("1_d_nonlinear.png")
```





3 2a

orbital simulation

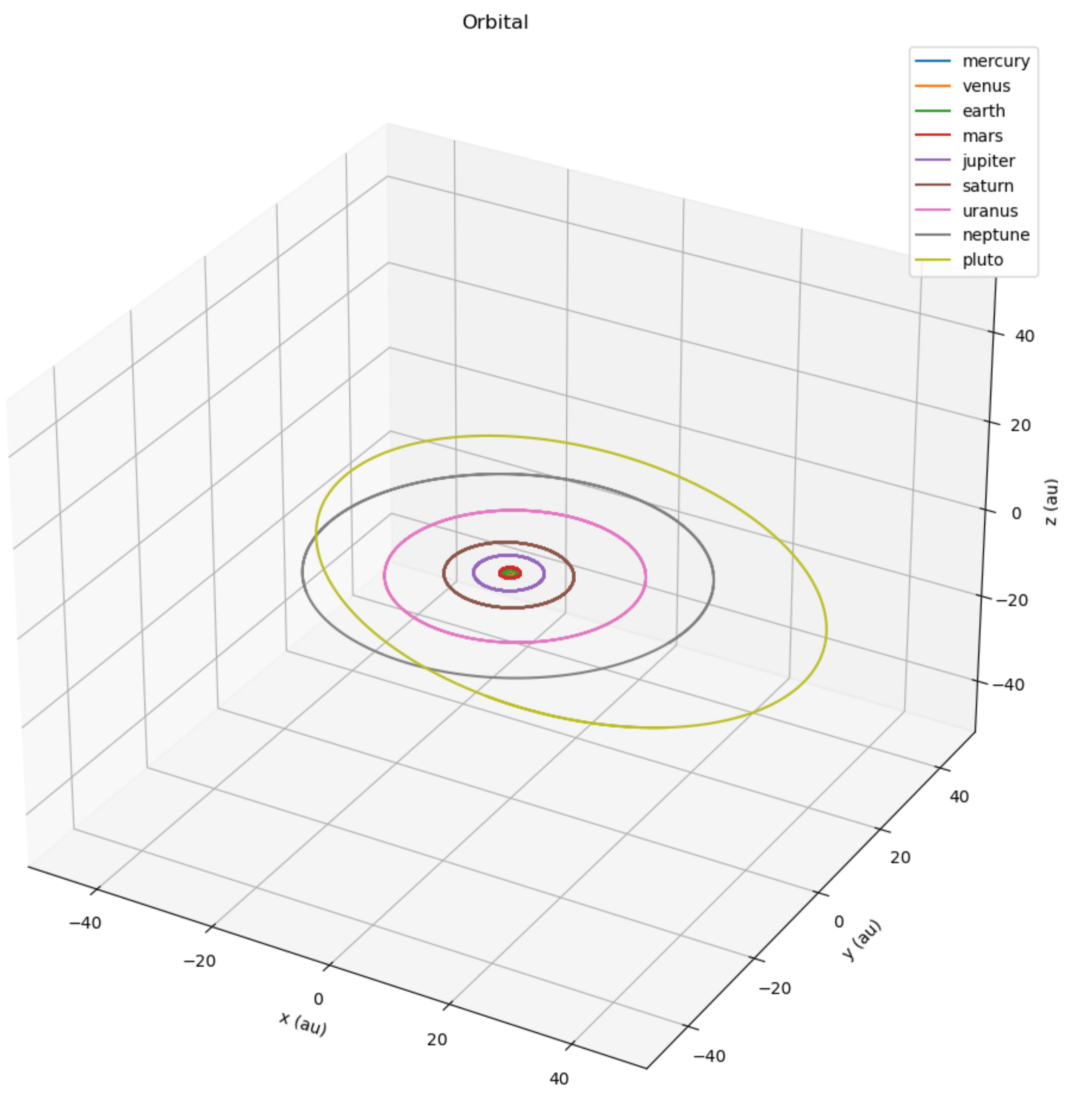
```
[ ]: def leapstep(x, v, n, dt, G, M, sun_pos, debugger):
         """LEAPSTEP: take one step using the leapfrog integrator, formulated
         as a mapping from t to t + dt.  WARNING: this integrator is not
         accurate unless the timestep dt is fixed from one call to another.
         Args:
             x (np.array): positions of all points
             v (np.array): velocities of all points
             n (int): number of planets
```

```python
        dt (float): timestep for integration
        sun_pos (3-d np.array): position of the sun
        debugger: debug purposes (not using)
    Returns:
        x: positions
        v: velocities
    """

    v = v + 0.5*dt*accel(x,n,G,M,sun_pos)
    x = x + v*dt
    v = v + 0.5*dt*accel(x,n,G, M, sun_pos)


    return x,v


def accel(x: np.ndarray, n: int, G: float, M: float, sun_pos: np.ndarray):
    """ACCEL: compute accelerations for harmonic oscillator(s)
    Args:
        x a (array of shape (n, 3)): positions of planets
        n (int): number of planets
        G (float): gravitational constant
        M (float): mass of sun
        sun_pos a (array of shape (1,3)): position of the sun

    Returns:
        a (array of shape (N, 3)): accelerations of points
    """
    a = np.zeros((n,3))
    for i in range(n):
        r_vec = x[i] - sun_pos
        r_mag = np.linalg.norm(r_vec) #np.sqrt((r_vec**2).sum())
        acc = -(G*M*r_vec/r_mag**3)
        a[i] = acc
    return a

def save_state(n, x, tnow, storage):
    """Save the current state to array.

    Args:
        n : size
        x (array of shape (N, 3)): positions of all points
        tnow (float): current time, in 0.01 year
        storage (python array): storing time, planet_label, r_x, r_y, r_z
    """
    for i in range(n):  # loop over all points...
        storage.append([tnow, i, x[i][0], x[i][1], x[i][2]])
```

```
[ ]: def main():

    # first, set up constants
    n = 9  # set number of points
    tnow = 0.0  # intial time in days
    tmax = 100000
    dt = 1  # day
    mstep = tmax / dt # total number of time steps
    x = np.zeros((n,3)) # position vector
    v = np.zeros((n,3)) # velocity vector

    # Gravitational constant and Sun's mass
    G = 2.959159*10**(-4)
    M = 1


    # set initial position and velocity
    f = open("initial.txt", "r")
    content = f.readlines()
    f.close()
    # reading from files the initial condition
    for i in range(n):
        line = content[i].replace('\n', '').split()
        x[i] = [float(line[0]), float(line[1]), float(line[2])]
        v[i] = [float(line[3]), float(line[4]), float(line[5])]
    sun_pos = np.array([0.0, 0.0, 0.0])

    result = []

    save_state(n,x,tnow, result)

    # leapfrog integration
    for i in range(int(mstep)):
        x, v = leapstep(x,v,n,dt,G,M, sun_pos, debugger=i)
        tnow = tnow + dt
        save_state(n, x, tnow, result)

    # storing result in panda dataframe
    result = np.array(result)
    df = pd.DataFrame(result, columns=["time", "planet_label", "r_x", "r_y",␣
 ↪"r_z"])
    return df

data_sim = main()
sim = data_sim # save for problem 3 use
sim.to_csv("sim.csv")
```

4 2b

```
[ ]: # order: mercury, venus, earth, mars, jupiter, saturn, uranus, neptune, pluto

planetDict = {'0': "mercury", '1' : 'venus', '2' : 'earth', '3': 'mars', '4':␣
 ↪'jupiter', '5': 'saturn', '6': 'uranus', '7': 'neptune', '8': 'pluto'}

fig = plt.figure(figsize = (12,12))
ax = plt.axes(projection='3d')

for i in range(9):
    planet = data_sim[data_sim.planet_label == i]
    x = planet['r_x'].to_numpy()
    y = planet['r_y'].to_numpy()
    z = planet['r_z'].to_numpy()
    ax.plot(x, y, z, label = planetDict[str(i)])

ax.set_xlim((-50, 50))
ax.set_ylim((-50, 50))
ax.set_zlim((-50, 50))
ax.set_title('Orbital')

ax.set_xlabel("x (au)")
ax.set_ylabel("y (au)")
ax.set_zlabel("z (au)")
ax.legend()

plt.show()
```



5 3

```
[ ]: # set initial position and velocity
     x = np.zeros((9,3))
     v = np.zeros((9,3))
     f = open("initial.txt", "r")
     content = f.readlines()
     f.close()
     for i in range(9):
         line = content[i].replace('\n', '').split()
         x[i] = [float(line[0]), float(line[1]), float(line[2])]
```

```
    v[i] = [float(line[3]), float(line[4]), float(line[5])]

# normal vector to the rotational plane
# L is (n by 3)
L = np.cross(x, v)
for i in range(9):
    L[i] = L[i] / np.linalg.norm(L[i])


# Gravitational constant and Sun's mass
G = 2.959159*10**(-4)
M = 1

# eccentricity, semi-major axis, and simulation parameter
ecc = np.array([0.2056, 0.0068, 0.0167, 0.0934, 0.0484, 0.0541, 0.0472, 0.0086,␣
 ↪0.2488]) # from wikipedia
axis = np.array([0.3871, 0.7233, 1, 1.5236, 5.2028, 9.5388, 19.1914, 30.0611,␣
 ↪39.5294]) # https://www.windows2universe.org/our_solar_system/
 ↪planets_orbits_table.html
n=9 # number of planets
npoint = 100 # number of discrete points
result = [] # store result
theta = np.linspace(0, 360, npoint) # initial angles
theta = np.radians(theta) # convert to radians

# calculate the radius given angle in radians
def kepler(radian, eccentricity, a):
    r = np.zeros((a.size, radian.size))
    for i in range(a.size):
        r[i] = a[i]*(1-eccentricity[i]**2)/(1-(eccentricity[i])*np.cos(radian))
    return r

# calculate the initial distance
def dist(r):
    r_init = []
    for i in range(n):
        r_init.append(np.linalg.norm(r[i]))
    return np.array(r_init)

r = kepler(theta, ecc, axis)
r_init = dist(x)
theta_init = np.arccos((1-axis*(1-ecc**2)/r_init)/ecc)
for i in range(n):
    if np.dot(x[i], v[i]) > 0:
        theta_init = - theta_init

result = [] # storing result
```

```
for i in range(n):
    for j in range(npoint):
        # get the rotational vector for from_rotvec function
        rot = L[i] * (theta[j] - theta_init[i])
        rot_operation = R.from_rotvec(rot) # rotational operator
        rotated_result = rot_operation.apply(x[i]/r_init[i]*r[i][j])
        result.append([i, rotated_result[0], rotated_result[1],␣
 ↪rotated_result[2]]) # saving the result

result = np.array(result) # convert to numpy array
result = pd.DataFrame(result, columns=["planet_label", "r_x", "r_y", "r_z"]) #␣
 ↪convert to panda dataframe
data_ana = result
data_ana.to_csv("ana.csv")
```

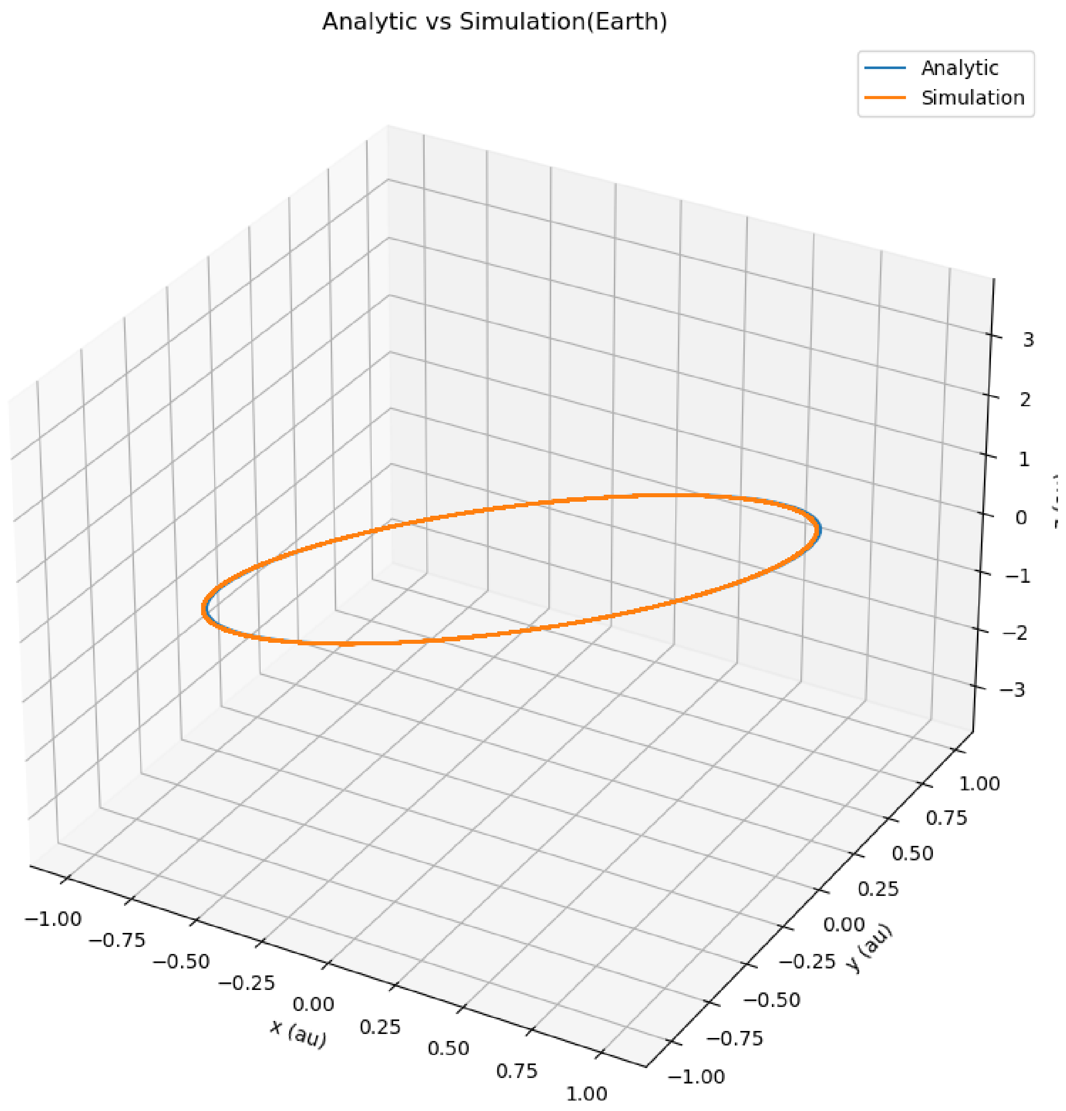
6 3a

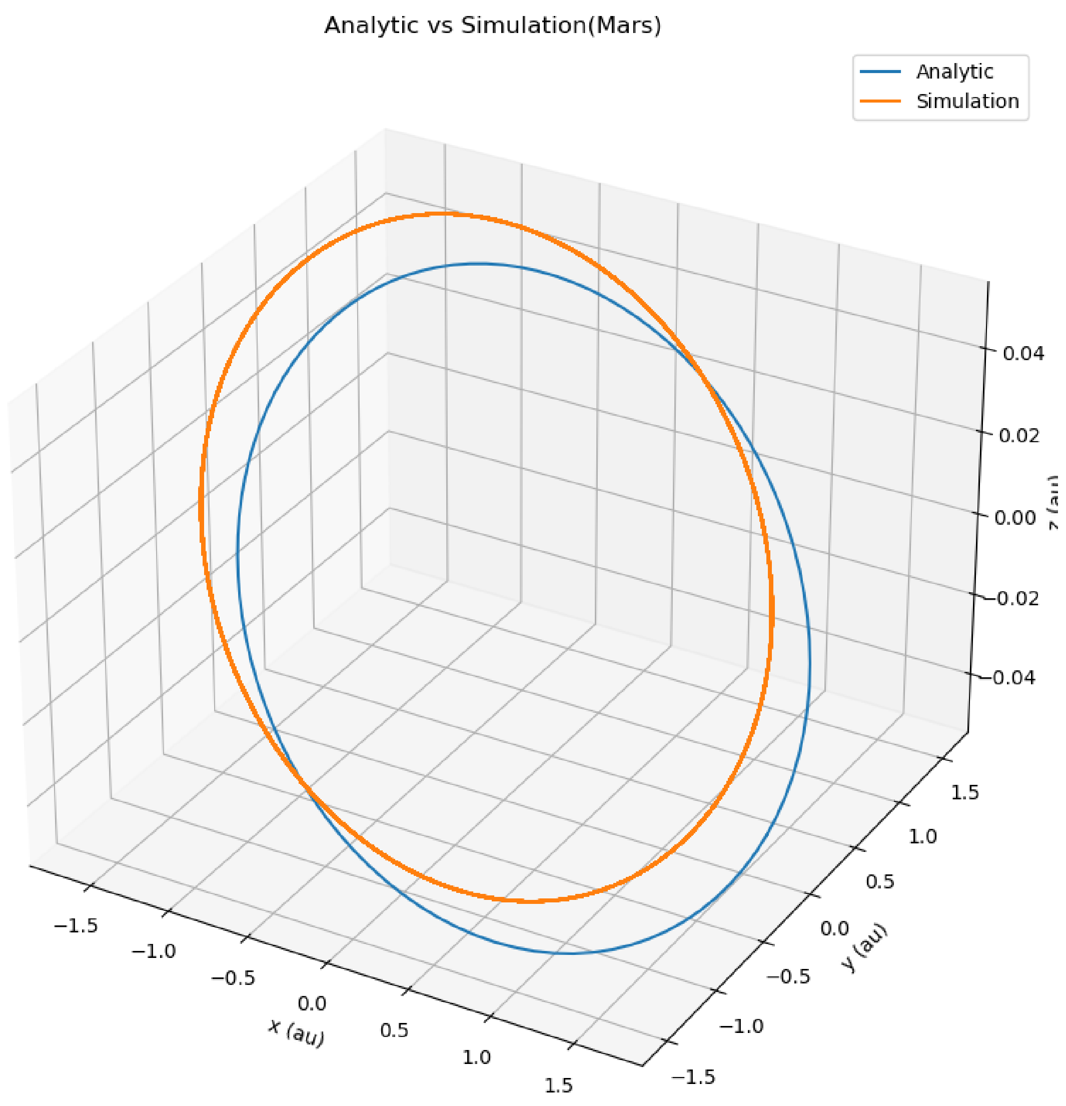
Indeed, if we look at the plot below, there's seems to be some slight deterioration for earth, and some big deterioration for mars. The deterioration for sun is likely be caused by the ecc and axis constant from the internet.

7 3b

[ ]:
```
#########################
#########################
# plotting for earth
# initialize plot
fig = plt.figure(figsize = (12,10))
ax = plt.axes(projection='3d')

# getting planet data
planet = data_ana[data_ana.planet_label == 2]
x = planet['r_x']
y = planet['r_y']
z = planet['r_z']
ax.plot3D(x, y, z, label = "Analytic")

planet_sim = sim[sim.planet_label == 2]
ax.plot3D(planet_sim['r_x'], planet_sim['r_y'], planet_sim['r_z'], label =␣
 ↪"Simulation")

# plotting parameters
ax.set_title("Analytic vs Simulation(Earth)")

ax.set_xlabel("x (au)")
```

```python
ax.set_ylabel("y (au)")
ax.set_zlabel("z (au)")
ax.legend()
plt.savefig("3b_Earth.png")

#########################
#########################
# plotting for mars
# initialize plot
fig = plt.figure(figsize = (12,10))
ax = plt.axes(projection='3d')

# getting planet data from analytic result
planet = data_ana[data_ana.planet_label == 3]
x = planet['r_x']
y = planet['r_y']
z = planet['r_z']
ax.plot3D(x, y, z, label = "Analytic")

# getting planet data from simulation result
planet_sim = sim[sim.planet_label == 3]
ax.plot3D(planet_sim['r_x'], planet_sim['r_y'], planet_sim['r_z'], label =␣
 ↪"Simulation")

ax.set_title("Analytic vs Simulation(Mars)")

ax.set_xlabel("x (au)")
ax.set_ylabel("y (au)")
ax.set_zlabel("z (au)")
ax.legend()
plt.savefig("3b_Mars.png")
```


Analytic vs Simulation(Earth)
Analytic
Simulation
x (au)
y (au)
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
3
2
1
0
−1
−2
−3



8 3c

in separate python file

9 4

Proof of conservation of angular momentum

```
[ ]: # define the position and velocity vector
     r = sympy.symbols("r")
     r_x = sympy.symbols("r_x")
```

```
r_y = sympy.symbols("r_y")
r_z = sympy.symbols("r_z")
v = sympy.symbols("v")
v_x = sympy.symbols("v_x")
v_y = sympy.symbols("v_y")
v_z = sympy.symbols("v_z")

# define timestep
dt = sympy.symbols("dt")

# define constants
G = sympy.symbols("G") # G constant
M = sympy.symbols("M") # mass of sun
m = sympy.symbols("m") # mass of planet

# create position and velocity vector
r = sympy.Matrix([r_x, r_y, r_z])
v = sympy.Matrix([v_x, v_y, v_z])

# calculate the accelaration
def acc(position, grav_c, M_value):
    return grav_c*M_value*position/(position.dot(position))

# leapfrog step
def leap(position, velocity, step, grav_c, M_value):
    velocity = velocity + 0.5*step*acc(position, grav_c, M_value)
    position = position + velocity*step
    velocity = velocity + 0.5*step*acc(position, grav_c, M_value)

    return position, velocity

# calling leapfrog function to get new position and new velocity
new_r, new_v = leap(r, v, dt, G, M)
print(new_r)
print(new_v)

# L = m*(x cross v)
L_before = m*sympy.simplify(r.cross(v))
L_after = m*sympy.simplify(new_r.cross(new_v))

# showing that angular momentum is conserved after leapfrog transformation
print(L_before.equals(L_after))
```

```
Matrix([[dt*(0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) + v_x) + r_x],
[dt*(0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) + v_y) + r_y],
[dt*(0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) + v_z) + r_z]])
Matrix([[0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) +
0.5*G*M*dt*(dt*(0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) + v_x) +
r_x)/((dt*(0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) + v_x) + r_x)**2 +
(dt*(0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) + v_y) + r_y)**2 +
(dt*(0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) + v_z) + r_z)**2) + v_x],
[0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) +
0.5*G*M*dt*(dt*(0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) + v_y) +
r_y)/((dt*(0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) + v_x) + r_x)**2 +
(dt*(0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) + v_y) + r_y)**2 +
(dt*(0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) + v_z) + r_z)**2) + v_y],
[0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) +
0.5*G*M*dt*(dt*(0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) + v_z) +
r_z)/((dt*(0.5*G*M*dt*r_x/(r_x**2 + r_y**2 + r_z**2) + v_x) + r_x)**2 +
(dt*(0.5*G*M*dt*r_y/(r_x**2 + r_y**2 + r_z**2) + v_y) + r_y)**2 +
(dt*(0.5*G*M*dt*r_z/(r_x**2 + r_y**2 + r_z**2) + v_z) + r_z)**2) + v_z]])
True
```

```
[ ]:
```