THEORETICAL ASTROPHYSICS LAB - EXERCISE 1

Planetary Orbits Around A Star

Sohaib Ali[1]

[a]*MSc Physics and Astronomy (PandA)*

Prof. Krzysztof Gęsicki

Abstract—This report employs Newton's laws of motion and law of gravitation to calculate the elliptical motion of a planet around the Sun. These laws were used to develop a numerical integrator of leap-frog type **[2]**. Various orbits have been predicted by changing initial conditions for the integrator. The integrator was developed using Python.

1. Introduction

For a body orbiting a central mass, Newton's law of universal gravitation states:

$$\mathbf{F} = -G\frac{M\,m}{r^2}\hat{\mathbf{r}} \tag{1}$$

Where **F** is the force vector, G is the gravitational constant, M and m are the masses of central object and orbiting body respectively, r is the distance between them, $\hat{\mathbf{r}}$ is the unit vector pointing along the line connecting the two masses, and the negative sign indicates that the force is attractive (directed toward the center). In this exercise, to simplify things and minimize numerical errors, we use normalized units G=M=1. This reduces 1 to:

$$\mathbf{a} = -\frac{\hat{\mathbf{r}}}{r^2} \tag{2}$$

Moreover, if we know the position and velocity of a body at a specific time t, we can use Newton's laws of motion to predict the trajectory of a particle in an N-dimensional space.

2. Methodology

We start by defining boundary conditions for the planet. We assume a two-dimensional space with x_0 and y_0 as initial position coordinates of the planet and the corresponding velocity components vx_0 and vy_0. We also assume that G and M in 1 are dimensionless and are equal to 1. Based on these initial positions and assumptions, I calculated r_0, $vx_{0,half}$, $vy_{0,half}$, ax_0, ay_0 using the following equations:

$$r_0 = \sqrt{(x_0^2 + y_0^2)} \tag{3}$$

$$ax_0 = -x_0/r_0^3 \tag{4}$$

$$ay_0 = -y_0/r_0^3 \tag{5}$$

$$vx_{0,half} = vx_0 + ax_0(dt/2) \tag{6}$$

$$vy_{0,half} = vy_0 + ay_0(dt/2) \tag{7}$$

Where r_0 is radial position at x_0 and y_0, ax_0 and ay_0 are accelerations due gravity at x_0 and y_0. Eqs. 4 and 5 are derived from 2. This was done to simplify our calculations for the orbit. As a consequence, no gravitational interaction was considered between the Sun and the planet. The half step velocities defined by 6 and 7 are a signature element of Leapfrog integration scheme in orbital dynamics. As per leapfrog method, half-step velocities allow the positions to be updated using a velocity value that represents an average between the current and future states. Feynmann **[1]** discusses the need to have mid-point velocities between two time-steps in order to not "miss-out" on the information about the motion of particles while computing its motion. By computing velocities at half-time steps, the algorithm effectively centers the update of positions and velocities in time, which leads to a local truncation error of order 3 and an overall second-order accurate integration.

3. Code Implementation

The above equations were translated into Python and a leapfrog integrator in the form of a python function was developed.3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  5 14:29:09 2025

@author: sohaib
"""

import numpy as np
from tabulate import tabulate
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd


def leapfrog(dt,steps,x0,y0,vx0,vy0):


    r0=np.sqrt(x0**2+y0**2) #radial position of planet at x0,y0
    print("r0 :",r0)

    ax0=-x0/r0**3 #acceleration component at x0
    ay0=-y0/r0**3 #acceleration component at y0
    print("ax(0) :",ax0)
    print("ay(0) :",ay0)

    vxhalf=vx0+ax0*(dt/2) #half integer time step velocity in x direction
    vyhalf=vy0+ay0*(dt/2) #half integer time step velocity in y direction
    print("vx(e/2) :",vxhalf)
    print("vy(e/2) :",vyhalf)

    time = 0.0 #started at time 0

    #bunch of initializations for arrays to store data later

    results=[]

    results.append([time, x0, y0, ax0, ay0, vx0, vy0,r0 ])


    #just to match the table in Feynman's book, I included an additional row in the data table to show half step velocities
    results.append([None, None, None, None, None, vxhalf,  vyhalf,None])

    xcurr = x0
    ycurr = y0
    vxhalf_curr = vxhalf
    vyhalf_curr = vyhalf

    #looping and calculating parameters until time interval reaches the limit

    for n in range(steps + 1):
        xnew= xcurr + vxhalf_curr*dt
        ynew= ycurr + vyhalf_curr*dt
```

```
        time+=dt

        rnew= np.sqrt(xnew**2+ynew**2)


        axnew=-xnew/rnew**3
        aynew=-ynew/rnew**3

        vxnew=vxhalf_curr + axnew*(dt/2)
        vynew=vyhalf_curr + aynew*(dt/2)

        results.append([time,xnew, ynew, axnew, aynew,
    ↪ vxnew, vynew,rnew])

        vxhalf_curr= vxnew + (dt/2)*axnew
        vyhalf_curr = vynew + (dt/2)*aynew

        results.append([None, None, None, None, None,
    ↪ vxhalf_curr, vyhalf_curr,None])

        xcurr = xnew
        ycurr = ynew




    #to print the table

    headers = ["time","x", "y", "ax", "ay", "vx", "vy",
     ↪ "r"]
    print(tabulate(
        results,
        headers=headers,
        tablefmt="plain",
        floatfmt=".3f",
        missingval=""
    ))

    #since i used seaborn to plot, converting the stored
     ↪  values in arrays to a pandas df was useful for
     ↪ plotting

    df = pd.DataFrame(results, columns=["time", "x", "y",
     ↪  "ax", "ay", "vx", "vy","r"])

    ax=sns.lineplot(data=df, x="x", y="y", marker=False,
     ↪  legend=0,sort=False)
    sns.scatterplot(x=[0], y=[0], color="yellow", s=150,
     ↪  edgecolor="black", ax=ax)




    ax.set(xlabel='x position ', ylabel= 'y position')
    ax.set(title=f'(dt: {dt}, steps: {steps}, x0: {x0},
     ↪ y0: {y0}, vx0: {vx0}, vy0: {vy0})')
    plt.suptitle('Motion of a planet around Sun')
    plt.figure(dpi=300)
    plt.show()

    return


#1st iteration - a coarse timestep
dt=0.1
steps=2000
x0=0.500
y0=0.000
vx0=0
vy0=1.630

leapfrog(dt,steps,x0,y0,vx0,vy0)
```

Using a coarse time step dt=0.100 in the first iteration leads to a Rosette-shaped orbit (Figure 1). A pseudo precession of the orbit often called numerical precession. The leapfrog method assumes that velocities at half-steps are good approximations for updating positions. However, if dt is too large, the velocity updates become less accurate. Due to a coarse time-step over a large number of steps i.e. 2000, the planet arrives at a slightly different position at the end of each iteration of loop. This leads to an open ellipse, consequently creating a precession-like effect that isn't physical. To make the large dt conform to the physical expectation, decreasing step-size dt to finer values e.g. dt=0.01 generates a perfectly elliptical orbit as shown in 2.

The values of each parameters calculated using eqs. 2, 3, 4, 5, 6 and 7 during each iteration are shown in truncated tables 1 and 2. As expected, we see more precise half-step velocities in 2, meaning that we can now model the position of the planet much more accurately within each time-step. This consequently removes the observed numerical precession and generates a perfectly elliptical orbit.

Other combination of initial conditions also yield some interesting orbits shown in figure 3. While figures 3a and 3b are obtained just by exchanging the initial velocities and positions from x to y, figures 3c represents a special orbit where increasing the velocity beyond a certain limit makes the Hamiltonian of the system positive, thereby, causing the planet to escape the system. 3d shows the limitations of our integrator that even though finer time-steps lead to more realistic solutions, in some combination of initial conditions, errors continue to add up which consequently lead to "drifts" nonetheless.

```

#2nd iteration
dt=0.01
steps=2000
x0=0.500
y0=0.000
vx0=0
vy0=1.630
leapfrog(dt,steps,x0,y0,vx0,vy0)
```

4. Figures

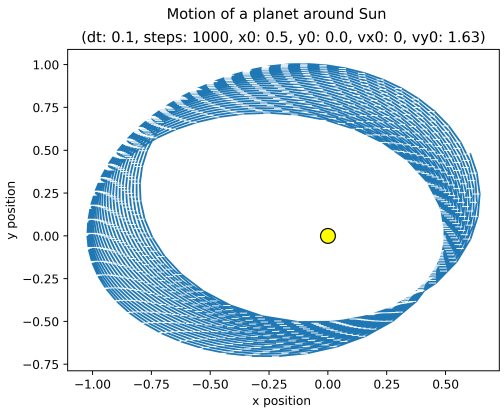


Figure 1. Rosette-shaped orbit, a direct consequence of using a coarse time-step over a large time interval which leads to accumulation of truncation errors introduced during each time-step.

5. Simulation Data

5.1. Iteration 1 - Coarse time-step

The simulation was initialized with $r_0 = 0.5$, $a_x(0) = -4.0$, $a_y(0) = -0.0$, $v_x(e/2) = -0.02$, and $v_y(e/2) = 1.63$. Table 1 presents the time evolution data. The second row after each full step contains the half-step velocity values.

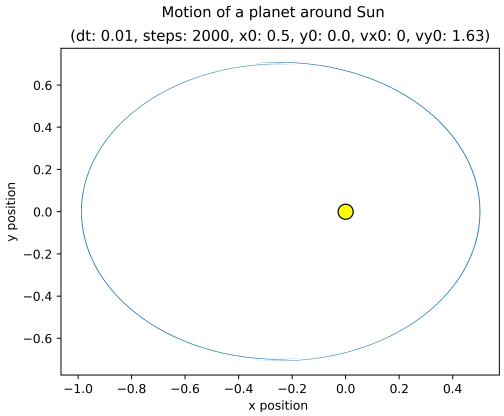


Figure 2. An elliptical orbit of a planet with a closed ellipse.

Table 1. Iteration 1 with dt=0.1

Time	x	y	a_x	a_y	v_x	v_y	r
0.000	0.500	0.000	-4.000	-0.000	0.000	1.630	0.500
					-0.200	1.630	
0.100	0.480	0.163	-3.685	-1.251	-0.384	1.567	0.507
					-0.568	1.505	
0.200	0.423	0.313	-2.897	-2.146	-0.713	1.398	0.527
					-0.858	1.290	
0.300	0.337	0.443	-1.958	-2.569	-0.956	1.162	0.556
					-1.054	1.033	
0.400	0.232	0.546	-1.112	-2.617	-1.110	0.903	0.593
					-1.165	0.772	
0.500	0.115	0.623	-0.454	-2.449	-1.188	0.649	0.634
					-1.211	0.527	
0.600	-0.006	0.676	0.018	-2.190	-1.210	0.417	0.676
					-1.209	0.308	
0.700	-0.127	0.706	0.342	-1.911	-1.192	0.212	0.718
					-1.175	0.117	
0.800	-0.244	0.718	0.559	-1.646	-1.147	0.034	0.758
					-1.119	-0.048	
0.900	-0.356	0.713	0.702	-1.408	-1.083	-0.118	0.797
					-1.048	-0.189	
1.000	-0.461	0.694	0.796	-1.200	-1.009	-0.249	0.833
					-0.969	-0.309	
1.100	-0.558	0.664	0.856	-1.019	-0.926	-0.360	0.867
					-0.883	-0.411	
1.200	-0.646	0.623	0.895	-0.862	-0.838	-0.454	0.897
					-0.794	-0.497	
1.300	-0.725	0.573	0.919	-0.726	-0.748	-0.533	0.924
					-0.702	-0.569	
1.400	-0.795	0.516	0.933	-0.605	-0.655	-0.600	0.948
					-0.608	-0.630	
1.500	-0.856	0.453	0.942	-0.498	-0.561	-0.655	0.969
					-0.514	-0.680	
1.600	-0.908	0.385	0.947	-0.402	-0.467	-0.700	0.986
					-0.420	-0.720	
1.700	-0.950	0.313	0.950	-0.313	-0.372	-0.736	1.000
					-0.325	-0.751	
1.800	-0.982	0.238	0.952	-0.230	-0.277	-0.763	1.010
					-0.229	-0.774	
1.900	-1.005	0.160	0.953	-0.152	-0.182	-0.782	1.018
					-0.134	-0.790	
2.000	-1.018	0.081	0.955	-0.076	-0.086	-0.793	1.022
					-0.038	-0.797	
2.100	-1.022	0.002	0.957	-0.002	0.009	-0.797	1.022
					0.057	-0.797	

Note: The second row after each full time step contains half-step velocity values.

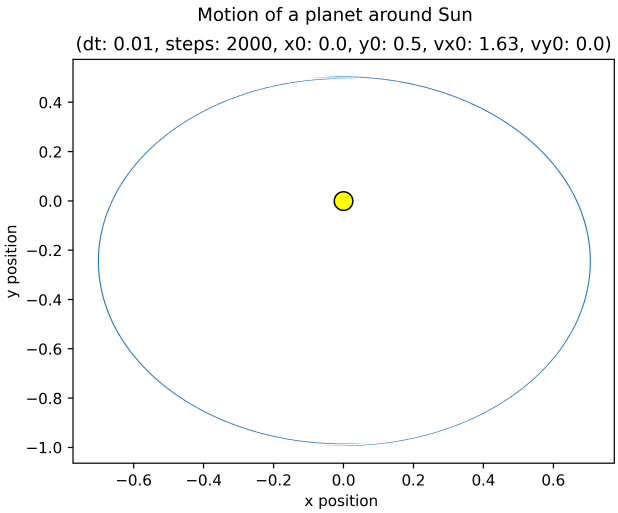
5.2. Iteration 2 - Fine time-step

The simulation was initialized with $r_0 = 0.5$, $a_x(0) = -4.0$, $a_y(0) = -0.0$, $v_x(e/2) = -0.2$, and $v_y(e/2) = 1.63$. Table 2 presents the time evolution data. Note that the second row after each full time step contains the half-step velocity values.

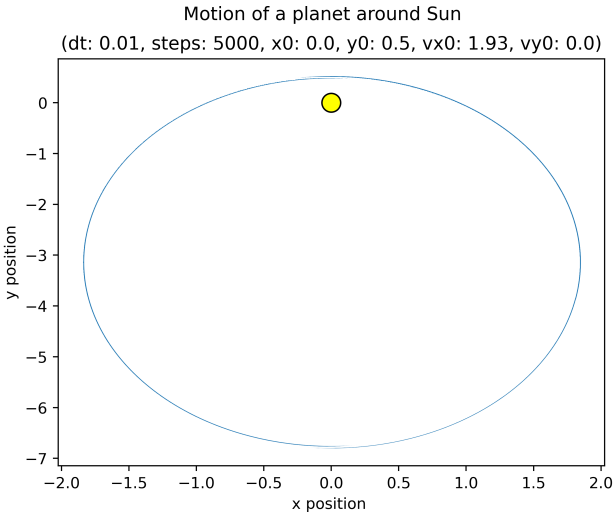
Table 2. Iteration 2 with dt=0.01

Time	x	y	a_x	a_y	v_x	v_y	r
0.000	0.500	0.000	-4.000	-0.000	0.000	1.630	0.500
					-0.020	1.630	
0.010	0.500	0.016	-3.997	-0.130	-0.040	1.629	0.500
					-0.060	1.629	
0.020	0.499	0.033	-3.987	-0.260	-0.080	1.627	0.500
					-0.100	1.626	
0.030	0.498	0.049	-3.972	-0.389	-0.120	1.624	0.501
					-0.140	1.622	
0.040	0.497	0.065	-3.950	-0.517	-0.159	1.620	0.501
					-0.179	1.617	
0.050	0.495	0.081	-3.921	-0.644	-0.199	1.614	0.502
					-0.218	1.611	
0.060	0.493	0.097	-3.887	-0.768	-0.238	1.607	0.502
					-0.257	1.603	
0.070	0.490	0.113	-3.848	-0.890	-0.276	1.598	0.503
					-0.296	1.594	
0.080	0.487	0.129	-3.802	-1.009	-0.315	1.589	0.504
					-0.334	1.584	
0.090	0.484	0.145	-3.752	-1.125	-0.352	1.578	0.505
					-0.371	1.573	
0.100	0.480	0.161	-3.696	-1.238	-0.390	1.566	0.506
					-0.408	1.560	
0.110	0.476	0.176	-3.636	-1.348	-0.426	1.554	0.508
					-0.444	1.547	
0.120	0.472	0.192	-3.571	-1.453	-0.462	1.540	0.509
					-0.480	1.532	
0.130	0.467	0.207	-3.502	-1.555	-0.498	1.525	0.511
					-0.515	1.517	
0.140	0.462	0.222	-3.429	-1.652	-0.532	1.508	0.513
					-0.550	1.500	
0.150	0.456	0.237	-3.353	-1.745	-0.566	1.491	0.514
					-0.583	1.483	
0.160	0.450	0.252	-3.273	-1.833	-0.599	1.474	0.516
					-0.616	1.464	
0.170	0.444	0.267	-3.191	-1.917	-0.632	1.455	0.518
					-0.648	1.445	
0.180	0.438	0.281	-3.106	-1.996	-0.663	1.435	0.520
					-0.679	1.425	
0.190	0.431	0.296	-3.019	-2.070	-0.694	1.415	0.523
					-0.709	1.405	
0.200	0.424	0.310	-2.930	-2.140	-0.724	1.394	0.525
					-0.738	1.383	
0.210	0.417	0.324	-2.839	-2.205	-0.752	1.372	0.527
					-0.767	1.361	

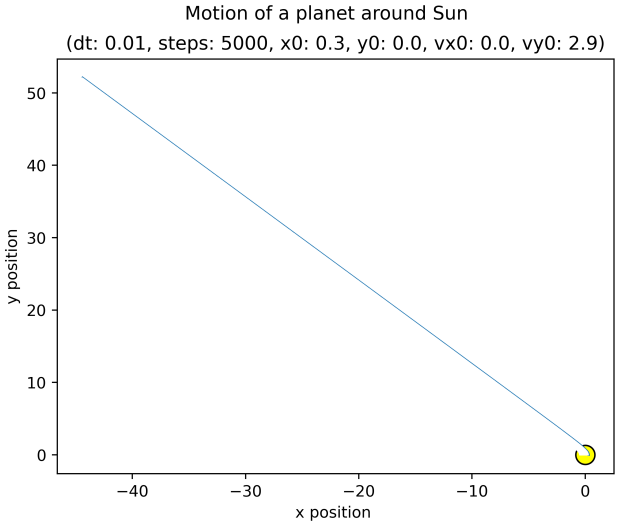
Note: The second row after each full step contains half-step velocity values.



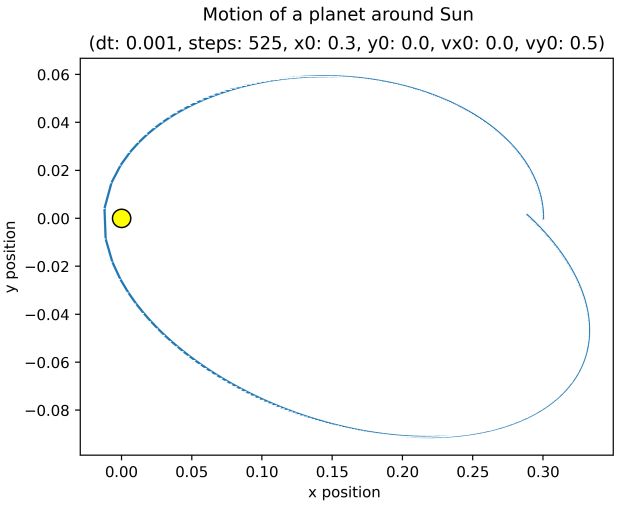
(a) Another example of a closed orbit at $y_0 = 0.5$



(b) Changing from a non-zero v_y to a non-zero v_x



(c) An example of an unbound/hyperbolic orbit by increasing velocity (positive total energy)



(d) Limitations of the 2nd-order leapfrog method leading to drifts

Figure 3. A montage of four different orbital scenarios demonstrating various initial conditions and outcomes.

References

[1] Feynman, R. P., Leighton, R. B., & Sands, M. (1964). *The Feynman Lectures on Physics: Volume I, Chapter 9*. Available at: https://www.feynmanlectures.caltech.edu/I_09.html (Accessed: March 10, 2025).

[2] Hairer, E., Lubich, C., & Wanner, G. (2006). *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, Berlin.

[3] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press, Cambridge.