FYS3150-Project3

Building a model for the solar system using ordinary differential equation

Chenghao Liu

October 23, 2018

1. Introduction

The purpose of this project is to simulate the solar system through algorithms, mainly through Newton's law of universal gravitation. In this project we need to use our own code to get results by running and simulating code. With the law of universal gravitation and Newton's second law, with knowing the orbital parameters of the major planets of the solar system, we can simulate by designing our own algorithms. The orbital parameters of each planet can be obtained on NASA or other websites. The first task of this project is to algorithmize the Earth-Sun system. By transforming the continuous differential equation into discrete segments, an algorithm can be designed to simulate. The second task is to write the code. Here I also use the dynamic drawing function of matplotlib to make the image clearer. The third task is to run the program, save the data using a list, and draw an X-T diagram, a Y-T diagram. The fourth task is to find the escape velocity and change the inverse square law of the law of universal gravitation to see the situation under different coefficients. The fifth task is to add Jupiter to simulate a three-body system. The sixth task is to simulate all major planetary systems in the solar system. The seventh task is to add a relativistic term to the force of Mercury to study the impact of relativity on Mercury's orbit.

2. Method

Here I chose Python for the simulation of this project because python's numpy and matplotlib can be used to simulate 3D dynamic images and save data for data analysis. I used my own algorithm to simulate the solar system. The whole algorithm is like this. At such a large scale of the solar system, I chose a small differential time. At this differential time, the motion of the planet is considered to be a uniform linear motion because the motion is so small compared to universe scale, that is to say, the planet travels in a straight line during this differential time, and the speed does not change. At the end of this time, the speed is increased by an a*t, that

is, the acceleration multiplied by the differential time. And then start the next period of differential movement. This is also the idea of calculus. Under this algorithm, we can use a for or while loop, through the continuous loop, the planet can move little by little, and finally equivalent to calculus roughly, forming the orbit we simulate.

Here for example, the sun, 1000 times the mass of Jupiter, and the earth, which is three-body movement:

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import math
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.image as img

G = 6.67*10**(-11)

m1 = 6*10**24 #earth
m2 = 2*10**30 #sun
m3 = 1.9*1000*10**27 #jupiter

#m1
x10 = -149597870000
y10 = 0
z10 = 0
p10 = 0
q10 = 29783
r10 = 0
#m2
x20 = 0
y20 = 0
z20 = 0
p20 = 0
q20 = 0
r20 = 0
#m3
x30 = 778547200000
y30 = 0
z30 = 0
p30 = 0
q30 = -13070
r30 = 0


dt = 2000
n = 1000000
```

```
#initialize the position
x1 = [0]
y1 = [0]
z1 = [0]

x2 = [0]
y2 = [0]
z2 = [0]

x3 = [0]
y3 = [0]
z3 = [0]

x1[0] = x10 #set up initial position
y1[0] = y10
z1[0] = z10

x2[0] = x20
y2[0] = y20
z2[0] = z20

x3[0] = x30
y3[0] = y30
z3[0] = z30

#initialzie the speed, create a list
p1 = [0]
q1 = [0]
r1 = [0]

p2 = [0]
q2 = [0]
r2 = [0]

p3 = [0]
q3 = [0]
r3 = [0]


#loop
i = 0
while i < n-1:
    #three distances
    S12 = math.sqrt((x2[i]-x1[i])**2+(y2[i]-y1[i])**2+(z2[i]-z1[i])**2)
    S13 = math.sqrt((x3[i]-x1[i])**2+(y3[i]-y1[i])**2+(z3[i]-z1[i])**2)
    S23 = math.sqrt((x2[i]-x3[i])**2+(y2[i]-y3[i])**2+(z2[i]-z3[i])**2)


    x1.append(x1[i]+p1[i]*dt)
    y1.append(y1[i]+q1[i]*dt)
    z1.append(z1[i]+r1[i]*dt)
    x2.append(x2[i]+p2[i]*dt)
    y2.append(y2[i]+q2[i]*dt)
    z2.append(z2[i]+r2[i]*dt)
    x3.append(x3[i]+p3[i]*dt)
    y3.append(y3[i]+q3[i]*dt)
    z3.append(z3[i]+r3[i]*dt)

    p1.append(dt*(G*m3*(x3[i+1]-x1[i+1])/((S13)**3)+G*m2*(x2[i+1]-x1[i+1])/((S12)**3))+p1[i])
    q1.append(dt*(G*m3*(y3[i+1]-y1[i+1])/((S13)**3)+G*m2*(y2[i+1]-y1[i+1])/((S12)**3))+q1[i])
    r1.append(dt*(G*m3*(z3[i+1]-z1[i+1])/((S13)**3)+G*m2*(z2[i+1]-z1[i+1])/((S12)**3))+r1[i])
    p2.append(dt*(G*m1*(x1[i+1]-x2[i+1])/((S12)**3)+G*m3*(x3[i+1]-x2[i+1])/((S23)**3))+p2[i])
    q2.append(dt*(G*m1*(y1[i+1]-y2[i+1])/((S12)**3)+G*m3*(y3[i+1]-y2[i+1])/((S23)**3))+q2[i])
    r2.append(dt*(G*m1*(z1[i+1]-z2[i+1])/((S12)**3)+G*m3*(z3[i+1]-z2[i+1])/((S23)**3))+r2[i])
    p3.append(dt*(G*m1*(x1[i+1]-x3[i+1])/((S13)**3)+G*m2*(x2[i+1]-x3[i+1])/((S23)**3))+p3[i])
    q3.append(dt*(G*m1*(y1[i+1]-y3[i+1])/((S13)**3)+G*m2*(y2[i+1]-y3[i+1])/((S23)**3))+q3[i])
    r3.append(dt*(G*m1*(z1[i+1]-z3[i+1])/((S13)**3)+G*m2*(z2[i+1]-z3[i+1])/((S23)**3))+r3[i])

    #next loop
    i += 1
```

```
import mpl_toolkits.mplot3d.axes3d as p3

#make plot
def update_lines(num,datalines,lines):
    for line,data in zip(lines,datalines):
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines
fig = plt.figure()
ax = p3.Axes3D(fig)

#data turple
data=[np.array([x1,y1,z1])[:,0:1000000:100],
      np.array([x2,y2,z2])[:,0:1000000:100],
      np.array([x3,y3,z3])[:,0:1000000:100]]

lines =[ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]

#make plot
ax.set_xlim3d([-10**12,10**12])
ax.set_xlabel('X')
ax.set_ylim3d([-10**12,10**12])
ax.set_ylabel('Y')
ax.set_zlim3d([-10**12,10**12])
ax.set_zlabel('Z')
ax.set_title('Simulation on three-body')

#figure show
line_ani = animation.FuncAnimation(fig, update_lines,fargs = (data,lines),interval =1,blit = False)
plt.show()
```

Figure 1 Code for example

First set initial parameters such as gravitational constant, planet mass, position, initial velocity, etc. Then, at the beginning of each cycle, the relative position between the stars is calculated. With the distance, the gravitation can be calculated to obtain the acceleration, and then the idea of the above algorithm is used to gradually integrate the differential time to obtain the changes of the orbit and speed of the planet and save data in the list. With the data in the list, we can use python's matplotlib to draw a figure to perform 3D dynamic figure of the image, so that we can clearly see the orbital changes of the planet, and successfully simulate the solar system.

3. Implementation

3a

First, we all know the equation that the force between sun and earth is followed relation:

$$F_G = \frac{GM_{\odot}M_{\mathrm{Earth}}}{r^2},$$

We use the velocity Verlet method and Euler's forward method to set up our algorithm.

First, there is a velocity for the earth's orbit, and we regard sun as a fixed point which don't move. Second, for a very short time t, We regard the movement of the earth as a uniform linear motion, the velocity is a constant, but at the end of t, the velocity will increase a*t, a is

the acceleration. Third, we can put the change of position, velocity into a list and save it as our data. And use this data, we can simulate the movement of sun-earth system.

3b

As the idea of my algorithm above, next we can design the program code. The main part code of the whole program is as shown in the figure.

```
#loop
i = 0
while i < n-1:
    #distance between sun and earth
    S12 = math.sqrt((x2[i]-x1[i])**2+(y2[i]-y1[i])**2+(z2[i]-z1[i])**2)


    #regard the movement of the earth as a uniform linear motion
    x1.append(x1[i]+p1[i]*dt)
    y1.append(y1[i]+q1[i]*dt)
    z1.append(z1[i]+r1[i]*dt)
    x2.append(x2[i]+p2[i]*dt)
    y2.append(y2[i]+q2[i]*dt)
    z2.append(z2[i]+r2[i]*dt)

    #at the end of t, the velocity will increase a*t
    p1.append(dt*(G*m2*(x2[i+1]-x1[i+1])/((S12)**3))+p1[i])
    q1.append(dt*(G*m2*(y2[i+1]-y1[i+1])/((S12)**3))+q1[i])
    r1.append(dt*(G*m2*(z2[i+1]-z1[i+1])/((S12)**3))+r1[i])
    p2.append(dt*(G*m1*(x1[i+1]-x2[i+1])/((S12)**3))+p2[i])
    q2.append(dt*(G*m1*(y1[i+1]-y2[i+1])/((S12)**3))+q2[i])
    r2.append(dt*(G*m1*(z1[i+1]-z2[i+1])/((S12)**3))+r2[i])

    #next loop
    i += 1

import mpl_toolkits.mplot3d.axes3d as p3

#make motion plot
def update_lines(num, datalines, lines):
    for line, data in zip(lines, datalines):
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines
fig = plt.figure()
ax = p3.Axes3D(fig)

#3D data turple
data=[np.array([x1,y1,z1])[:,0:1000000:100],
      np.array([x2,y2,z2])[:,0:1000000:100]]

lines =[ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]

#make plot
ax.set_xlim3d([-2*10**11,2*10**11])
ax.set_xlabel('X')
ax.set_ylim3d([-2*10**11,2*10**11])
ax.set_ylabel('Y')
ax.set_zlim3d([-2*10**11,2*10**11])
ax.set_zlabel('Z')
ax.set_title('Simulation on earth-sun')

#figure show
line_ani = animation.FuncAnimation(fig, update_lines,fargs = (data,lines),interval =1,blit = False)
plt.show()
```

Figure 2 Code for task b

Since the quality of the sun is too large relative to the Earth, the positional change of the sun is basically negligible. First, set the parameters of the Earth and the Sun, such as the mass position and velocity, and then calculate the acceleration method to obtain the change of the velocity and the position change in each differential time, so that we can accumulate the simulated image and data of the Earth-Sun system step by step and obtain the result. Through the simulation, the running program can obtain the dynamic orbit map of the Earth,

but it is kind of slow because all the data is calculated and then stored in the list, and then plotted, instead of drawing while calculating. The simulated image is shown in the figure. it is static because it can only put the image. But in fact, it is the dynamic image that is displayed after running the program.

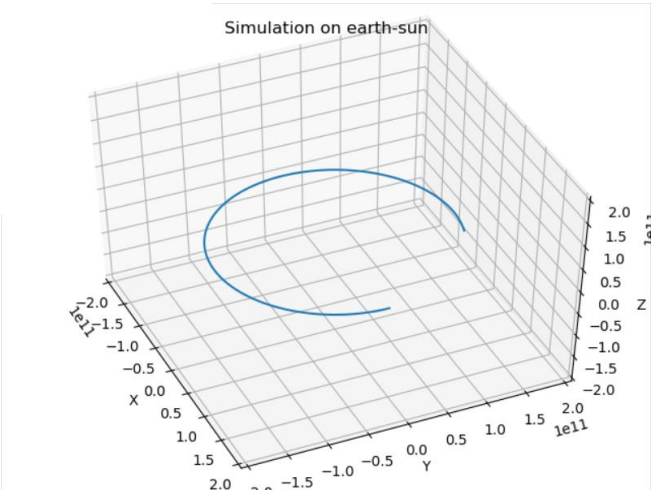

Figure 3 Simulation image of sun-earth

3c

After constant experimentation and simulation, I found that when dt is small, the theoretical orbit is more accurate than when dt is large. But consider the Python's floating precision and the scale of the solar system, the change is almost equivalent. And the image update speed is slower when the dt is quite large to simulating the dynamic graph, for example, if the dt is 4000 seconds or more, the orbit will be obviously inaccurate, so I set the dt of the simulation to 500s. and the stability of the planetary orbit obtained by the simulated image is also quite good.

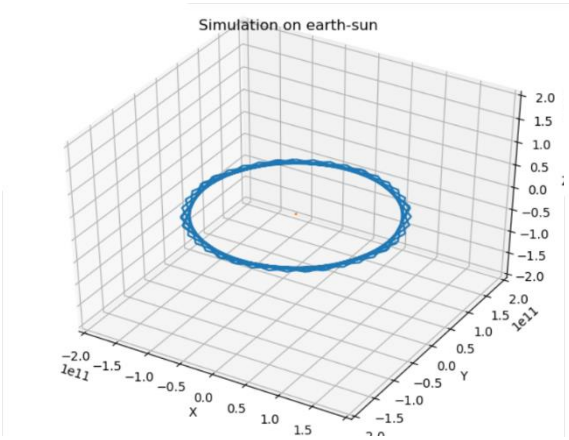

Figure 5 dt at 50000s

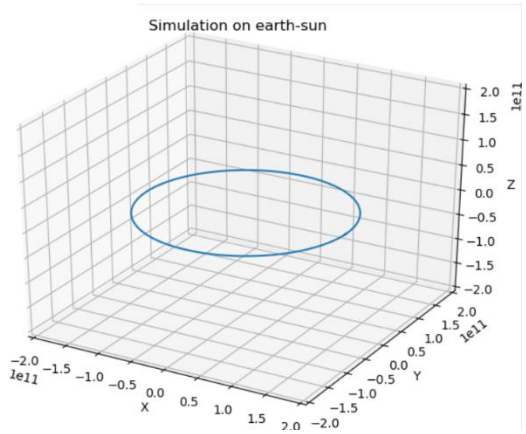

Figure 4 dt at 500s

After the simulation, the x, y position of the planet, the change of x, y speed with time are

recorded in the list. We extract the list and use the drawing function of matplotlib again to get the X-T diagram, Y-T diagram, the code and final drawing. My image is as shown below

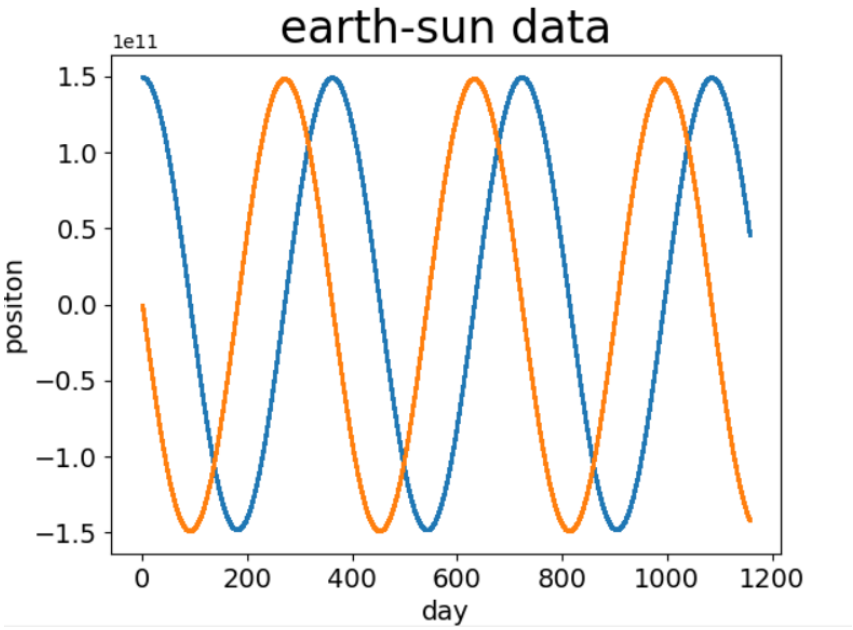

Figure 6 X-T diagram(blue), Y-T diagram(orange)

3d

In this section, I consider this star to be the mass of the earth because mass is not related to the result in this part, so any mass is suitable for this part. Then set the speed of the star away from the sun and see if it reverses at the end of the loop. I used a step of 1000m/s to increase the speed step by step. I found that there is an escape speed between 42000 and 43000. Through mathematical calculations, the escape velocity of the solar system at this position is 42170m/s, which means that the experimental results are in good agreement with the mathematical calculations, but since the code is only a roughly simulated relationship, it is not possible to get the exact solution directly, you can get the exact solution by letting the code calculate the formula. The simulation diagram of the escape is shown in the figure.

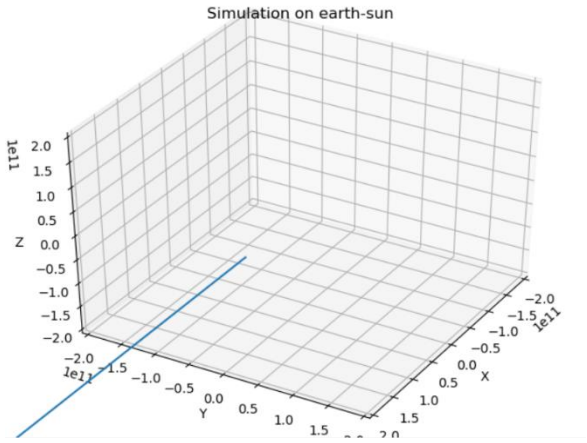

Figure 7 Escape velocity simulation

We can also change the gravitational force and modify the original inverse square law to become the inverse cube law or other. Here I use the relationship between the integral of the modified gravitational formula and the kinetic energy to obtain the formula of the speed and the new universal law of gravity. Using code to perform calculations and simulations, and finally obtaining a change in the escape velocity between beta and 2 to 3

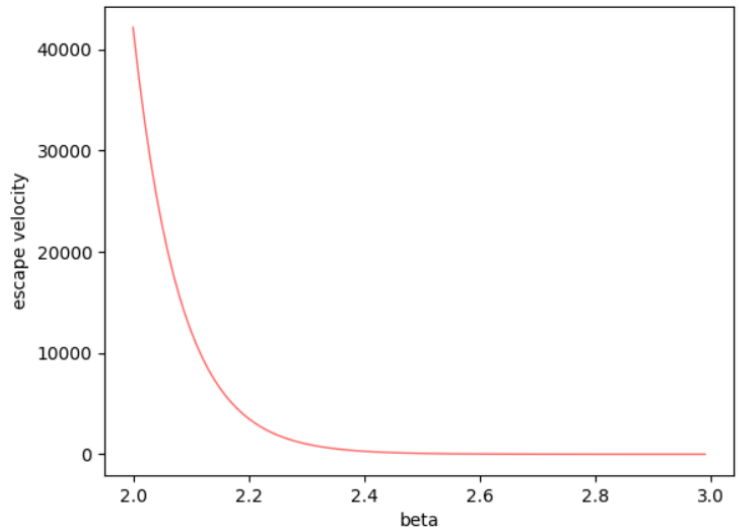

Figure 8 Escape velocity under different beta

3e

In this part of the project, which is required to simulate is the three-body system, in which there are the sun, Jupiter, and the Earth. Using the same algorithm to simulate the three-body system, you can get its dynamic images and data. We can observe that when Jupiter is in its original mass, the orbit is stable, and although the Earth is slightly disturbed, the orbital changes are not obvious.

By increasing the mass of Jupiter ten times, it can be found that the orbit is not stable a little, and there is a slight deviation from the orbit after one week. This is caused by the huge mass of Jupiter, but in the end it is similar to dynamic balance status.

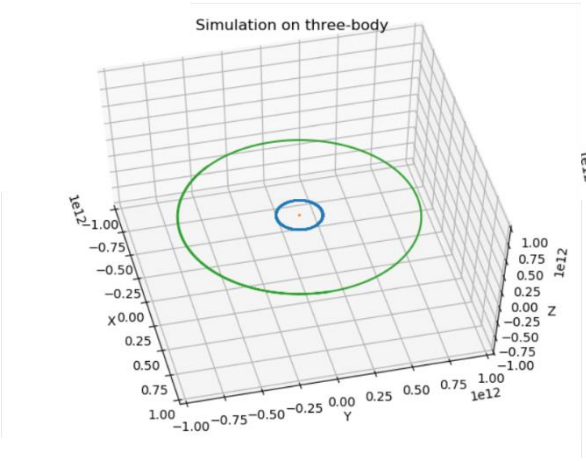

Figure 10 three-body system when original mass

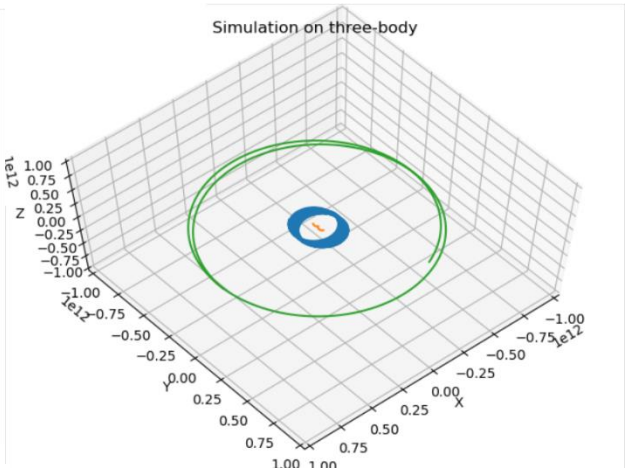

Figure 9 10 times of Jupiter mass

By increasing the mass of Jupiter by a hundred times, it can be seen that there is a significant deviation from the orbit, the sun begins to drift, the Earth's orbit is drifting with the sun, and the Jupiter orbit is also greatly deviated.

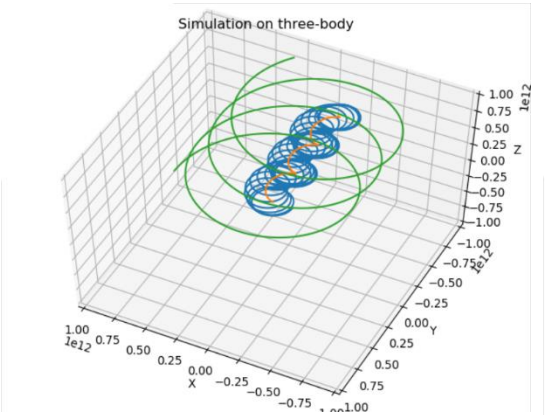

Figure 12 100 times of Jupiter mass

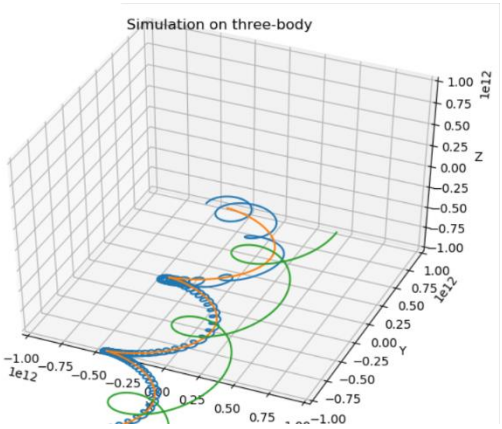

Figure 11 1000times of Jupiter mass

The mass of Jupiter is increased by a thousand times. At this time, the orbital deviation of the three-body system becomes more and more obvious. In a very short time, the orbit becomes completely different from before, the track becomes unstable, and then there is no collision. A rough dynamic balance was reached.

3f

Since the mass of the sun accounts for more than 95% of the entire solar system, we can still

think that the sun is fixed at such a large mass. Because of its large mass, the influence of the momentum of other planets on it is almost negligible. To simulate the entire solar system, we need to get the orbital parameters of all eight planets and add them to the code, which means we have 48 lists to store the position and velocity of the eight planets. When the entire code runs, the speed will be very slow due to the large data. This time we don't simulate the dynamic graph, but only draw its orbit, so the method is still feasible. After adding all the parameters of the eight planets to the code and rewrite the code, we can get the figure.

```
#main loop
i = 0
while i < n-1:
    #distance
    S12 = x20
    S13 = x30
    S14 = x40
    S15 = x50
    S16 = x60
    S17 = x70
    S18 = x80
    S19 = x90

    #same as the code before
    x2.append(x2[i]+p2[i]*dt)
    y2.append(y2[i]+q2[i]*dt)

    x3.append(x3[i]+p3[i]*dt)
    y3.append(y3[i]+q3[i]*dt)

    x4.append(x4[i]+p4[i]*dt)
    y4.append(y4[i]+q4[i]*dt)

    x5.append(x5[i]+p5[i]*dt)
    y5.append(y5[i]+q5[i]*dt)

    x6.append(x6[i]+p6[i]*dt)
    y6.append(y6[i]+q6[i]*dt)

    x7.append(x7[i]+p7[i]*dt)
    y7.append(y7[i]+q7[i]*dt)

    x8.append(x8[i]+p8[i]*dt)
    y8.append(y8[i]+q8[i]*dt)

    x9.append(x9[i]+p9[i]*dt)
    y9.append(y9[i]+q9[i]*dt)

    #same as the code before, only more planets
    p2.append(dt*(G*m1*(x1[i+1]-x2[i+1])/((S12)**3))+p2[i])
    q2.append(dt*(G*m1*(y1[i+1]-y2[i+1])/((S12)**3))+q2[i])

    p3.append(dt*(G*m1*(x1[i+1]-x3[i+1])/((S13)**3))+p3[i])
    q3.append(dt*(G*m1*(y1[i+1]-y3[i+1])/((S13)**3))+q3[i])

    p4.append(dt*(G*m1*(x1[i+1]-x4[i+1])/((S14)**3))+p4[i])
    q4.append(dt*(G*m1*(y1[i+1]-y4[i+1])/((S14)**3))+q4[i])

    p5.append(dt*(G*m1*(x1[i+1]-x5[i+1])/((S15)**3))+p5[i])
    q5.append(dt*(G*m1*(y1[i+1]-y5[i+1])/((S15)**3))+q5[i])

    p6.append(dt*(G*m1*(x1[i+1]-x6[i+1])/((S16)**3))+p6[i])
    q6.append(dt*(G*m1*(y1[i+1]-y6[i+1])/((S16)**3))+q6[i])

    p7.append(dt*(G*m1*(x1[i+1]-x7[i+1])/((S17)**3))+p7[i])
    q7.append(dt*(G*m1*(y1[i+1]-y7[i+1])/((S17)**3))+q7[i])

    p8.append(dt*(G*m1*(x1[i+1]-x8[i+1])/((S18)**3))+p8[i])
    q8.append(dt*(G*m1*(y1[i+1]-y8[i+1])/((S18)**3))+q8[i])

    p9.append(dt*(G*m1*(x1[i+1]-x9[i+1])/((S19)**3))+p9[i])
    q9.append(dt*(G*m1*(y1[i+1]-y9[i+1])/((S19)**3))+q9[i])

    #next loop
    i += 1
```

Figure 13 main loop code for task f

Run the code and get the orbital simulation map of the eight planets. The final result is shown in the figure. It can be seen that the simulation results are still very good to simulate the planetary motion orbit of the entire solar system.

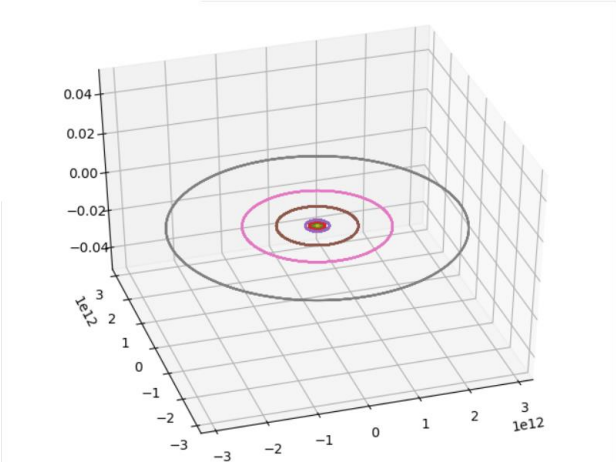

Figure 14 Simulation image for all eight planets

3g

We need to add a relativistic term to Mercury's universal gravitational formula. Since the l term is numerically equal to v*r, we can approximate r, and finally the entire gravitational formula has one more item (1+3*v^2/c^2). We use this new universal gravitational formula to substitute the code for the orbital simulation of Mercury and input the starting parameters of the orbit of Mercury to finally obtain a simulated image of the orbit of Mercury. The project requires a simulation time of one century. I set each differential time to 0.1h and control the total number of cycles to reach the requirements of a century although the speed will become slow. Finally, I got the simulation of Mercury orbit. We can Seeing that there is a little change in the orbit of Mercury in one century, and we have obtained data on the orbit of Mercury, we can know the x, y position of Mercury at different times, so that we can find the point of Perihelion to analyze the change of the tan value of the perihelion point. Here I designed an if judgment to determine whether the distance from the sun is smaller than the front and rear positions to determine whether it is a perihelion point, so that the data of the perihelion point is obtained and saved in the list. The code is as shown in the figure.

```
tan_angle = [0]
#loop
i = 0
while i < n-1:
    #distance between sun and mercury
    S12 = math.sqrt((x2[i]-x1[i])**2+(y2[i]-y1[i])**2+(z2[i]-z1[i])**2)


    #regard the movement of the earth as a uniform linear motion
    x1.append(x1[i]+p1[i]*dt)
    y1.append(y1[i]+q1[i]*dt)
    z1.append(z1[i]+r1[i]*dt)
    x2.append(x2[i]+p2[i]*dt)
    y2.append(y2[i]+q2[i]*dt)
    z2.append(z2[i]+r2[i]*dt)

    radius = x1[i]**2+y1[i]**2
    radius_after = x1[i+1]**2+y1[i+1]**2
    radius_before =x1[i-1]**2+y1[i-1]**2

    if radius < radius_after and radius <radius_before:
        tan_angle.append(x1[i]/y1[i])
    else:
        pass

    #at the end of t, the velocity will increase a*t, but now we have a phase for relativity
    #which is 1+3*(p1[i]**2+q1[i]**2)/(6*10**18)
    p1.append(dt*((1+3*(p1[i]**2+q1[i]**2)/(6*10**18))*G*m2*(x2[i+1]-x1[i+1])/((S12)**3))+p1[i])
    q1.append(dt*((1+3*(p1[i]**2+q1[i]**2)/(6*10**18))*G*m2*(y2[i+1]-y1[i+1])/((S12)**3))+q1[i])
    r1.append(dt*((1+3*(p1[i]**2+q1[i]**2)/(6*10**18))*G*m2*(z2[i+1]-z1[i+1])/((S12)**3))+r1[i])
    p2.append(dt*((1+3*(p2[i]**2+q2[i]**2)/(6*10**18))*G*m1*(x1[i+1]-x2[i+1])/((S12)**3))+p2[i])
    q2.append(dt*((1+3*(p2[i]**2+q2[i]**2)/(6*10**18))*G*m1*(y1[i+1]-y2[i+1])/((S12)**3))+q2[i])
    r2.append(dt*((1+3*(p2[i]**2+q2[i]**2)/(6*10**18))*G*m1*(z1[i+1]-z2[i+1])/((S12)**3))+r2[i])

    #next loop
    i += 1



import mpl_toolkits.mplot3d.axes3d as p3

#make motion plot
def update_lines(num, datalines, lines):
    for line, data in zip(lines, datalines):
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines
fig = plt.figure()
ax = p3.Axes3D(fig)

#3D data turple
data=[np.array([x1, y1, z1])[:, 0:1000000:100],
      np.array([x2, y2, z2])[:, 0:1000000:100]]

lines =[ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]

#make plot
ax.set_xlim3d([-10**11, 10**11])
ax.set_xlabel('X')
ax.set_ylim3d([-10**11, 10**11])
ax.set_ylabel('Y')
ax.set_zlim3d([-10**11, 10**11])
ax.set_zlabel('Z')
ax.set_title('Simulation on mercury-sun')

#figure show
line_ani = animation.FuncAnimation(fig, update_lines, fargs = (data, lines), interval =1, blit = False)
plt.show()
```

Figure 15 main code for task g

Finally, we obtained the data of all the perihelion points in a century. We can use the drawing function of matplotlib to get the change of the tan value of the perihelion. The resulting change image is shown in the figure. It can be found that the theory of relativity does affect the precession of Mercury. The absolute value of the tan value of the perihelion is continuously increased. So we can judge from the data and explain that Mercury's orbit is affected by relativity.

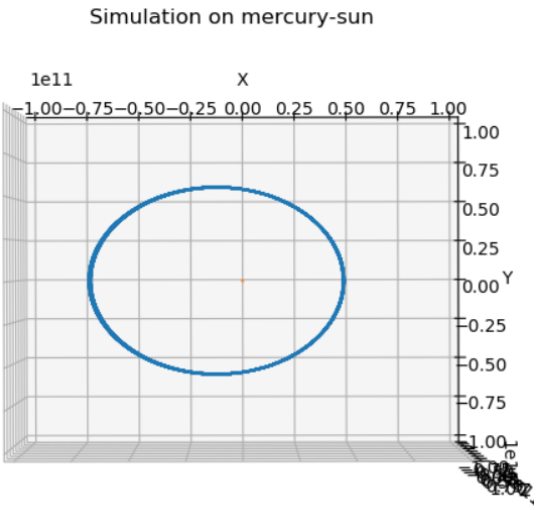

Figure 17 Simulation of Mercury oribit

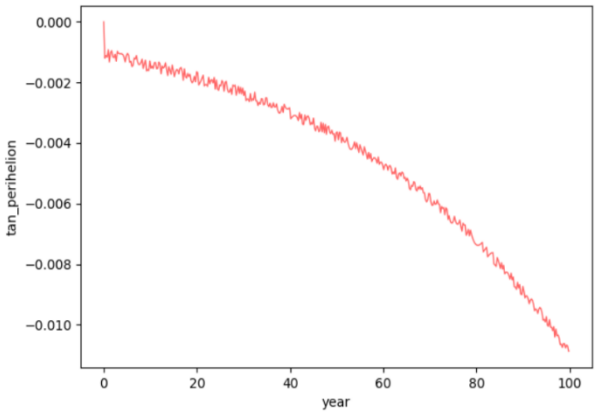

Figure 16 Change of Perihelion tan values by time

4. Results

3a

We get the algorithm of the project 3a. First, there is a velocity for the earth's orbit, and we regard sun as a fixed point which don't move. Second, for a very short time dt, We regard the movement of the earth as a Uniform linear motion, the velocity is a constant, but at the end of t, the velocity will increase a*t, a is the acceleration. Third, we can put the change of position, velocity into a list and save it as our data and use this data, we can simulate the movement of sun-earth system.

3b

We got a simulated image of the Sun-Earth system, as shown below

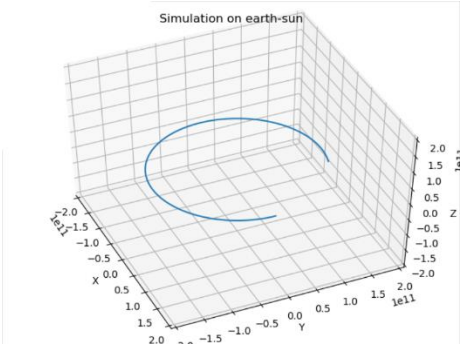

Figure 18 Simulation image of Sun-Earth system

3c

By continuously debugging dt, we obtained the simulated image when the track was stable. And we finally draw the X-T diagram and the Y-T diagram.

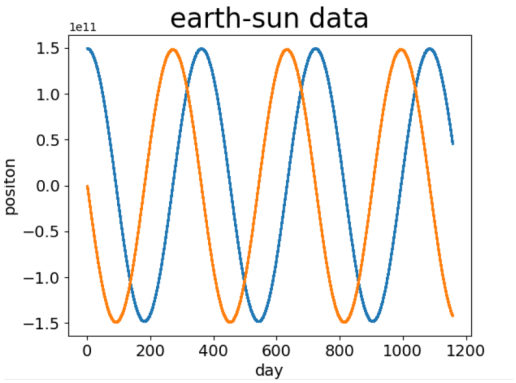

Figure 19 X-T diagram(blue) and the Y-T diagram(orange)

3d

The debugging results of the program show that the escape speed is between 42000m/s and 43000m/s, while the mathematical calculation shows that the escape speed is 42170m/s, which also meets the running result of the program. After that, the beta coefficients were modified to obtain images of the change in escape velocity at different beta coefficients.

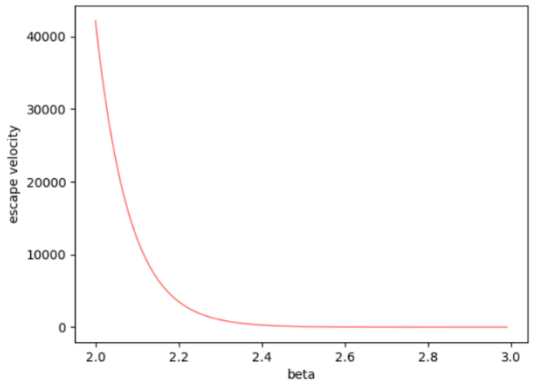

Figure 20 Escape velocity at different beta

3e

Under the four different Jupiter masses, we simulated the three-body system of Sun-Earth-Jupiter and found that Jupiter under different masses have an impact on the Earth and the Sun's orbit. Among them, when Jupiter has the highest mass, the effects of the earth and the sun are most obvious.

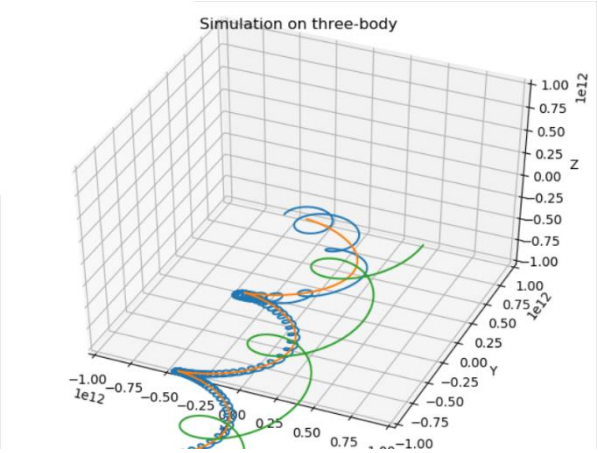

Figure 21 Three-body system under 1000 times of Jupiter mass

3f

By simulating the parameters of the eight major planets, the orbital data of the eight planets were obtained and the orbital images of the eight planets around the sun were drawn.

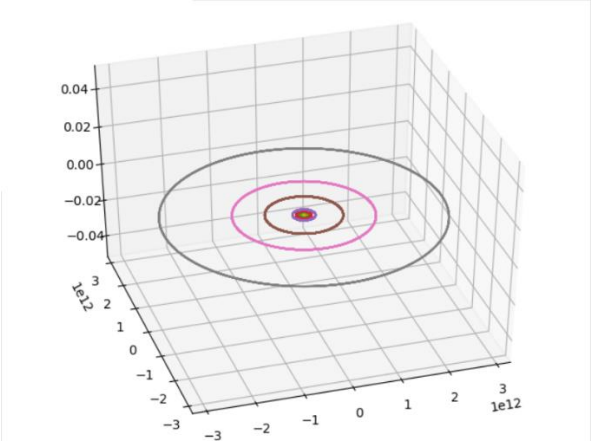

Figure 22 Simulation of all eight planets orbits

3g

By adding a relativity term to Mercury's universal gravitation, the change data of Mercury orbit was successfully obtained, and the change of tan value over time was calculated. The image of Mercury orbit and the change of tan value with time were successfully drawn.

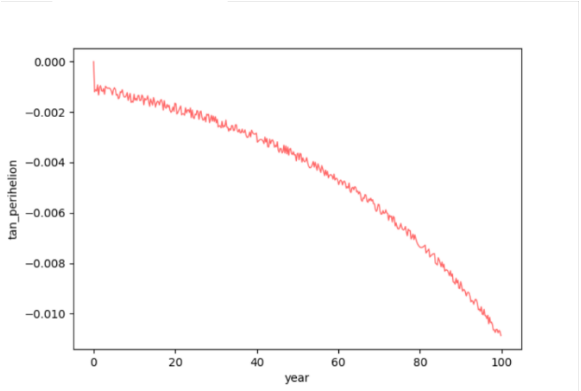

Figure 23 Perihelion tan values changes in one century

5. Concluding Remarks

This project successfully completed various simulation experiments and simulated the movement of the entire solar system. The results were good, and we succeeded in obtaining Earth-Sun orbital simulations, three-body orbital simulations, eight major planetary orbital simulations of the solar system, and Mercury orbital simulations. These data are accurate to some extent. However, because this algorithm uses the method of differential time, it is not calculus, so there is a certain degree of error, which may affect the data accuracy of the tan value of the orbital point of Mercury, but since dt is very small, this effect of the value relative to its own relativity is also a small amount that can be ignored, so the result can still be considered accurate. If you want to further reduce this error, you can also continue to shrink dt, but this will also bring the program operation speed slower.

6. Reference

http://ssd.jpl.nasa.gov/horizons.cgi#top

Hjort-Jensen,M., 2015. Computational physics.

The program can be found in https://github.com/lch172061365/Computational-Physics.git