THREE-BODY PROBLEM AND SIMULATION

SHENGYUAN WANG, ORIANNA WANG, ZIYI WANG

ABSTRACT. The three-body problem is a fundamental model in astrophysics that studies the law-of-motion problem of three objects that are considered as masses under the gravitation influence. This paper utilizes classical mechanical equations and the advantages of computers in terms of numerical operations to iteratively solve the three-body problem and determine its bound state. In the model, the three objects are analyzed by force; equations are listed using Newton's second law and law of universal gravitation; and the Python program is written according to the equations for the iterative solution, which calculates the position and velocity of the three objects at a specific time under different circumstances. In addition, in this paper, the matplotlib and Python animation are utilized to investigate and display the problem. We designed a simple and friendly interface, and the position and velocity of the objects are also shown in the axes in real-time through the animation to perform astrophysical experiments on the computer.

1. INTRODUCTION

The general three-body classical problem is one of the oldest and most complex problems in physics and mathematics. In short, the task is to study and model the motion of a three-body system in space under the influence of pairwise[1] interactions of bodies in accordance with Newton's theory of gravitation. Solutions to this problem require that future and past motions of the bodies be uniquely determined based on their present velocities and positions.[5] In general, the motions of the bodies take place in three dimensions, and no restrictions will be added on their masses nor on the initial conditions, referring to the general three-body problem. At the first glance, the complexity of the problem is not obvious, since two-body systems have been well-known closed-form solutions given in terms of elementary functions. However, adding one extra body makes the system too complicated to obtain similar types of solutions. In the past, many physicists, astronomers and mathematicians attempted unsuccessfully to find close-form solutions to the three-body problem. Such solutions do not exist because the three bodies are generally unpredictable.

Since the difficulties in solving the three-body problem lie in its uncertainty and unpredictability, one can consider a third type of approach based on force analysis and classical mechanical equations, taking advantage of the iterative computation of computers to find numerical solutions.

Date: May 28, 2023.

[1]in pair

2. Model Assumption

The restricted three-body problem assumes that the mass of one of the bodies is negligible, including the interaction forces between it and the other two bodies. So the problem is simplified as a two-body problem with their orbitals being conic sections with the center of mass on one of their focuses. We have four kinds of restricted three-body problems respectively: the circular restricted three-body problem, the elliptical restricted three-body problem, the parabolic restricted three-body problem, and the hyperbolic restricted three-body problem.

The circular restricted three-body problem is the special case in which two of the bodies are in circular orbits around their common center of mass, and the third mass is small and moves in the same plane. In the circular problem, with respect to a rotating reference frame, the two co-orbiting bodies are stationary, and the third has five equilibrium points. Three of them are collinear[2] with the masses and are unstable. The other two are located on the third vertex of both equilateral triangles of which the two bodies are the first and second vertices. For a sufficiently small mass ratio, the triangular equilibrium points are stable. The five equilibrium points are known as the Lagrange points in the circular restricted three-body problem.[7] In addition, it can be quite helpful to consider the effective potential energy[3].

3. Notations

Table 1. Variable Description

Symbol	Definition
M_i	Mass of the body i
R_i	The position of body i with respect to the origin of inertial Cartesian coordinate system
R_c	The center of mass
r_{ij}	The position of body i with respect to body j
t	Time
E_{kin}	Kinetic energy
E_{pot}	Potential energy
p_{ki}	Momentum of body i in direction of k
q_{ki}	The position of body P_i in direction of k

[2]lying on or passing through the same straight line.

[3]The effective potential (also known as effective potential energy) combines multiple, perhaps opposing, effects into a single potential. It may be used to determine the orbits of planets (both Newtonian and relativistic) and to perform semi-classical atomic calculations, and often allows problems to be reduced to fewer dimensions.

4. Model Building

FIGURE 1. The last day of 2019, it was surprised that "three suns overhead". The three-body civilization is really coming?

4.1. **Basic formulation.** In the general three-body problem, three bodies will move in three-dimensional space under their mutual gravitational interactions. Throughout this paper, we will use Newton's theory of gravity to describe the gravitational interactions between the three bodies. To fully solve the general three-body problem, it requires the past and future motions of the bodies to be determined by their present positions and velocities.[1] We denote the three masses by M_i, where $i = 1, 2, 3$, and the positions with respect to the origin of inertial Cartesian coordinate system by the vector R_i, and define the position of one body with respect to another by $r_{ij} = R_i - R_j$, where $r_{ij} = -r_{ij}, j = 1, 2, 3$ and $i \neq j$. With the assumption that Newton's gravitational force is the only force acting upon the bodies, the resulting equations of motion will be

$$M_i \ddot{R}_i = G \sum_{j=1}^{3} \frac{M_i M_j}{r_{ij}^3} r_{ij} \tag{1}$$

where G is the universal gravitational constant.

4.2. **Integrals of Motion.** We can sum up the equation (1) over all three bodies. The result is

$$\sum_{i=1}^{3} M_i \ddot{R}_i = 0 \tag{2}$$

after integration, we can get

$$\sum_{i=1}^{3} M_i \dot{R}_i = C_1 \tag{3}$$

where C_1 is a constant, and one more integration yields

$$\sum_{i=1}^{3} M_i R_i = C_1 t + C_2 \tag{4}$$

with C_2 = constant.

Since the center of mass is defined as $R_c = \frac{\sum_{i=1}^{3} M_i R_i}{\sum_{i=1}^{3} M_i}$, the equation(4) determines the motion, and the equation(3) shows that it moves with a constant velocity. The two vectors C_1, C_2 are the integrals of the motion, and we will have six integrals of motion by taking the components of these vectors.

The conservation of angular momentum around the center of the coordinate system in the general three-body problem gives other integrals of motion. We choose to take a vector product of R_i with equation (2) and obtain

$$\sum_{i=1}^{3} M_i R_i \ddot{R}_i = 0 \tag{5}$$

which after integration will be

$$\sum_{i=1}^{3} M_i R_i \dot{R}_i = C_3 \tag{6}$$

where C_3 = constant. Hence, there are three more integrals of motion.

According to the conservation of the total energy of the system, we will add the integral of motion with the kinetic energy E_{kin} given by

$$E_{kin} = \frac{1}{2} \sum_{i=1}^{3} M_i \dot{R}_i^2 \tag{7}$$

$$E_{pot} = -\frac{G}{2} \sum_{i,j=1}^{3} \frac{M_i M_j}{r_{ij}} \tag{8}$$

where $i \neq j$, we have the total energy $E_{tot} = E_{kin} + E_{pot}$, which will be an constant to be also an integral of motion.

4.3. **Hamilton Formulation.** The standard formulation of the general three-body problem presented in the previous subsection is often replaced by the Hamilton formulation.

We consider the natural unit and introduce G = 1 into the equation (1) and (8). Moreover, we write $R_i = (R_{xi}, R_{yi}, R_{zi})$, where R_{xi}, R_{yi}, and R_{zi} are components if the vector R_i in the inertial Cartesian coordinate system and k = x, y, z. Using this notion , we can define the momentum, p_{ki}, as

$$p_{ki} = M_i \frac{dq_{ki}}{dt} \tag{9}$$

and the kinetic energy as

$$E_{kin} = \sum_{k,i=1}^{3} \frac{p_{ki}^2}{2M_i} \tag{10}$$

and introduce the Hamiltonian $H = E_{kin} + E_{pot}$ that allows us to write the equations of motion in the following Hamiltonian form

$$\frac{dq_{ki}}{dt} = \frac{\partial H}{\partial p_{ki}} \text{ and } \frac{dp_{ki}}{dt} = -\frac{\partial H}{q_{ki}} \tag{11}$$

We can set up two first-order ODEs for each one of three planets in each of three directions, which will make up a set of 18 first-order ODEs.

The equations of motion describing the general three-body problem (either equation (10) or equation (11)) can be further reduced by considering the general Hill problem[4], in which $M_1 >> M_2$ and $M_1 >> M_3$, however M_3 is not negligible when compared to M_2

4.4. **General Properties of Solutions of ODEs.** Solutions of the ODEs describing the general three-body problem can be represented by curves in 3D. In a special case, a solution can be a single point known also as a fixed point or as an equilibrium solution; a trajectory can either reach the fixed point or approach it symptomatically. And periodic orbits are centered around the fixed points, which are called stable points or stable centers. Moreover, if a trajectory spirals toward a fixed point or moves away from it, the point is called a spiral sink or spiral source.[6]

From a mathematical point of view, it requires solutions to ODEs existing and being unique. The existence can be either global when a solution is defined for any time in the past, present, and future, or local when it is defined only for a short period. The uniqueness of the solution means there is only one solution at each point. For given ODEs, the existence and uniqueness are typically stated by mathematical theorems. Let us consider the following first-order ODE $y'(x) = f(y(x), x)$, where $y' = \frac{dy}{dx}$, and the initial condition is $y(x_0) = y_0$.

[4]the study of the motion of two small and gravitationally interacting particles in a field of force not affected by the presence of these particles

5. Model Solving and System Simulation

5.1. **Programming and Interfacial Design.** First, we set up and initialize the masses, initial position, and initial velocity of planets. Then we model the evolution of the system by finding out the positions and velocity for one more step of δt. Finally, we draw the points of each second and wrap it up as a gif picture to simulate the moving path of each planet.

5.2. **Divergence.** We have so far observed the behavior of one initial set of points in three-dimensional space for each planet. However, it is especially interesting to consider the behavior of many initial points. One can consider the behavior of points in R^3, but this is often difficult to visualize as the trajectories may form something like a solid object. To avoid this difficulty, we can restrict our initial points to some subspace.[8]

In R^3 we can choose vectors in two basis vectors, x and y for example, to vary while z stays constant. Allowing two basis vectors to change freely in R^3 forms a two-dimensional sheet in that three-dimensional space, or a plane. What happens to points embedded in this plane as they observe Newton's laws of gravitation, and in particular to different points in this plane that have different sensitivities to initial conditions.

As we have three bodies, we can choose one of them to allow us to start in different locations in a two-dimensional plane while holding the other two bodies fixed in their starting points before observing the trajectory resulting from each of these different starting locations. As we are interested in divergence, we can once again compare the trajectory of the planet at each of the starting points to a slightly shifted planet. As we are now dealing with a multidimensional array of initial points rather than only three per planet, a slightly different approach is required. [2]

The computational structure to calculate sensitivity is initialized by changing the $3 \times 1 \times 1$ initial points for planet 1 to an array of size 3 by y-residue and x-residue for the planet. And we still need to have arrays of identical dimensions for all the other planets, as their trajectories each depend on the initial points of planet 1. This means that we must make arrays that track the motion of each planet in three-dimensional space for all points in the initial plane of possible plant 1 values.[3]

Then, we use "Numpy", an indispensable library for mathematical computations, and "PyTorch" to speed up the three-body divergence calculations.

After 40,000 time steps, initial points on the x, y plane of planet 1 on both x and y axes, we have the following figure2

We notice that something must exist in the initial positions of the second and third planets such that symmetry results for the y, z plane. In general terms, the two starting points that result in identical trajectories in space will experience the same divergence rate. Now consider the planets 2 and 3 in three-dimensional space. The points that are equidistant from any point on the line connecting planets 2 and 3, projected onto the y, z plane are of equal distance to planets 2 and 3. Since each pair of points in the y, z plane that is equidistant from any point on the line will thus have identical trajectories, they will have equal divergence.[10]

When we plot the line together with the map formed through divergence in the y,z plane for x = -10, we will see the line of our mirror symmetry. where lighter values indicate earlier divergence.

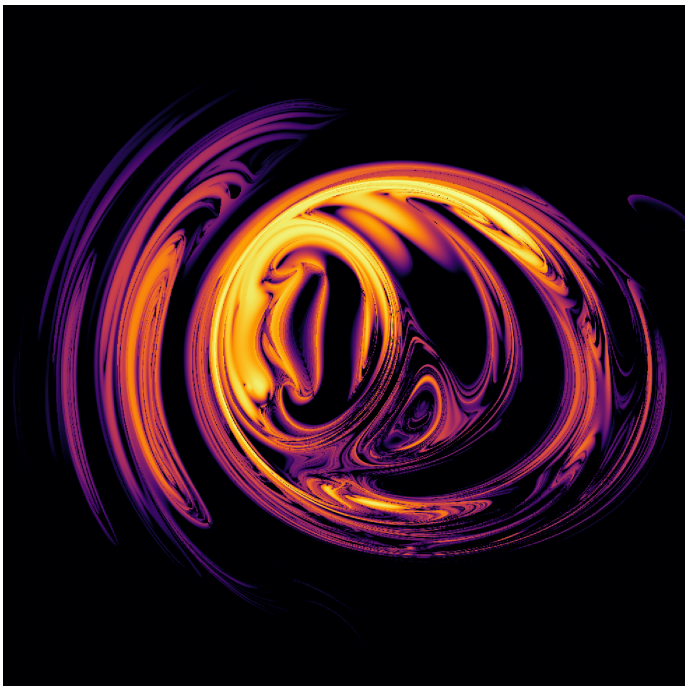

FIGURE 2. The divergence figure for three body system in the plane consisted by xy-axis, where lighter values indicating earlier divergence

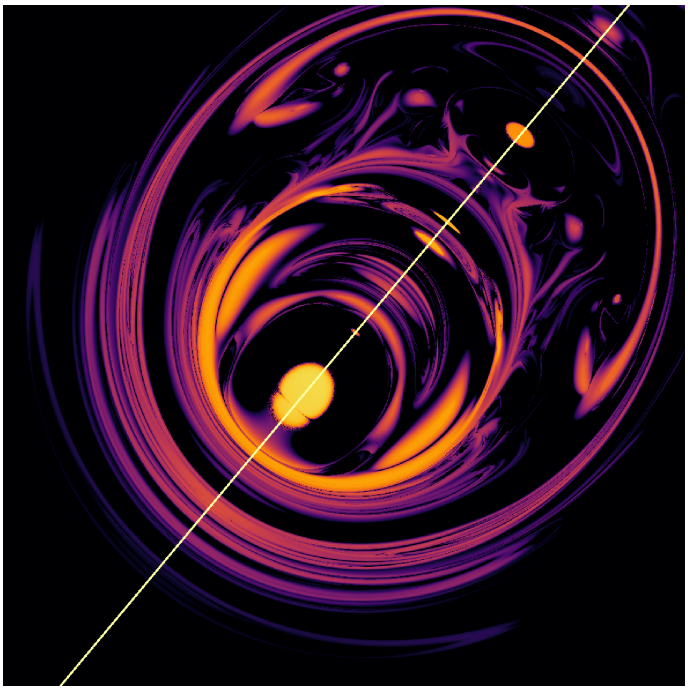

FIGURE 3. Mirror Symmetry: we can see a clear symmetry relationship within this figure. The divergence figure for three body system in the plane consisted by yz-axis, where lighter values indicating earlier divergence

5.3. **Three-body system Plot.** In the process of investigating the three-body system, researchers invested great energy on find a periodic route (actually people think investigating the period route is the only method to understand the complex three-body system). And in the past decades, researchers utilized the numerical integration method to find some tidy plate periodic routes.[4] For example:

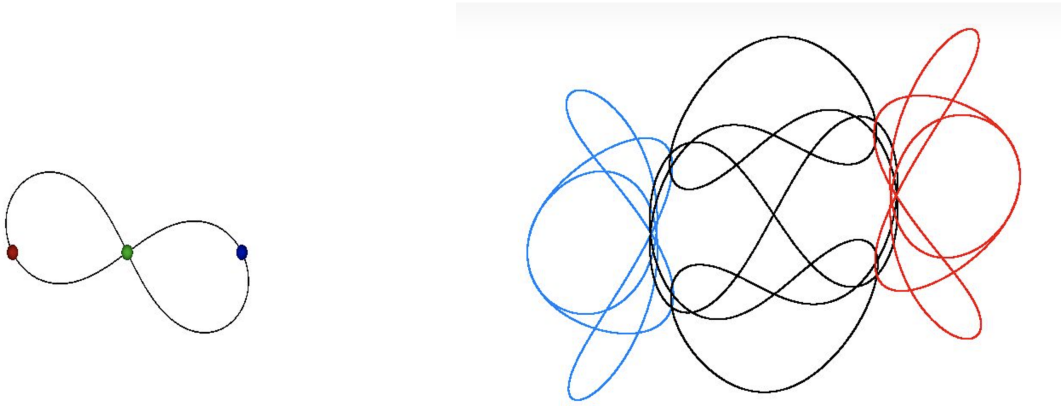

FIGURE 4. Periodic route Example 1

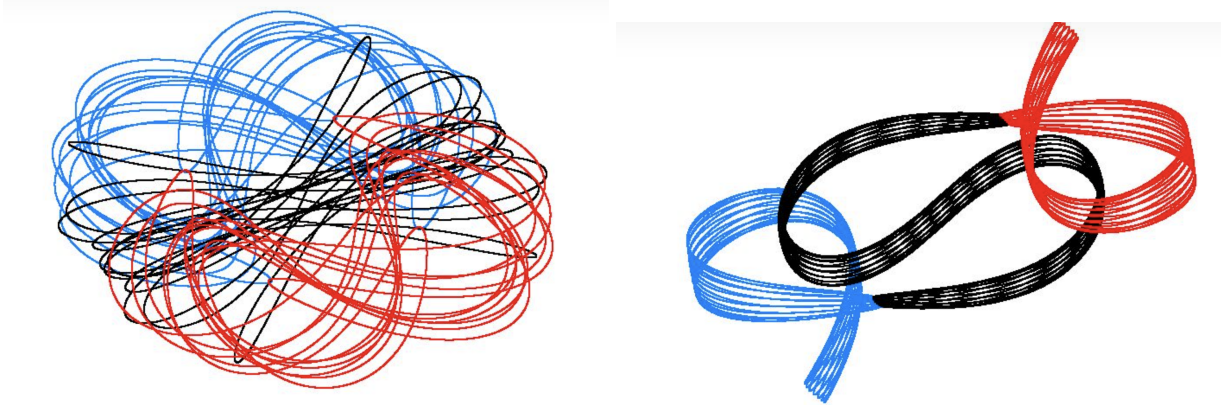

FIGURE 5. Periodic route Example 2

In our project, we use PyTorch and matplotlib to make several three-body system paths. When we set the three-body with the same masses, we get their following path in 1000 seconds.

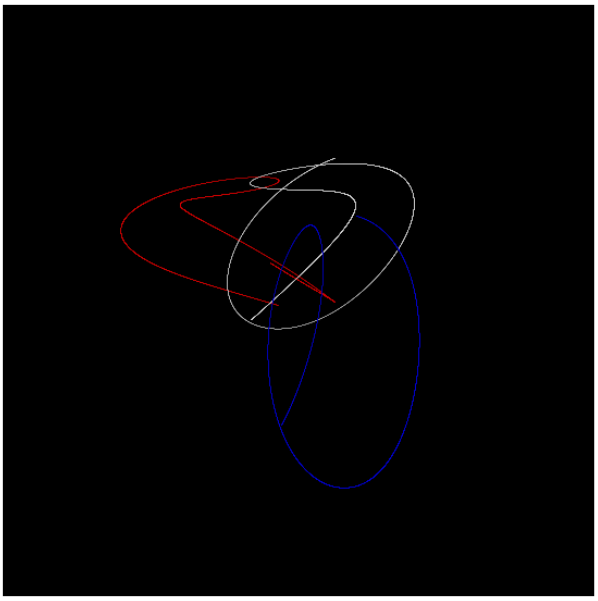

FIGURE 6. The path of three planets with the same masses

However, when we change the masses of the three-body into a system of one with extremely large mass and the other two have the same low masses, we get the following path in 1000 seconds.

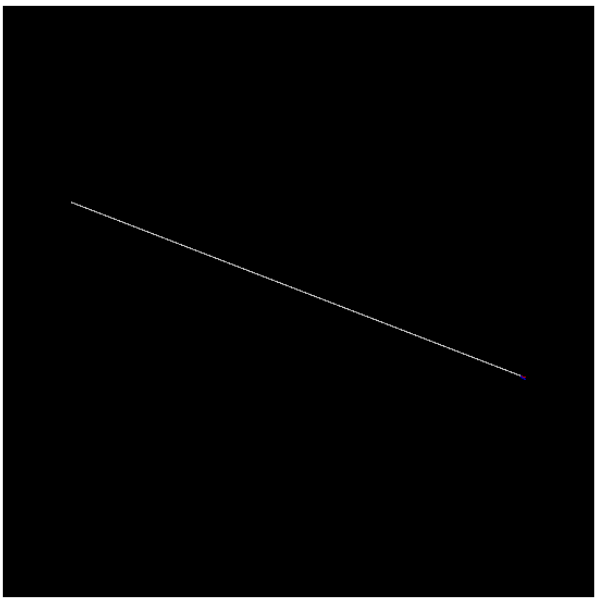

FIGURE 7. The path of three planets with one having extremely large mass

Also, we simulated the three-body system over the earth, Jupiter and Sun, and the following is the result. In the graph, x-axis and y-axis denotes the distance in these two directions with one unit being 1×10^8 km.

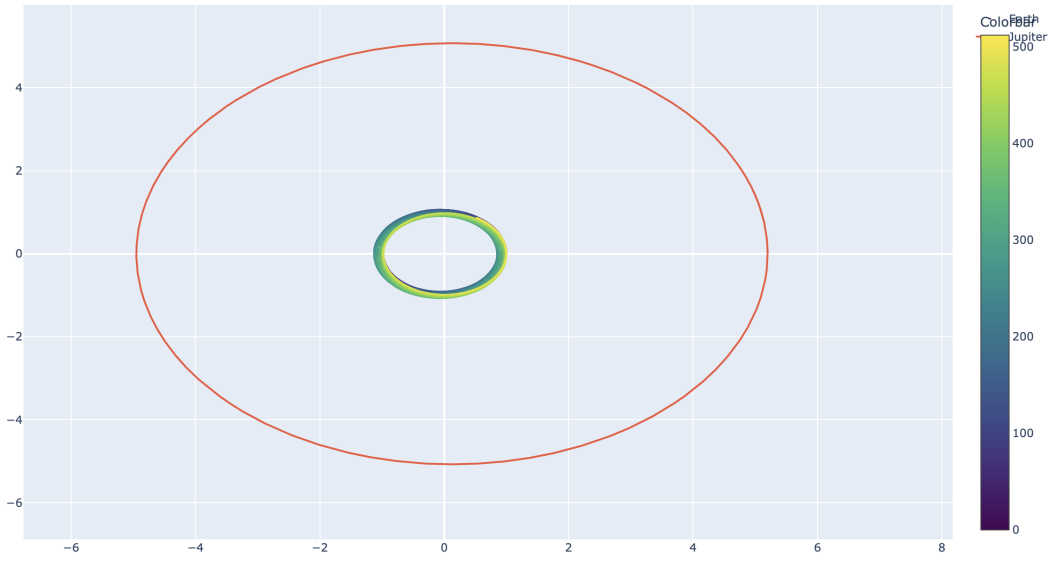

FIGURE 8. Earth vs. Jupiter

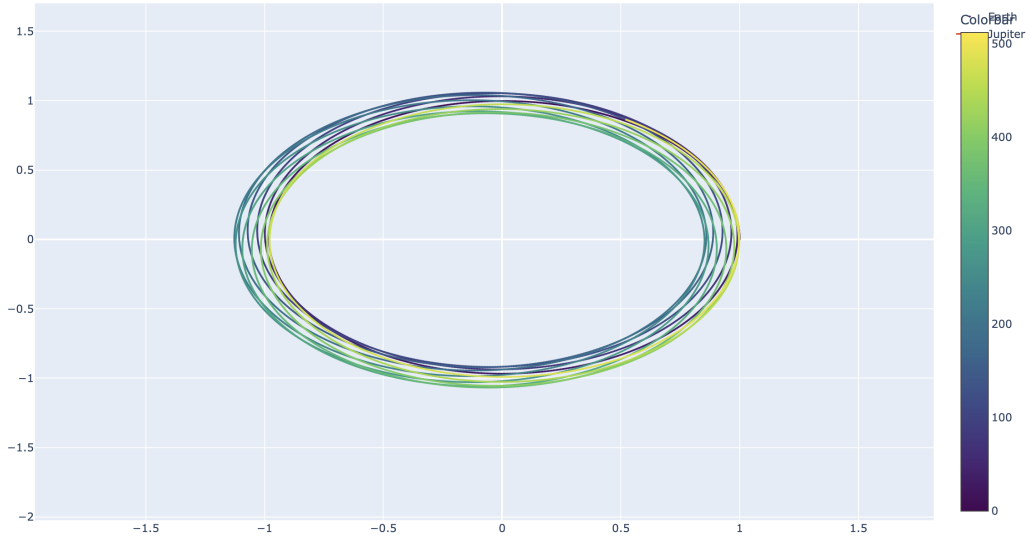

FIGURE 9. Larger Vision Earth vs. Jupiter

5.4. **Outlooks.** One can hope that the three-body problem is restricted to celestial mechanics and that it does not find its way into other fields of study. Great effort has been expended to learn about the orbitals an electron will make around the nucleus of a proton, so hopefully, this knowledge is transferable to an atom with more than one proton.

Solutions to the three-body problem are either restricted to a subset of initial conditions or are references to the process of numerical integration by a computer. The latter idea gives rise to the sometimes-held opinion that the three-body problem is solvable now that high-speed computers are present because one can simply use extremely precise numeric methods to provide a solution.[9]

To understand why computers can not solve our problems, we should first pretend that perfect observation were made. This means a perfect measurement of G will not help since it would take infinite time to enter into a computer exactly. That being said, computational methods are very good for determining short-term trajectories. Moreover, when certain bodies are much larger in mass than others, the ability to determine trajectories is substantially enhanced. However, for an aperiodic equation system, the ability to determine trajectories for all time is impossible.

6. Conclusion

We have reviewed the three-body problem in which three spherical masses interact with each other only through the gravitational interactions described by Newton's theory of gravity, and no restrictions are imposed on the initial position and velocities. We overview several solutions methods to the problem, giving special emphasis to the difficulty of finding closed-form solutions to the three-body problem due to its unpredictable behavior. We gave detailed descriptions of general and restricted three-body problems and described several analytical and numerical methods utilized to find solutions, perform stability analysis and make figures for periodic orbits and resonances. The three-body problem described in this paper may be considered classical since the bodies are assumed to be spherical objects and points of given masses, as described by Newton's theory of gravity.

7. Reference

References

[1] Francesco Calogero. Solution of a three-body problem in one dimension. *Journal of Mathematical Physics*, 10(12):2191–2196, 1969.

[2] Vitaly Efimov. Energy levels arising from resonant two-body forces in a three-body system. *Physics Letters B*, 33(8):563–564, 1970.

[3] Joshua Fitzgerald and Shane Ross. Geometry of transit orbits in the periodically-perturbed restricted three-body problem. *arXiv preprint arXiv:2203.16019*, 2022.

[4] Bastian Kaspschak and Ulf-G Meissner. Three-body renormalization group limit cycles based on unsupervised feature learning. *Machine Learning: Science and Technology*, 2022.

[5] Christian Marchal. The three-body problem. 2012.

[6] Valtonen MJ, Mauri Valtonen, and Hannu Karttunen. *The three-body problem*. Cambridge University Press, 2006.

[7] Zdzislaw E Musielak and Billy Quarles. The three-body problem. *Reports on Progress in Physics*, 77(6):065901, 2014.

[8] Esben Nielsen, Dmitri Vladimir Fedorov, Aksel S Jensen, and Eduardo Garrido. The three-body problem with short-range interactions. *Physics Reports*, 347(5):373–459, 2001.

[9] Yilun Shang. A system model of three-body interactions in complex networks: Consensus and conservation. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2022.

[10] Yi Zhou and Wei Zhang. Analysis on nonlinear dynamics of two first-order resonances in a three-body system. *The European Physical Journal Special Topics*, pages 1–18, 2022.

8. Appendix

8.1. Core Code.

```
# import the third-part library
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# set the 3D graph background
# plt.style.use('dark_background')
# set the masses of planets
m_1, m_2, m_3 = 10000, 100, 100

# p : position  v: velocity
# set the coordintes for planets
# p1_start = x_1, y_1, z_1
p1_start = np.array([-10, 10, -11])
v1_start = np.array([-3, 0, 0])

# p2_start = x_2, y_2, z_2
p2_start = np.array([0, 0, 0])
v2_start = np.array([0, 0, 0])

# p3_start = x_3, y_3, z_3
p3_start = np.array([10, 10, 12.00000])
v3_start = np.array([3, 0, 0])

# starting coordinates for planets shifted
# p1_start = x_1, y_1, z_1
p1_start_prime = np.array([-10, 10, -11])
v1_start_prime = np.array([-3, 0, 0])

# p2_start = x_2, y_2, z_2
p2_start_prime = np.array([0, 0, 0])
v2_start_prime = np.array([0, 0, 0])

# p3_start = x_3, y_3, z_3
p3_start_prime = np.array([10, 10, 12.000001])
v3_start_prime = np.array([3, 0, 0])
def accelerations(p1, p2, p3):
    '''A function to calculate the derivatives of x, y, and z
    given 3 object and their locations according to Newton's laws
    '''
    planet_1_dv = -9.8 * m_2 * (p1 - p2)/(np.sqrt(np.sum([i**2 for i in
    p1 - p2]))**3) - 9.8 * m_3 * (p1 - p3)/(np.sqrt(np.sum([i**2 for i
    in p1 - p3]))**3)

    planet_2_dv = -9.8 * m_3 * (p2 - p3)/(np.sqrt(np.sum([i**2 for i in
```

```
    p2 - p3]))**3) - 9.8 * m_1 * (p2 - p1)/(np.sqrt(np.sum([i**2 for i
    in p2 - p1]))**3)

    planet_3_dv = -9.8 * m_1 * (p3 - p1)/(np.sqrt(np.sum([i**2 for i in
    p3 - p1]))**3) - 9.8 * m_2 * (p3 - p2)/(np.sqrt(np.sum([i**2 for i
    in p3 - p2]))**3)

    return planet_1_dv, planet_2_dv, planet_3_dv
# time parameters
delta_t = 0.001
steps = 100000
# initialize solution array
p1 = np.array([[0.,0.,0.] for i in range(steps)])
v1 = np.array([[0.,0.,0.] for i in range(steps)])

p2 = np.array([[0.,0.,0.] for j in range(steps)])
v2 = np.array([[0.,0.,0.] for j in range(steps)])

p3 = np.array([[0.,0.,0.] for k in range(steps)])
v3 = np.array([[0.,0.,0.] for k in range(steps)])

# second trajectory start, for comparison to (p1, p2, p3)
p1_prime = np.array([[0.,0.,0.] for i in range(steps)])
v1_prime = np.array([[0.,0.,0.] for i in range(steps)])

p2_prime = np.array([[0.,0.,0.] for j in range(steps)])
v2_prime = np.array([[0.,0.,0.] for j in range(steps)])

p3_prime = np.array([[0.,0.,0.] for k in range(steps)])
v3_prime = np.array([[0.,0.,0.] for k in range(steps)])
# starting points
p1[0], p2[0], p3[0] = p1_start, p2_start, p3_start

v1[0], v2[0], v3[0] = v1_start, v2_start, v3_start

p1_prime[0], p2_prime[0], p3_prime[0] = p1_start_prime, p2_start_prime,
p3_start_prime

v1_prime[0], v2_prime[0], v3_prime[0] = v1_start_prime, v2_start_prime,
v3_start_prime

time = [0]

# evolution of the system
for i in range(steps-1):
    time.append(i)
```

```
    # calculate derivatives
    dv1, dv2, dv3 = accelerations(p1[i], p2[i], p3[i])
    dv1_prime, dv2_prime, dv3_prime = accelerations(p1_prime[i],
    p2_prime[i], p3_prime[i])

    v1[i + 1] = v1[i] + dv1 * delta_t
    v2[i + 1] = v2[i] + dv2 * delta_t
    v3[i + 1] = v3[i] + dv3 * delta_t

    p1[i + 1] = p1[i] + v1[i] * delta_t
    p2[i + 1] = p2[i] + v2[i] * delta_t
    p3[i + 1] = p3[i] + v3[i] * delta_t

    # alternate trajectory (primes are not derivatives)
    v1_prime[i + 1] = v1_prime[i] + dv1_prime * delta_t
    v2_prime[i + 1] = v2_prime[i] + dv2_prime * delta_t
    v3_prime[i + 1] = v3_prime[i] + dv3_prime * delta_t

    p1_prime[i + 1] = p1_prime[i] + v1_prime[i] * delta_t
    p2_prime[i + 1] = p2_prime[i] + v2_prime[i] * delta_t
    p3_prime[i + 1] = p3_prime[i] + v3_prime[i] * delta_t

# For the purposes of plotting trajectories overt time

if i % 1000 == 0:
    fig = plt.figure(figsize=(10, 10))
    ax = fig.gca(projection='3d')
    plt.gca().patch.set_facecolor('black')
    ax.set_xlim([-50, 300])
    ax.set_ylim([-10, 30])
    ax.set_zlim([-30, 70])

    plt.plot([i[0] for i in p1], [j[1] for j in p1], [k[2] for k in
    p1] , '^', color='red', lw = 0.05, markersize = 0.01, alpha=0.5)
    plt.plot([i[0] for i in p2], [j[1] for j in p2], [k[2] for k in
    p2] , '^', color='white', lw = 0.05, markersize = 0.01, alpha=0.5)
    plt.plot([i[0] for i in p3], [j[1] for j in p3], [k[2] for k in
    p3] , '^', color='blue', lw = 0.05, markersize = 0.01, alpha=0.5)
    # plt.plot([i[0] for i in p1_prime], [j[1] for j in p1_prime],
    [k[2] for k in p1_prime], '^', color='blue', lw=0.05,
    markersize=0.01, alpha=0.5)

    plt.axis('on')

    # optional: use if reference axes skeleton is desired,
```

```
    # ie plt.axis is set to 'on'
    ax.set_xticks([]), ax.set_yticks([]), ax.set_zticks([])

    # make pane's have the same colors as background
    ax.w_xaxis.set_pane_color((0.0, 0.0, 0.0, 1.0)),
    ax.w_yaxis.set_pane_color((0.0, 0.0, 0.0, 1.0)),
    ax.w_zaxis.set_pane_color((0.0, 0.0, 0.0, 1.0))
    ax.view_init(elev = 20, azim = i//1000)
    # set up saving path
    plt.savefig('graphB{}'.format(i//1000), bbox_inches='tight',
    dpi=300)
    plt.close()
fig = plt.figure(figsize=(10, 10))
ax = fig.gca(projection='3d')
plt.gca().patch.set_color("black")
plt.gca().patch.set_edgecolor("black")
plt.gca().patch.set_facecolor('black')

plt.plot([i[0] for i in p1], [j[1] for j in p1], [k[2] for k in p1] ,
'^', color='red', lw = 0.05, markersize = 0.01, alpha=0.5)
plt.plot([i[0] for i in p2], [j[1] for j in p2], [k[2] for k in p2] ,
'^', color='white', lw = 0.05, markersize = 0.01, alpha=0.5)
plt.plot([i[0] for i in p3], [j[1] for j in p3], [k[2] for k in p3] ,
'^', color='blue', lw = 0.05, markersize = 0.01, alpha=0.5)
# plt.plot([i[0] for i in p2_prime], [j[1] for j in p2_prime], [k[2] for
# k in p2_prime], '^', color='blue', lw=0.05, markersize=0.01, alpha=0.5)

plt.axis('on')

# optional: use if reference axes skeleton is desired,
# ie plt.axis is set to 'on'
ax.set_xticks([]), ax.set_yticks([]), ax.set_zticks([])

# make pane's have the same colors as background
ax.w_xaxis.set_pane_color((0.0, 0.0, 0.0, 1.0)),
ax.w_yaxis.set_pane_color((0.0, 0.0, 0.0, 1.0)),
ax.w_zaxis.set_pane_color((0.0, 0.0, 0.0, 1.0))

ax.view_init(elev = 20, azim = t)
plt.savefig('{}'.format(t), dpi=300, bbox_inches='tight')
plt.show()
plt.close()
```