Stability of Three-Body Solutions with a Fourth Body

Ibrahim Emad Ibrahim El-Sayed El-Serwy

Received August 06, 2024
Accepted September 13, 2024
Electronic access September 30, 2024

Thousands of solutions have been found to the three-body problem; however, the stability of the discovered solutions is rarely discussed. This paper tests the stability of eight solutions to the three-body problem found by Šuvakov and Dmitrašinović in 2013. The solutions are tested for stability in the case of equal masses ($m_i = 1$) and the addition a fourth body to the system of a limited mass (between 0.001 and 0.01). It was found that many solutions exhibit some kind of gravitational stability after the addition of a fourth body. The results showed that each solution had a unique range of masses at which it retained its gravitational binding. Other systems showed complete chaos after introducing even a small fourth body. Factors of stability such as the position of the fourth body in addition to its initial velocity were tested, adding to the precision of the tests. All four-body systems studied exhibited significantly chaotic configurations despite retaining relative gravitational binding in some cases. The results can help us further understand these complex systems and expand on to the n-body problem and the production of weak gravitational waves due to the chaos exhibited in such systems.

Introduction

The three-body problem, referring to the gravitational interaction of three bodies in space, has been puzzling since the days of Newton. Written at the end of the volume of Opticks, Newton expressed doubts about the stability of a system of mutual gravitational interaction among more than two bodies, stating that their Keplerian motion will be disturbed[1]. Different attempts have been made to solve the problem in the same way as that of two bodies, but no solutions were found, and the realization of this failure led to Poincare's statement that the system of equations of motion of the three-body problem is non-integrable[2]. Despite the chaos that exists in the system, Euler and Lagrange were able to find instances at which the system exhibited stability over periods of time. The solutions found were called periodicals, and scientists have since been finding additional periodic solutions to the system.

The solutions to the three-body problem are categorized in families according to the topographical methods of Montgomery. The first discovered family was Lagrange-Euler family, dating back to the 18th century[3]. The second discovered family was the The Broucke-Hadjidemetriou-Henon family (BHH family), discovered by Broucke (1975), Hadjidemetriou (1975), and Henon (1976)[4]. The third family, the figure-8 family, was discovered by Moore in 1993 and rediscovered by Montgomery in the 2000s[5]. A breakthrough occurred when Šuvakov and Dmitrašinović discovered 13 new families in 2013; hundreds of solutions have been discovered afterwards[6].

The three-body problem has applications in different fields like astronomy, physics, and chemistry. It was used to study the effect of the sun on the moon and the trojan system. Using the idea of the three-body problem, the literature was able to expand it to the n-body problem, a system collecting n–bodies gravitationally, allowing the study of the solar system and clusters of stars containing millions of stars. In 2012, Laskar used the idea of the n-body problem to study the stability of the solar system[1]. In chemistry, collision theory can be used to describe the reaction of three elements[7]. In this paper, the stability of zero angular momentum systems will be scrutinized for stability after the introduction of a fourth body considering three factors: the mass of the fourth body, its initial position, and its initial velocity.

Methodology

Notation and Conditions

I will be using Newton's notation for differentiation (i.e $\dot{y} = \frac{dy}{dx}$). In addition, a geometrized unit system is used for the sake of simplicity in computations ($G = 1; m_1 = 1$, where G is the universal gravitational constant and m_l is the mass of the body i in the plane). The masses, distances, and time in the system are all relative, unitless quantities.

The equations used in this paper ate alldescribed for two-dimensional space $\mathbb{R}^2$ (the x-y plane). Since trio-systems always exist in a two-dimensional plane, then $\mathbb{R}^2$ can be used to describe the system; this was proved in Montgomery's paper about the topology of the three-body systems[8]. The four-bodjes case will, however, require working in the three-dimensional space $\mathbb{R}^3$ (x - y - z). Dealing with the system in a 3 -dimensional space is power-coysyming so to decrease dependency on computational power, the systems will all be tested in the two-dimensional

plane; however, the methods developed in the paper can be used to test the stability in a three-dimensional notion.

Equations of Motion

Objects are atracted to each other gravitationally due to their masses according to the equations of motion proposed by Newton, so the gravitational acceleration of each body follows Equation (1):

$$\ddot{r} = \sum_{i}^{\pi} G \frac{\hat{m}_i}{|r_i|^2} \hat{r}_2 \qquad (1)$$

where $\hat{r}_1$ is the unit direction of the position vector between the specified body and every other body i surrounding it, $|r_i|^2$ is the magnitude of the positional vector between the sought object and each surrounding object i. $\ddot{r}$ is the second derivative of the position vector, G is the gravitational constant, and m_i is the mass of each body. The differential equations of motion that describe, the interactions among bodies can be determined by positional vectors and passes as follows in. Equations (2) and(3):

$$\ddot{x}_1(t) = -\frac{Gm_2(x_1 - x_2)}{\left[(x_1 - x_2)^2 + (y_1 - y_2)^2\right]^{3/2}} - \frac{Gm_3(x_1 - x_3)}{\left[(x_1 - x_3)^2 + (y_1 - y_3)^2\right]^{3/2}}, \qquad (2)$$

$$\ddot{y}_1(t) = -\frac{Gm_2(y_1 - y_2)}{\left[(x_1 - x_2)^2 + (y_1 - y_2)^2\right]^{3/2}} - \frac{Gm_3(y_1 - y_3)}{\left[(x_1 - x_3)^2 + (y_1 - y_3)^2\right]^{3/2}}. \qquad (3)$$

The above equation describes the motion of body 1 . Similarly, we can describe the game two equations for both bodies 2 and 3 in the system ($1 \to 2 \to 3$ and $1 \to 3 \to 2$), hence we have gix equations of motion to describe the system fully. When the system contains a fourth body (as in the case tested in the paper), we must include a fourth factor to both x and y equations of motions of the body. The terms are expressed in Equations (4) and (5).

$$-\frac{Gm_4(x_1 - x_4)}{\left((x_1 - x_4)^2 + (y_1 - y_4)^2\right)^{3/2}} \qquad (4)$$

$$-\frac{Gm_4(y_1 - y_4)}{\left((x_1 - x_4)^2 + (y_1 - y_4)^2\right)^{3/2}} \qquad (5)$$

To study the evolution of the system, we have to solve for all the objects' positions ($\vec{r}_1, \vec{r}_2, \vec{r}_3$) and their derivative with respect to time ($\vec{v}_1, \vec{v}_2, \vec{v}_3$).As a result, we are working with a 12 -vector denoted as follows in Equation (6):

$$\vec{X}'(t) = \left(\vec{r}_1(t), \vec{r}_2(t), \vec{r}_3(t), \vec{v}_1(t), \vec{v}_2(t), \vec{v}_3(t)\right) \qquad (6)$$

Note that each component has two dimensions x and y, which makes the 12 -vector The aforementioned equations can guarantee conservation in angular momentum, linear momentum, and mechanical energy of the system. The angular momentum can be calculated using Equation (7):

$$\vec{L} = \vec{r} \times \vec{p} \qquad (7)$$

where $\vec{r}$ and $\vec{p}$ are the position and momentum vectors respectively. In addition, p canbedefined as the mass multiplied the velocity of the body ($\vec{p} = m \times \vec{v}$). Since we are working in a twodimensional plane, the magnitude of the angular momentum will be as shown in Equation (8).

$$L_i = x p_{ix} - y_i p_{iy} \qquad (8)$$

Where p_{ix} and p_{iy} are the x and y co-ordinates of the momentum of a body i. In addition, as the system in hand is a geometries unit system $(m_i = 1)$, the first derivative of the position vector can be used instead of the linear momentum in the above equation to get the angular momentum.

The system also guarantees a conservation in the mechanical energy (the sum of the kinetic energy and the potential energy of the system). Equations (9) and (10) areused to calculate these two quantities.

$$K_E = \frac{1}{2} \sum_{i=1}^{3} m_1 r_l^2 \qquad (9)$$

$$P_E = -\sum_{1/|} \frac{G m_t m_j}{|r_t - \eta_j|} \qquad (10)$$

The mechanical energy of the system will be the summation of the magnitude of both kinetic energy and potential energy as shown in Equation (11).

$$E_{\text{total}} = K_E + P_E \qquad (11)$$

Retaun Proximity Function

All the solutions to the three-body problem that waill be discossed in this paper can be checked for periodicity using the retuon proximity flontion. The fanction $d(X_0, T_0)$ is defined as

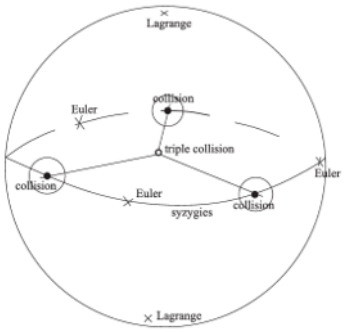


Figure 1: The Shape Sphere described by Jacobi Coordinates. [10]

the absolute minimum of the distance from the initial condition by $d(X_0, T_0) = \min_{\mathrm{R}T_0} |X(t) - X_0|$ as shown in Equation (12).

$$|X(t) - X_0| = \sqrt{\sum_{t=1}^{3} (\vec{r}_1(t) - \vec{n}_i(0))^2 + \sum_{i=1}^{3} |(\vec{p}_i(t) - \vec{p}_i(0))|^2} \tag{12}$$

To find a solution with some period $T < T_0$, the zeros of the function must be found at chosen orbits T_0 [9]. The algorithm used to solve the retum proximity function basically depends on the choosing some period T_0 and an initial condition y_0 and solving the equations of the motion to find y_{i+1} form y_1. After that, the distance between y_i and the initial condition is measured using linear interpolation. Initially, the distance to the initial condition will increase fism $t = 0$. When the distance starts decreasing, one must check whether it is minimal or not. . If the distance is minimal and is smaller than some arbitrary tolerance (error), we consider this initial condition to be a candidate for a periodic solution [9].

Every configuration of the three bodies (the shape of the triangle formed by them, independent of size), can be represented by a point in a unit sphere as the one shown in fig. 1, known as shape sphere [9]

The north and the south poles of the shape sphere corresponds to equilateral triangles that differ only in orientation of the bodies. In fact, a similar property holds true for the whole northern and southem hemigpheres; two triangles of the same shape but with opposite orientations are represented by two points that are symmetrical relative to the equatorial plane. The equator corresponds to degenerate triangles, where bodies are in syzygies - instants when all three bodies Basically Jacobki coordinates are relative coordinate vectors introduced by Carl Jacobi (even though Lagrange have reached them Larlier), and they are represented by Equation (13).

$$\rho = \frac{1}{\sqrt{2}}(r_1 - r_2) \text{ and } \lambda = \frac{1}{\sqrt{5}}(r_1 + r_2 - r_3) \tag{13}$$

The perk of these coordinate vectors is that they are mass-weighted vectors, so they work efficiently for the case of unequal masses in the three-body problem. To use the shape sphere for representing the solutions to the three-body problem, Equations (2) and (3) have to be solved then using ρ and λ, so we can graphically represent the solutions.

Although the return proximity function and the shape sphere are not used directly in this paper, it's crucial to define them to understand the topologies of the solutions tested [10]

The ODE Sobver

As mentioned earlier in the introduction, the six differential equations associated with describing the equations of motion cannot be solved unless at specific initial conditions. For that, the code used in the simulation will use an ordinary differential equations solver, namely solveivp [11]. The solver will be solving the ODEs numerically and is a part of the renowned python library SciPy. The method used in this paper to solve the ODEs will be the Rung-Kutta-Fehlberg Method for its precision in solving the equations while maintaining a non-draining memory usage [12].

Selection of Families

Through decades of work, hundreds of solutions to the three-body problem have been discovered; however, many of them showed instability after a short period of time, hence they are improbable to exist in real life. Contrary to that, some families with zero angular momentum in the system showed considerable stability over long periods of time, which implies that these systems are more likely to exist in real life. Consequently, this paper will focus mainly on testing the stability of zero-angular momentum families of solutions and categorizing them, all taken from the Three-body Gallery created by Šuvakov and Dmitrašinović. The tested solutions are the following: Figure-8; Butterfly-I; Butterfly-II; Bumblebee; Dragonfly; Goggles; Moth-I; and Moth-II [13].

The Code

In this section, significant snippets of the code will be explained.

Solution

```
import numpy as np
from scipy.integrate import solve_ivp
N = 100000
T = 0.001
t = np.linspace(0, N * T, N)
solution = solve_ivp(ThreeBody, [0, 670], y0, t_eval=t, rtol=1e-
12, args=(m1, m2
, m3))
```

This snippet represents laying out the time of simulation and calling out the function. NumPy library was used to lay down the line space of the time frame used. Here, N represents the number of points and T represents the time span of each step. For instance, T = 0.001 means that each step is 1 millisecond. The function ThreeBody was solved using solveivp, which is a feature found in SciPy library. solveivp solves an initial value problem for a system of ordinary differential equations. This function numerically integrates a system of ordinary differential equations given an initial value:

The initial value problem is defined as:

$$\dot{y} = f(t,y), \quad y(t_0) = y_0 \qquad (14)$$

where t is a 1-dimensional independent variable (time), and $y(t)$ and $f(t,y)$ are n-dimensional vector-valued functions. These elements determine the differential equations. The goal is to find some $y(t)$, which approximately satisfies the given initial conditions (i.e., $y(t_0) = y_0$).

In this piece of code, a main function was defined: VectorStability. The function takes one argument (m, px, py, vx, vy) to determine what factor will be tested. After that the function iterates over a certain range to change either mass of the fourth body, initial positions, or the initial velocity.

RESULTS AND DISCUSSION

While testing the stability of the solutions, it was found that the system under the influence of a fourth body doesn't show linear stability; instead, islands of stability were found to be stable among instability regions, which caused some issues. To solve this problem, only a specific interval was taken into account. The range chosen for the mass was (0.001, 0.01) with an interval of 0.0001. This can be justified by the fact that the introduced body will always have a very small mass in comparison with the masses of the existing three-body system; this can be found in the case of planets orbiting trinary star systems. In addition, initial positions and velocities varied between 0.1 and 1 with an interval of 0.01.

After testing the systems for change in initial positions and velocities, it was found that the systems turn chaotic without forming any topologically clear configuration in addition to a loose gravitational binding. An example of one of the systems is shown in fig. 2, and the changes in x and y coordinates for the

Stability Simulation

```
def VectorStability(g):
    x = 50
    distance = []
    if g == 'm':
        for i in stable_masses:
            used_mass = i
            print(i)
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, i))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_m.append(used_mass)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_m.append(used_mass)
                    break
            return output
    elif g == 'px':
        for i in positions_x:
            used_pos_x = i
            print(i)
            y0[6] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
```

```
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_px.append(used_pos_x)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_px.append(used_pos_x)
                    break
            return output
    elif g == 'py':
        for i in positions_y:
            used_pos_y = i
            print(i)
            y0[7] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_py.append(used_pos_y)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_py.append(used_pos_y)
                    break
            return output
```

```
    elif g == 'vx':
        for i in velocities_x:
            used_vel_x = i
            print(i)
            y0[14] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_vx.append(used_vel_x)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_vx.append(used_vel_x)
                    break
            return output
    elif g == 'vy':
        for i in velocities_y:
            used_vel_y = i
            print(i)
            y0[15] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
```

```
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_vy.append(used_vel_y)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_vy.append(used_vel_y)
                    break
            return output
```

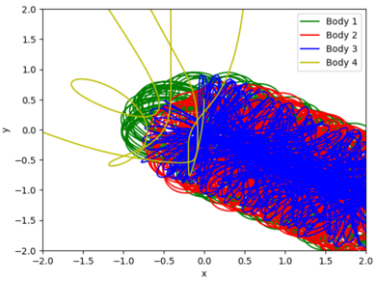


Figure 2: Dragonfly solution configuration after changing the x-position of body 4 to 0.1(made by the author).

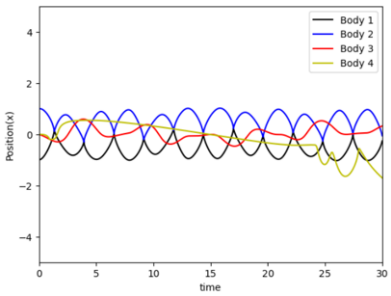


Figure 3: Dragonfly solution x-coordinate change with time after the addition of a fourth body with x-position 0.1 (made by the author).

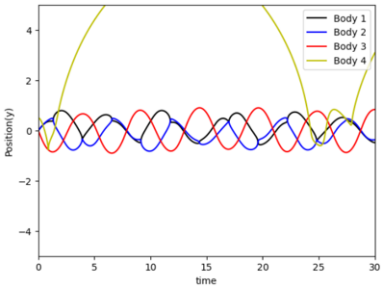


Figure 4: Dragonfly solution y-coordinate change with time after the addition of a fourth body with x-position 0.1 (made by the author).

system are shown in fig. 3and fig. 4. For that, only the stability due to change in masses will be discussed in detail.

Figure-8 Solution

The figure-8 solution, shown in fig. 5, was found to be sensitive to any changes in the system; however, the addition of a fourth mass did not disrupt the gravitational binding of the system. The figure-8 solution with the addition of a fourth mass (0.0079) is shown in fig. 6. To have a better understanding of the disturbed system, the x and y coordinates of the four bodies were graphed, and they're shown in fig. 7 and fig. 8 respectively. Interestingly, the solution

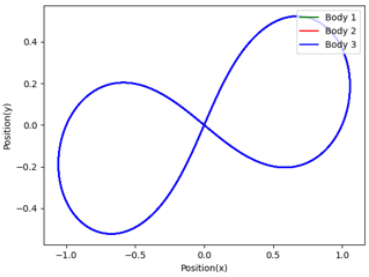


Figure 5: Figure-8 solution original configuration (made by the author).

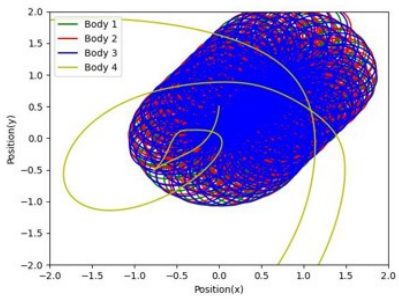


Figure 6: Figure-8 solution configuration after the addition of a fourth body with mass 0.0079 (made by the author).

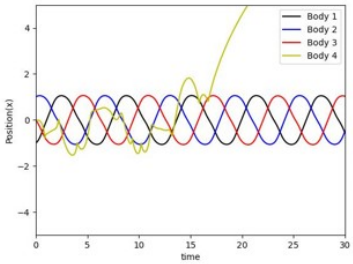


Figure 7: Figure-8 solution x-coordinate change with time after the addition of a fourth body with mass 0.0079 (made by the author).

exhibited stability points at which the overall topology of the system was not changed significantly, and these points are resembled in the following list: [0.00163, 0.00170, 0.00260, 0.00830]. The solution at 0.00163 is shown in fig. 9.

Butterfly-I

The butterfly-I solution, which is shown in fig. 10, was found to be sensitive to any changes in the system. The addition of any fourth body disturbed the system significantly. The disturbed system after the addition of a fourth body with mass 0.00163 is shown in fig. 11. The x and y coordinates of the unstable system

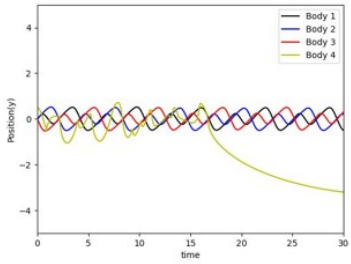


Figure 8: Figure-8 solution y-coordinate change with time after the addition of a fourth body with mass 0.0079 (made by the author).

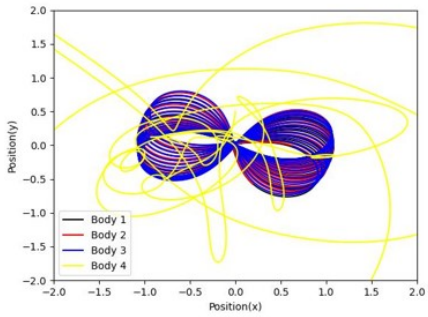


Figure 9: Figure-8 solution after the addition of a fourth body with mass 0.00163. The system is showing a stable configuration (made by the author).

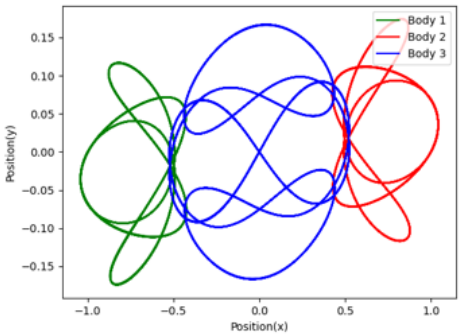


Figure 10: Butterfly-I original configuration (made by the author).

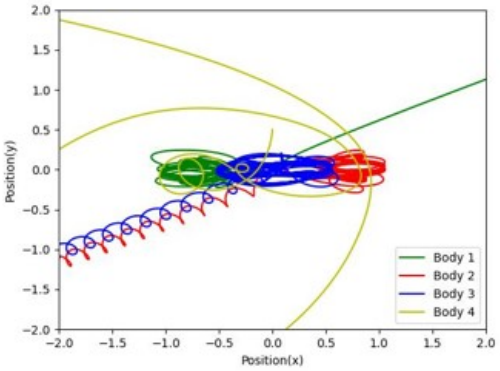


Figure 11: Butterfly-I configuration after the addition of a fourth body with mass 0.00163 (made by the author)

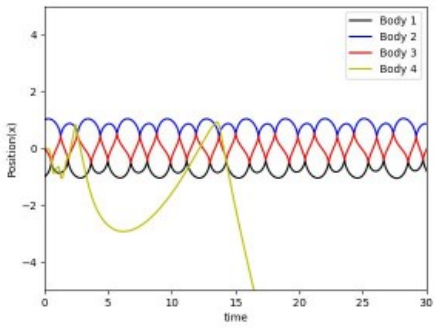


Figure 12: Butterfly-I solution x-coordinate change with time after the addition of a fourth body with mass 0.00163 (made by the author).

are shown in fig. 12 and fig. 13. We can see the drastic change in the y-coordinates that caused the chaos in the system.

Butterfly-II

The butterfly-II is shown in fig. 14. It was found to have a strong gravitational binding over the range [0.001:0.0019]. Over the specified range, the system failed to exist. The only effect that took place was a compression in the coordinates in the system. This can be traced back to the increase in the gravitational force in the system due to the introduction of a fourth body. The system with the addition of a fourth body with a mass of 0.0019 is shown in fig. 15. To have the full picture of the system, the x and y coordinates of the system were graphed and are shown in fig. 16 and fig. 17.

Bumblebee

This solution showed a very strange behavior under the influence of the fourth body. The system, in the stable cases, showed a considerable deformation from the original configuration shown

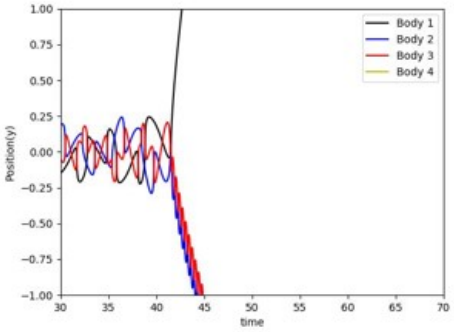


Figure 13: Butterfly-I solution x-coordinate change with time after the addition of a fourth body with mass 0.00163 (made by the author).

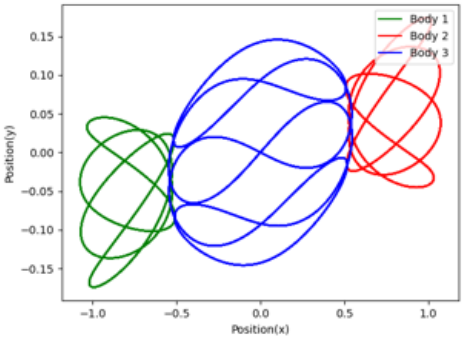


Figure 14: Butterfly-II original configuration (made by the author).

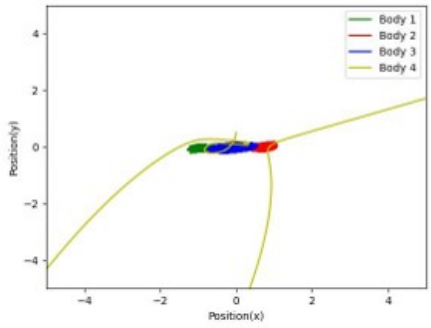


Figure 15: Butterfly-II configuration after the addition of a fourth body with mass 0.0019 (made by the author).

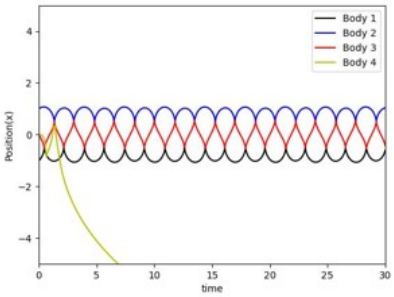


Figure 16: Butterfly-II solution x-coordinate change with time after the addition of a fourth body with mass 0.0019 (made by the author).

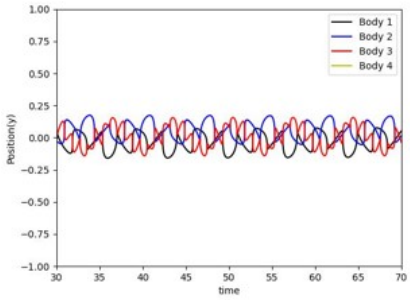


Figure 17: Butterfly-II solution y-coordinate change with time after the addition of a fourth body with mass 0.0019 (made by the author).

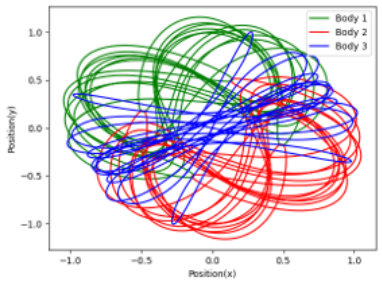


Figure 18: Bumblebee solution original configuration (made by the author).

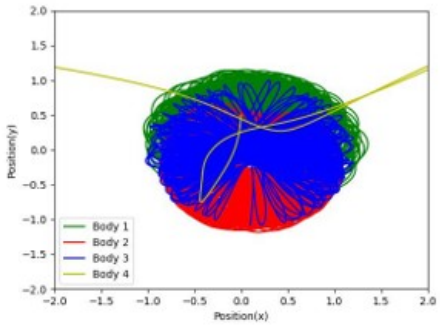


Figure 19: Bumblebee solution configuration after the addition of a fourth body with mass 0.001 (made by the author).

in fig. 18. However, among the gravitationally bound masses, certain points showed a gravitationally loose configuration with zero stability. The stability was in the range [0.001:0.0031]. fig. 19 is an example of gravitationally stable masses with its x and y coordinates shown in fig. 20 and fig. 21. The chaotic configuration is shown in fig. 22 and its x and y coordinates are shown in fig. 23 and fig. 24. The points at which the system exhibited chaos are [0.0021, 0.0023, 0.0025, 0.0028, 0.003]. This solution can be marked as stable over the defined range except for the unstable points.

Dragonfly

The dragonfly solution, in fig. 25, showed a strong gravitational binding over the range from 0.001 to 0.0021, after which the system failed to exist. The solution was, however, totally distorted by the introduction of a new mass. The original system was turned into various distorted configurations; one of them is fig. 26. The x and y coordinates of the distorted configuration are shown in fig. 27 and fig. 28. This solution can be labeled as stable over the mentioned range.

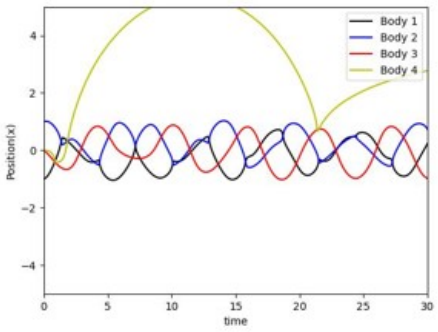


Figure 20: Bumblebee solution x-coordinate change with time after the addition of a fourth body with mass 0.00163 (made by the author).

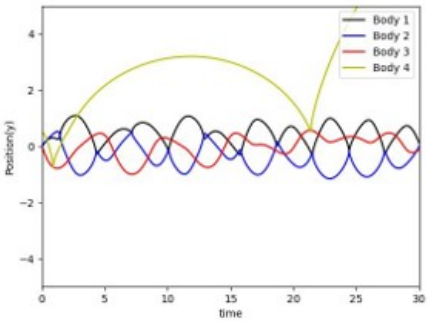


Figure 21: Bumblebee solution y-coordinate change with time after the addition of a fourth body with mass 0.00163 (made by the author).

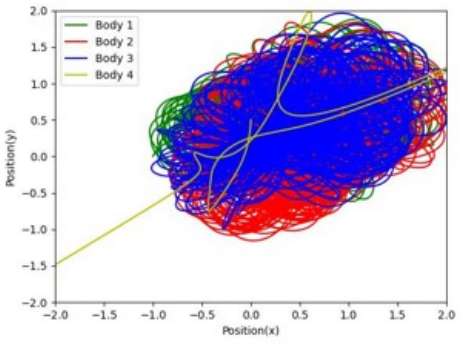


Figure 22: Bumblebee solution configuration after the addition of a fourth body with mass 0.0030 (made by the author).

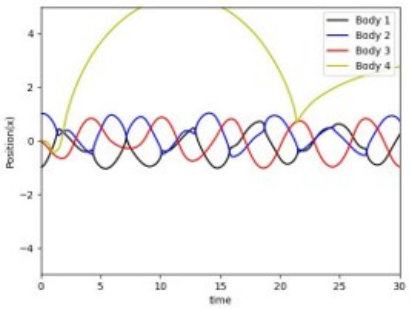


Figure 23: Bumblebee solution x-coordinate change with time after the addition of a fourth body with mass 0.0030 (made by the author).

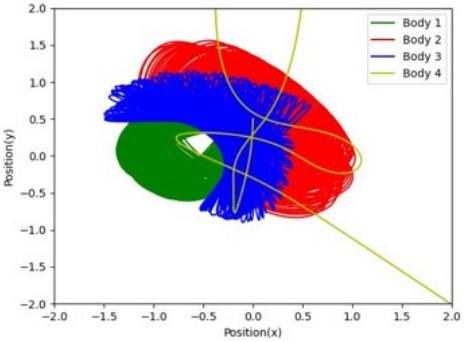


Figure 26: Dragonfly solution configuration after the addition of a fourth body with mass 0.0019 (made by the author).

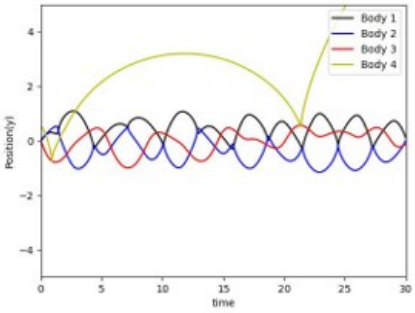


Figure 24: Bumblebee solution x-coordinate change with time after the addition of a fourth body with mass 0.0030 (made by the author).

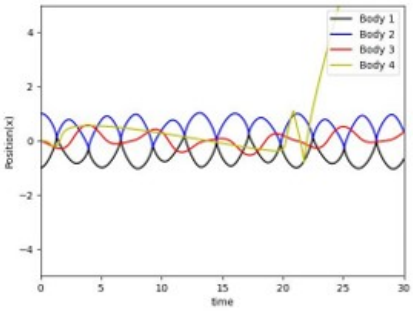


Figure 27: Dragonfly solution x-coordinate change with time after the addition of a fourth body with mass 0.0019 (made by the author).

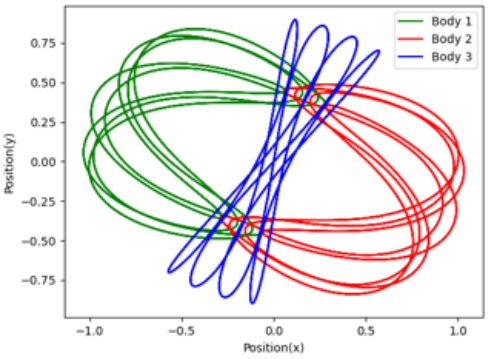


Figure 25: Dragonfly solution original configuration (made by the author).

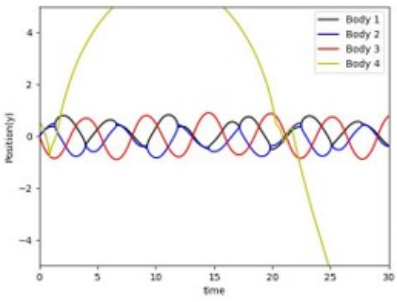


Figure 28: Dragonfly solution y-coordinate change with time after the addition of a fourth body with mass 0.0019 (made by the author).

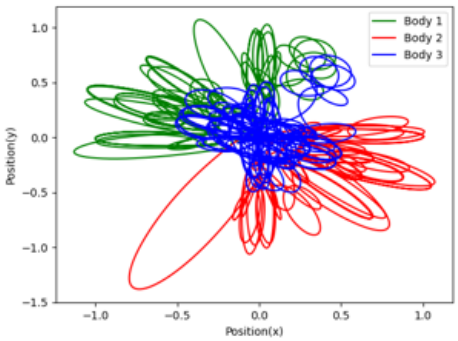


Figure 29: Goggles solution original configuration (made by the author).

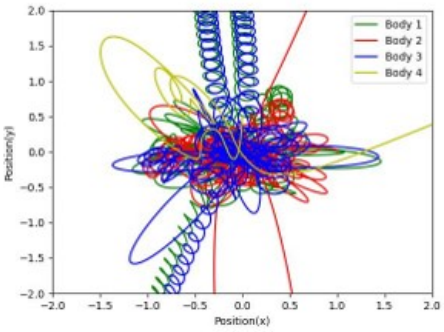


Figure 30: Goggles solution configuration after the addition of a fourth body with mass 0.0031 (made by the author).

Goggles

The Goggles solution was very sensitive to the introduction of a fourth body to the system, showing instant instability over the whole range of masses. The original system, shown in fig. 29, was changed to the system in fig. 30. To further investigate the behavior of the system, the x and y coordinates were graphed in fig. 31 and fig. 32. In the end, the system formed a two-body system and two single-body systems. This solution can be marked as unstable over the designated range.

Moth-I

The Moth-I solution showed a very strange behavior. Interestingly, the system was able to sustain its overall shape, with bodies moving faster, leading to the condensed graph; see fig. 33 and fig. 34 for comparison between the original configuration and the sustained configuration.

Among the stability range, there existed points at which the system was unbound and split in a two-body system and two single-body systems. Those points are [0.0035, 0.0036]. The

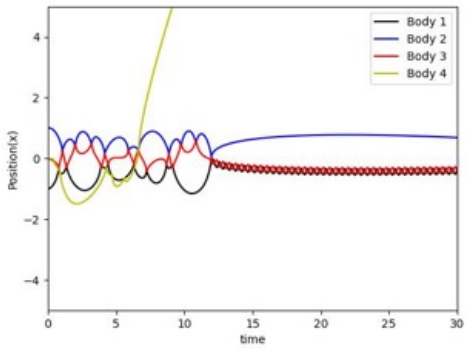


Figure 31: Goggles solution x-coordinate change with time after the addition of a fourth body with mass 0.0031 (made by the author).

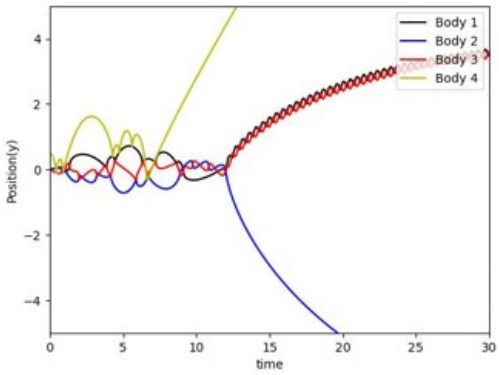


Figure 32: Bumblebee solution y-coordinate change with time after the addition of a fourth body with mass 0.0031 (made by the author).

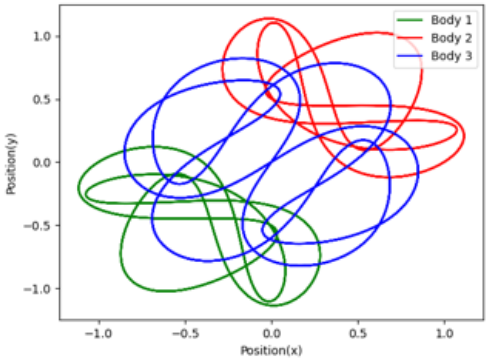


Figure 33: Moth-I original configuration (made by the author).

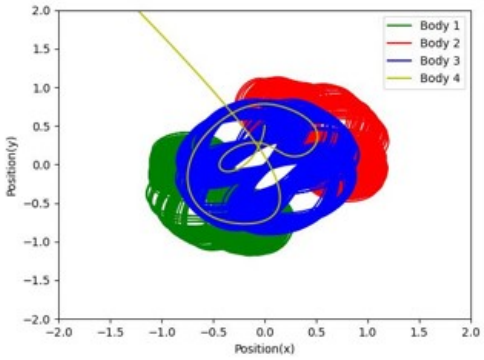


Figure 34: Moth-I solution configuration after the addition of a fourth body with mass 0.0024 (made by the author).

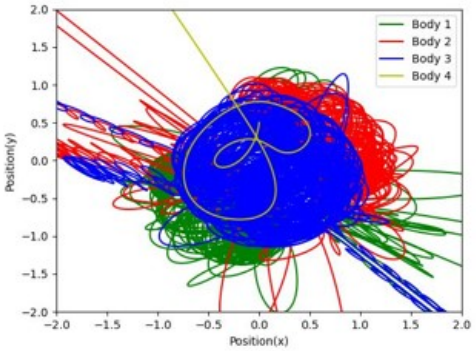


Figure 35: Moth-I solution configuration after the addition of a fourth body with mass 0.0036 (made by the author).

configuration after the addition of a fourth body with mass 0.0036 is shown in fig. 35. The x and y coordinates of the two four-body systems are shown in fig. 35 and fig. 36. Finally, the stability range for the solution was [0.001:0.0041], after which the system became unstable and kept splitting. This solution can be labeled as stable over the determined range with the exception of the two unstable points.

Moth-II

This solution was very sensitive to the introduction of any additional body to the system. The system failed to exist after the mass 0.0014. In this small range of four masses, the solution was able to keep itself gravitationally bound, with great deviation from the original configuration, for two masses (0.0011 and 0.0012) and split for the other two. The original configuration is shown in fig. 38, while the gravitationally bound and the distorted solution are shown in fig. 39 and fig. 40 respectively.

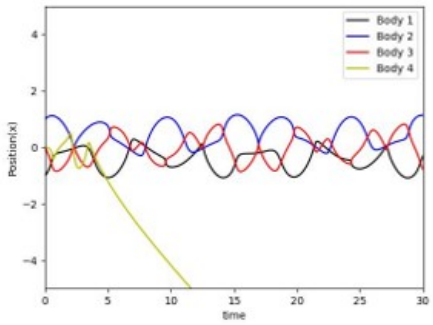


Figure 36: Moth-I solution x-coordinate change with time after the addition of a fourth body with mass 0.0036 (made by the author).

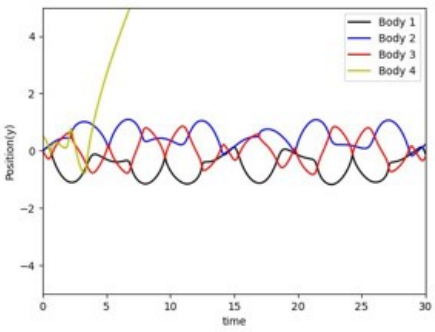


Figure 37: Moth-I solution y-coordinate change with time after the addition of a fourth body with mass 0.0036 (made by the author).

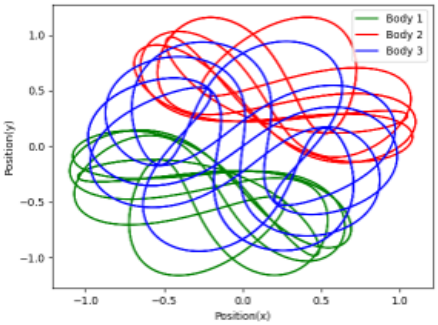


Figure 38: Moth-II solution original configuration (made by the author).

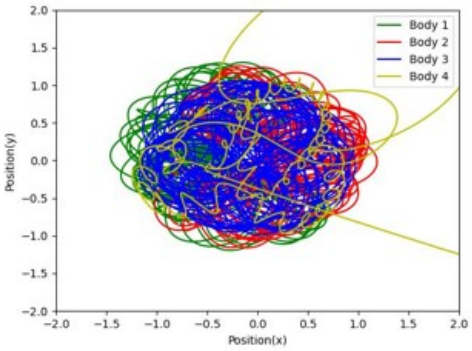


Figure 39: Moth-II solution configuration after the addition of a fourth body with mass 0.0010 (made by the author).

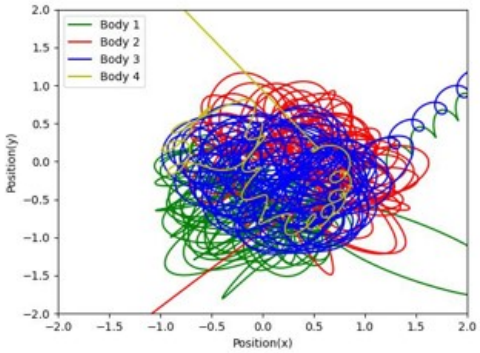


Figure 40: Moth-II solution configuration after the addition of a fourth body with mass 0.0012 (made by the author).

SUMMARY

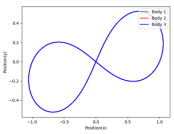
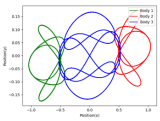
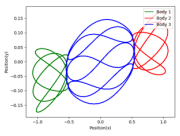
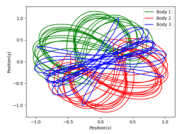
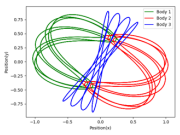
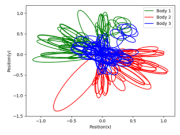
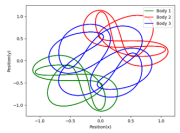
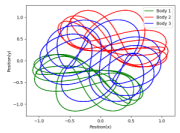
The following table summarizes the results and the categorization of the solutions.

Conclusion

In conclusion, it was found that all three-body solutions are significantly affected by the introduction of a fourth body with any additional mass, even if it is 0.1% of the other bodies. The majority of the solutions experienced a drastic change in the configuration of the original system while maintaining a gravitational binding among the bodies. However, this was only on a certain range of masses. In addition, some solutions had only short ranges (or certain masses) where the solution either showed less chaotic configurations (like the Figure-8 Solution) or failed at all (Bumblebee Solution) among wide ranges of stability or instability. In addition, changing the position of the fourth body or its initial velocity led to complete chaos in the system. By further studying the stability of these chaotic systems with more computational power, we can grasp a better understanding of these complex systems and further work on the expanded n-body problem and incorporate the effects on the gravitational waves.

ACKNOWLEDGEMENT

First and foremost, I'd like to thank PJ Tuckman, my mentor, for his continuous guidance and support throughout the whole project. The information and review he provided were unwavering. I'd also like to thank Kabir Hiranandani, my program manager, for assisting me through the program with the tasks and technicalities. Finally, I'd like to thank the writing center for their support and help with the writing process through the paper.

Solution Name	*Image of the System*	*Range of Stability*	*Category*
Figure-8		[0.001 – 0.01]	Stable over the full range with considerable deformation from the original configuration except for points 0.00163, 0.00170, 0.00260, 0.00830
Butterfly-I		×	Unstable
Butterfly-II		[0.001 – 0.0019]	Stable over the range with great chaos in the topological configuration
Bumblebee		[0.001 – 0.0031]	Stable over the range and exhibited a very chaotic configuration at the points 0.0021, 0.0023, 0.0025, 0.0028, 0.003
Dragonfly		[0.001 – 0.0021]	Stable over the range
Goggles		×	Unstable
Moth-I		[0.001 – 0.0041] – 0.0035, 0.0036	Stable over the specified range except for the two points 0.0035, 0.0036
Moth-II		0.0011, 0.0012	Unstable over the range except for the two points 0.0011, 0.0012

References

1 J. Laskar, *arXiv.org*, 2012.

2 H. Poincaré, *Les méthodes nouvelles de la mécanique céleste. Tome 1*, Lilliad - Université De Lille - Sciences Et Technologies, 2015.

3 S. R. Bistafa, *Euleriana*, 2021, **1**, 181–187.

4 R. Broucke, *Celestial Mechanics*, 1975, **12**, 439–462.

5 C. Moore, *Physical Review Letters*, 1993, **70**, 3675–3679.

6 M. Šuvakov and V. Dmitrašinović, *Physical Review Letters*, 2013, **110**, year.

7 M. C. Gutzwiller and J. V. José, *Chaos in classical and quantum mechanics*, Springer eBooks, 1990.

8 R. Montgomery, *Ergodic Theory and Dynamical Systems*, 2007, **27**, 1933–1946.

9 M. Šuvakov and V. Dmitrašinović, *American Journal of Physics*, 2014, **82**, 609–619.

10 R. Montgomery, *The American Mathematical Monthly*, 2015, **122**, 299.

11 S. community, *scipy.integrate.solve*$_i$*vp*$^-$*SciPyv*1.11.4*Manual*, 2024.*S*.*M*.*Poleshchikov*, Cosmic Research, 2018, ***56***, 151 − −163.

13 M. Šuvakov and V. Dmitrašinović, *Three-body Gallery*, 2013, `http://three-body.ipb.ac.rs/`.

Appendix

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
#Solutions List (initial conditions)
y01 = [-1      ,#x1
  0              ,#y1
  1              ,#x2
  0              ,#y2"
  0                ,#x3
  0              ,#y3
  0              ,#x4
  0.5            ,#y4
  0.347111        ,#vx1
  0.532728        ,#vy1
  0.347111        ,#vx2
  0.532728        ,#vy2
  -2 * 0.347111   ,#vx3
  -2 * 0.532728   ,#vy3
  0               ,#vx4
  0                #vy4
]
y02 = [-1      ,#x1
  0              ,#y1
  1              ,#x2
  0              ,#y2
  0                ,#x3
  0              ,#y3
  0              ,#x4
  0.5            ,#y4
  0.306893     ,#vx1
  0.125507        ,#vy1
  0.306893        ,#vx2
  0.125507        ,#vy2
  -2 * 0.306893   ,#vx3
  -2 * 0.125507   ,#vy3
  0               ,#vx4
  0                #vy4
]
y03 = [-1      ,#x1
  0              ,#y1
  1              ,#x2
  0              ,#y2
```

```
  0                ,#x3
  0              ,#y3
  0              ,#x4
  0.5            ,#y4
  0.392955     ,#vx1
  0.097579       ,#vy1
  0.392955       ,#vx2
  0.097579       ,#vy2
  -2 * 0.392955  ,#vx3
  -2 * 0.097579  ,#vy3
  0              ,#vx4
  0               #vy4
]
y04 = [-1      ,#x1
  0              ,#y1
  1             ,#x2
  0              ,#y2
  0                ,#x3
  0              ,#y3
  0              ,#x4
  0.5            ,#y4
  0.184279     ,#vx1
  0.587188       ,#vy1
  0.184279       ,#vx2
  0.587188       ,#vy2
  -2 * 0.184279  ,#vx3
  -2 * 0.587188  ,#vy3
  0              ,#vx4
  0               #vy4
]
y05 = [-1      ,#x1
  0              ,#y1
  1             ,#x2
  0              ,#y2
  0                ,#x3
  0              ,#y3
  0              ,#x4
  0.5            ,#y4
  0.080584     ,#vx1
  0.588836       ,#vy1
  0.080584       ,#vx2
  0.588836       ,#vy2
  -2 * 0.080584  ,#vx3
  -2 * 0.588836  ,#vy3
```

```
  0                ,#vx4
  0                 #vy4
]
y06 = [-1       ,#x1
  0               ,#y1
  1              ,#x2
  0               ,#y2
  0                 ,#x3
  0               ,#y3
  0               ,#x4
  0.5             ,#y4
  0.083300      ,#vx1
  0.127889        ,#vy1
  0.083300        ,#vx2
  0.127889        ,#vy2
  -2 * 0.083300   ,#vx3
  -2 * 0.127889   ,#vy3
  0                ,#vx4
  0                 #vy4
]
y07 = [-1       ,#x1
  0               ,#y1
  1              ,#x2
  0               ,#y2
  0                 ,#x3
  0               ,#y3
  0               ,#x4
  0.5             ,#y4
  0.464445      ,#vx1
  0.396060        ,#vy1
  0.464445        ,#vx2
  0.396060        ,#vy2
  -2 * 0.464445   ,#vx3
  -2 * 0.396060   ,#vy3
  0                ,#vx4
  0                 #vy4
]
y08 = [-1       ,#x1
  0               ,#y1
  1              ,#x2
  0               ,#y2
  0                 ,#x3
  0               ,#y3
  0               ,#x4
```

```
  0.5            ,#y4
  0.439166    ,#vx1
  0.452968       ,#vy1
  0.439166       ,#vx2
  0.452968       ,#vy2
  -2 * 0.439166  ,#vx3
  -2 * 0.452968  ,#vy3
  0              ,#vx4
  0               #vy4
]
solutionList = [y01, y02, y03, y04, y05, y06, y07, y08]
inp = int(input("Choose a solution from 1 to 8: "))
y0 = solutionList[inp - 1]
# The masses of the 4 bodies
m1 = 1
m2 = 1
m3 = 1
m4 = 0.0053
#0.0016363636363636363
# Time steps and the total number of steps
N = 670000
T = 0.001
#Definition of the Function
def ThreeBody(t,y, m1, m2, m3, m4):
    f = np.zeros(16)
  # The velocities of the three bodies
    f[0] = y[8]
    f[1] = y[9]
    f[2] = y[10]
    f[3] = y[11]
    f[4] = y[12]
    f[5] = y[13]
    f[6] = y[14]
    f[7] = y[15]
  # The x and y accelerations of each object respectively
    f[8] = -m2*(y[0]-y[2])/(((y[0]-y[2])**2+(y[1]-
y[3])**2)**(3/2)) \
         - m3*(y[0]-y[4])/(((y[0]-y[4])**2+(y[1]-
y[5])**2)**(3/2)) \
         - m4*(y[0]-y[6])/(((y[0]-y[6])**2+(y[1]-
y[7])**2)**(3/2))
    f[9] = -m2*(y[1]-y[3])/(((y[0]-y[2])**2+(y[1]-
y[3])**2)**(3/2)) \
         -m3*(y[1]-y[5])/(((y[0]-y[4])**2+(y[1]-y[5])**2)**(3/2))
```

```
\
        - m4*(y[1]-y[7])/(((y[0]-y[6])**2+(y[1]-
y[7])**2)**(3/2))
   f[10] = -m1*(y[2]-y[0])/(((y[2]-y[0])**2+(y[3]-
y[1])**2)**(3/2)) \
           -m3*(y[2]-y[4])/(((y[2]-y[4])**2+(y[3]-
y[5])**2)**(3/2)) \
           -m4*(y[2]-y[6])/(((y[2]-y[6])**2+(y[3]-
y[7])**2)**(3/2))
   f[11] = -m1*(y[3]-y[1])/(((y[2]-y[0])**2+(y[3]-
y[1])**2)**(3/2)) \
        -m3*(y[3]-y[5])/(((y[2]-y[4])**2+(y[3]-y[5])**2)**(3/2))
\
        -m4*(y[3]-y[7])/(((y[2]-y[6])**2+(y[3]-y[7])**2)**(3/2))

   f[12]= -m1*(y[4]-y[0])/(((y[4]-y[0])**2+(y[5]-
y[1])**2)**(3/2)) \
        -m2*(y[4]-y[2])/(((y[4]-y[2])**2+(y[5]-y[3])**2)**(3/2))
\
        -m4*(y[4]-y[6])/(((y[4]-y[6])**2+(y[5]-y[7])**2)**(3/2))
   f[13]= -m1*(y[5]-y[1])/(((y[4]-y[0])**2+(y[5]-
y[1])**2)**(3/2)) \
        -m2*(y[5]-y[3])/(((y[4]-y[2])**2+(y[5]-y[3])**2)**(3/2))
\
        -m4*(y[5]-y[7])/(((y[4]-y[6])**2+(y[5]-y[7])**2)**(3/2))
   f[14] = -m1*(y[6]-y[0])/(((y[6]-y[0])**2+(y[7]-
y[1])**2)**(3/2)) \
        -m2*(y[6]-y[2])/(((y[6]-y[2])**2+(y[7]-y[3])**2)**(3/2))
\
        -m3*(y[6]-y[4])/(((y[6]-y[4])**2+(y[7]-y[5])**2)**(3/2))
   f[15] = -m1*(y[7]-y[1])/(((y[6]-y[0])**2+(y[7]-
y[1])**2)**(3/2)) \
        -m2*(y[7]-y[3])/(((y[6]-y[2])**2+(y[7]-y[3])**2)**(3/2))
\
        -m3*(y[7]-y[5])/(((y[6]-y[4])**2+(y[7]-y[5])**2)**(3/2))
   return f
#Solving for the positions of all bodies
t = np.linspace(0,N*T,N)
solution = solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9,
args = (m1, m2, m3, m4))
stable_solutions_m = []
unstable_solutions_m = []
stable_solutions_px= []
unstable_solutions_px = []
```

```
stable_solutions_py= []
unstable_solutions_py = []
stable_solutions_vx= []
unstable_solutions_vx = []
stable_solutions_vy= []
unstable_solutions_vy = []
func_inpt = ['m','px','py','vx','vy']
def plt_4bodies():
    plt.plot(solution.y[0],solution.y[1],'-g', label = "Body 1")
#Positions body 1
    plt.plot(solution.y[2],solution.y[3],'-r', label = "Body 2")
#Positions body 2
    plt.plot(solution.y[4],solution.y[5],'-b', label = "Body 3")
#Positions body 3
    plt.plot(solution.y[6],solution.y[7],'-y', label = "Body 4")
#Positions body 4
    plt.legend()
    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    plt.ylabel("y")
    plt.xlabel("x")
    plt.show()
    plt.plot(t, solution.y[0],'black', label = 'Body 1')
#Position-x body 1
    plt.plot(t, solution.y[2],'b', label = 'Body 2') #Position-x
body 2
    plt.plot(t, solution.y[4],'-r', label = 'Body 3')#Position-x
body 3
    plt.plot(t, solution.y[6], '-y', label = 'Body 4')#Position-
x body 4
    plt.legend(loc='upper right')
    plt.ylabel("Position(x)")
    plt.xlabel("time")
    plt.xlim(0, 30)
    plt.ylim(-5, 5)
    plt.show()
    plt.plot(t,solution.y[1],'black', label = 'Body 1')
#Position-y body 1
    plt.plot(t,solution.y[3],'b', label = 'Body 2') #Position-y
body 2
    plt.plot(t,solution.y[5],'-r', label = 'Body 3')#Position-y
body 3
    plt.plot(t, solution.y[7], '-y', label = 'Body 4')#Position-
y body 4
```

```
    plt.legend(loc='upper right')
    plt.ylabel("Position(y)")
    plt.xlabel("time")
    plt.xlim(0, 30)
    plt.ylim(-5, 5)
    plt.show()
g = input("Choose one of the following arguments to do your
testing: m, px, py, vx, vy")
if g in func_inpt:
    print("we'll process your order.")
stable_masses = np.arange(0.001,0.01,0.0001)
print(stable_masses)
positions_x = np.arange(0.1,1,0.01)
print(positions_x)
positions_y = np.arange(0.1,1,0.01)
print(positions_y)
velocities_x = np.arange(0.1,1,0.01)
print(velocities_x)
velocities_y = np.arange(0.1,1,0.01)
print(velocities_y)
def VectorStability(g):
    x = 50
    distance = []
    if g == 'm':
        for i in stable_masses:
            used_mass = i
            print(i)
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, i))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_m.append(used_mass)
```

```
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_m.append(used_mass)
                    break
            return output
    elif g == 'px':
        for i in positions_x:
            used_pos_x = i
            print(i)
            y0[6] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_px.append(used_pos_x)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_px.append(used_pos_x)
                    break
            return output
    elif g == 'py':
        for i in positions_y:
            used_pos_y = i
            print(i)
            y0[7] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
```

```
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_py.append(used_pos_y)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_py.append(used_pos_y)
                    break
            return output
    elif g == 'vx':
        for i in velocities_x:
            used_vel_x = i
            print(i)
            y0[14] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_vx.append(used_vel_x)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_vx.append(used_vel_x)
```

```
                    break
            return output
    elif g == 'vy':
        for i in velocities_y:
            used_vel_y = i
            print(i)
            y0[15] = i
            solution =
solve_ivp(ThreeBody,[0,670],y0,t_eval=t,rtol=1e-9, args = (m1,
m2, m3, m4))
            distance.append(np.sqrt((solution.y[2][N-1] -
solution.y[0][N-1])**2 + (solution.y[3][N-1] - solution.y[1][N-
1])**2)) #r12
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[0][N-1])**2 + (solution.y[5][N-1] - solution.y[1][N-
1])**2)) #r13
            distance.append(np.sqrt((solution.y[4][N-1] -
solution.y[2][N-1])**2 + (solution.y[5][N-1] - solution.y[3][N-
1])**2)) #r23
            plt_4bodies()
            for i in distance:
                if i > 10:
                    output = print("This system is unstable")
                    unstable_solutions_vy.append(used_vel_y)
                    break
                else:
                    output = print("This system is stable")
                    stable_solutions_vy.append(used_vel_y)
                    break
            return output

VectorStability(g)
if g == 'm':
    print(f"stable solutions:{stable_solutions_m}")
    print(f"unstable solutions:{unstable_solutions_m}")
elif g == 'px':
    print(f"stable solutions:{stable_solutions_px}")
    print(f"unstable solutions:{unstable_solutions_px}")
elif g == 'py':
    print(f"stable solutions:{stable_solutions_py}")
    print(f"unstable solutions:{unstable_solutions_py}")
elif g == 'vx':
    print(f"stable solutions:{stable_solutions_vx}")
    print(f"unstable solutions:{unstable_solutions_vx}")
```

```
elif g == 'vy':
    print(f"stable solutions:{stable_solutions_vy}")
    print(f"unstable solutions:{unstable_solutions_vy}")
```