دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

Final simulation project

Autor : Nima Yadollahi – 40111046

Program : Chaos Effect in the Three-Body Problem

Course : Fundamentals of Numerical Simulation

Teacher : Dr. Naem Abadi

Year 1403/1404

Table of Contents

Introduction

1.1 *Explanation of Chaos Theory*

Chaos theory is an interdisciplinary area of scientific study and a branch of mathematics. It focuses on underlying patterns and deterministic laws of dynamical systems that are highly sensitive to initial conditions.

Sensitivity to initial conditions is popularly known as the "butterfly effect", so-called because of the title of a paper given by Edward Lorenz in 1972 to the American Association for the Advancement of Science in Washington, D.C., entitled Predictability: Does the Flap of a Butterfly's Wings in Brazil set off a Tornado in Texas?

The butterfly effect, an underlying principle of chaos, describes how a small change in one state of a deterministic nonlinear system can result in large differences in a later state, like a strong wind in one country causing a tornado in another country.

Chaos often arises in nonlinear systems, where the relationship between inputs and outputs is not proportional.

A consequence of sensitivity to initial conditions is that if we start with a limited amount of information about the system, then beyond a certain time, the system would no longer be predictable.

Chaos theory primarily deals with deterministic systems, systems governed by specific rules or equations. Unlike random systems, where outcomes are fundamentally unpredictable,

deterministic systems have predictable behaviors based on initial conditions. However, due to the extreme sensitivity to initial conditions, the future states of chaotic systems can become unpredictable in practice.

The amount of time for whichh the b”havi’r of a chaotic system can be effectively predicted depends on three things: how much uncertainty can be tolerated in the forecast, how accurately its current state can be measured, and a time scale depending on the dynamics of the system, called the Lyapunov time. Some examples of Lyapunov times are: chaotic electrical circuits, about 1 millisecond; weather systems, a few days; the inner solar system, 4 to 5 million years.

Although chaos theory was born from observing weather patterns, it has become applicable to a variety of other situations like mathematics, biology, computer science, economics, engineering, economics, physics, and transportation.

We can say that a few categories are listed below with examples.

Computer science

- ✓ Robotics systems use chaos theory to improve adaptability and learning in unpredictable environments.
- ✓ Algorithms and Optimization
- ✓ Neural networks often exhibit chaotic behavior, influencing the design of machine learning algorithms.
- ✓ Random Number Generation

Biology and Medicine

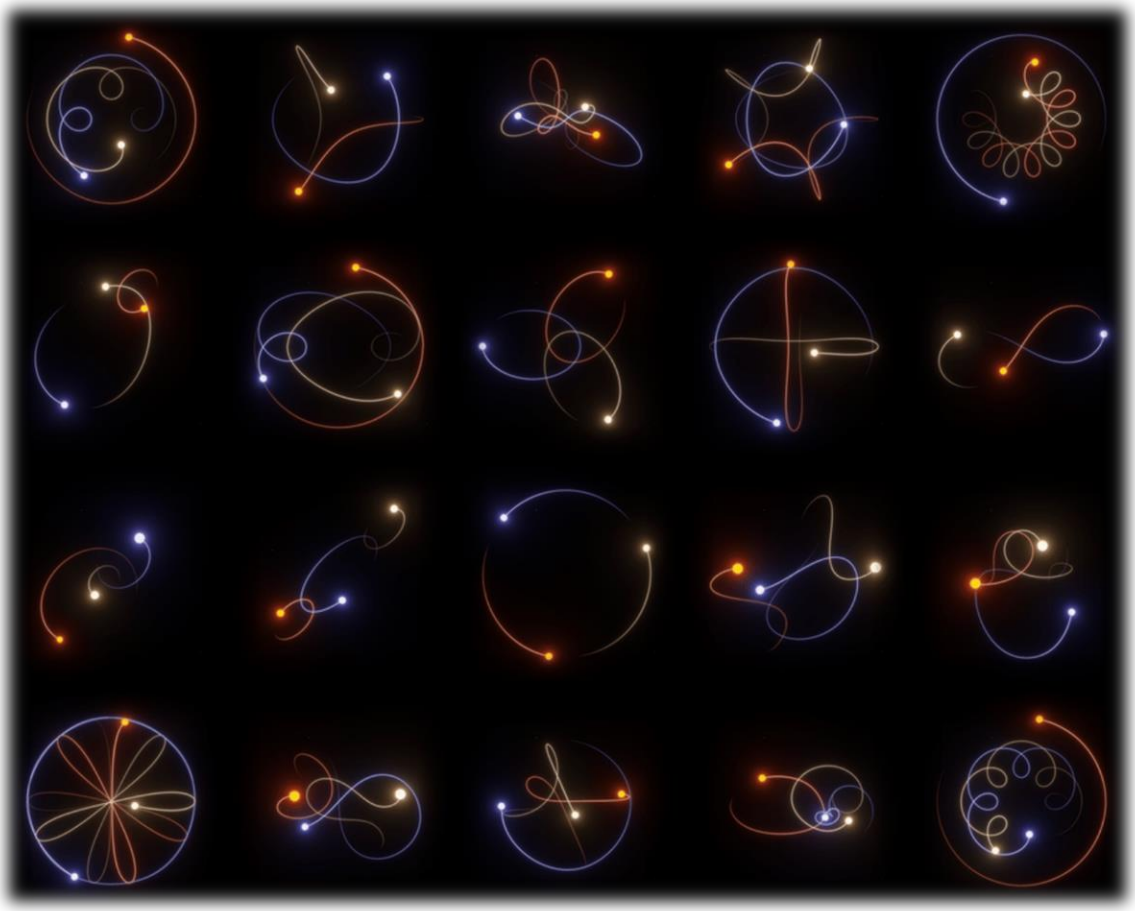
- ✓ The chaotic dynamics of heart rhythms are studied to understand arrhythmias (irregular heartbeats).
- ✓ Neural dynamics in the brain exhibit chaotic behavior. Understanding this helps in diagnosing and treating disorders such as epilepsy.

Economics

- ✓ Financial markets often display chaotic behavior due to the interaction of numerous nonlinear factors.
- ✓ Applications include optimizing resource allocation and market stability.

Transportation

- ✓ Chaos theory is used in traffic flow modeling and optimization, helping design better transportation systems.



1.2 *Three-Body Problem*

The Three-Body Problem in physics refers to the challenge of predicting the motions of three celestial bodies under their mutual gravitational influence and point masses. This problem arises in classical mechanics and is a generalization of the simpler Two-Body Problem, which can be solved exactly.

Unlike the two-body problem, the three-body problem has no general closed-form solution, meaning there is no equation that always solves it. When three bodies orbit each other, the resulting dynamical system is chaotic for most initial conditions. Because there are no solvable equations for most three-body

systems, the only way to predict the motions of the bodies is to estimate them using numerical methods.

The addition of a third body leads to chaotic dynamics, meaning that small changes in initial conditions can result in vastly different outcomes.

From the Newtonian equations of motion for vector positions, we know that:

$$\ddot{r}_1 = -Gm_2 \frac{r_1 - r_2}{|r_1 - r_2|^3} - Gm_3 \frac{r_1 - r_3}{|r_1 - r_3|^3}$$
$$\ddot{r}_2 = -Gm_3 \frac{r_2 - r_3}{|r_2 - r_3|^3} - Gm_1 \frac{r_2 - r_1}{|r_2 - r_1|^3}$$
$$\ddot{r}_3 = -Gm_1 \frac{r_3 - r_1}{|r_3 - r_1|^3} - Gm_2 \frac{r_3 - r_2}{|r_3 - r_2|^3}$$

- where G is the gravitational constant.
- r and m represent position and mass.

There is no general closed-form solution to the three-body problem, but we can use the computer so the problem may be solved to arbitrarily high precision using numerical integration. There have been attempts to create computer programs that numerically solve the three-body problem involving both electromagnetic and gravitational interactions and incorporating modern theories of physics such as special relativity. In addition,

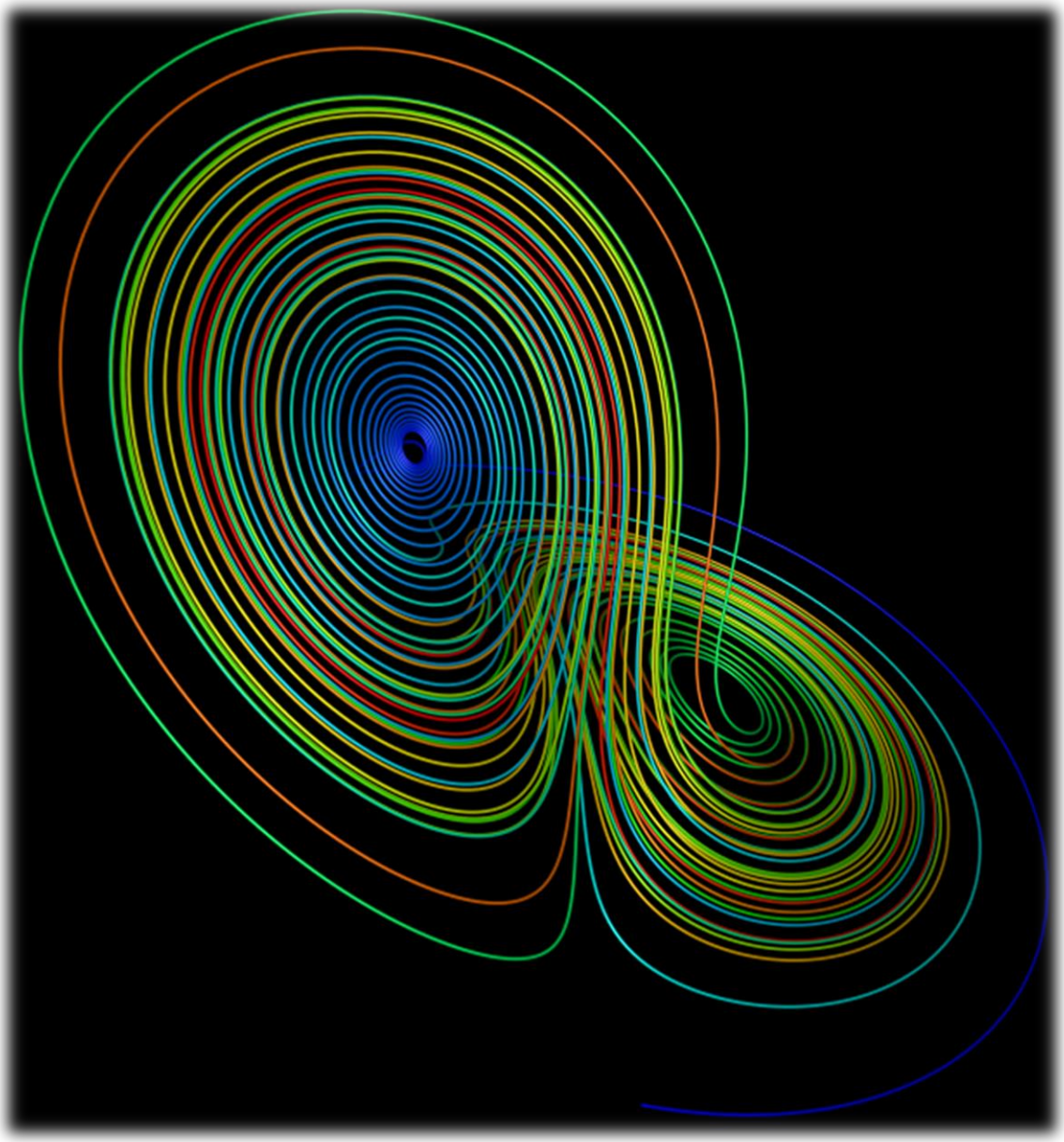
using the theory of random walks, an approximate probability of different outcomes may be computed.

The term "three-body problem" is sometimes used in the more general sense to refer to any physical problem involving the interaction of three bodies.

A quantum-mechanical analogue of the gravitational three-body problem in classical mechanics is the helium atom, in which a helium nucleus and two electrons interact according to the inverse-square Coulomb interaction. Like the gravitational three-body problem, the helium atom cannot be solved exactly.

The gravitational three-body problem has also been studied using general relativity. Physically, a relativistic treatment becomes necessary in systems with very strong gravitational fields, such as near the event horizon of a black hole.

If we expand our review, we reach the n-body problem, which describes how n objects move under one of the physical forces, such as gravity. Atoms, ions, and molecules can be treated in terms of the quantum n-body problem.



1.3 *Overview of the Lorenz system*

The Lorenz system is a system of ordinary differential equations first studied by mathematician and meteorologist Edward Lorenz. In particular, the Lorenz attractor is a set of chaotic solutions of the Lorenz system. The shape of the Lorenz attractor itself, when plotted in phase space, may also be seen to resemble a butterfly.

In 1963, Edward Lorenz, with the help of Ellen Fetter who was responsible for the numerical simulations and figures, and Margaret Hamilton who helped in the initial, numerical computations leading up to the findings of the Lorenz model, developed a simplified mathematical model for atmospheric convection. The model is a system of three ordinary differential equations now known as the Lorenz equations:

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(p - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

- x is proportional to the rate of convection, y to the horizontal temperature variation, and z to the vertical temperature variation.
- σ is the **Prandtl number**, which is a dimensionless number that describes the relative thickness of the velocity boundary layer to the thermal boundary layer.
- ρ is the **Rayleigh number**, a dimensionless quantity that represents the driving force for convection.
- β is a parameter related to the physical properties of the fluid, specifically the aspect ratio of the convection cells.

In the context of the three-body problem, x, y, and z correspond to position, and σ, ρ, β correspond to any variables under which the system becomes chaotic, like angular momentum, mass, torque, etc.

Chaos in the Three-Body Problem

2.1 *Simulations of three-body problem*

After all the explanations, it is time to simulate

Libraries

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from matplotlib.animation import FuncAnimation
```

We use `solve_ivp` from the SciPy library to solve a set of differential equations problems.

initial conditions

[34]:
```
m1 = 5
m2 = 1
m3 = 0.5
mass = (m1,m2,m3)

c = 1e3  #speed of light

# Position
position1 = [1, 2, 0]
position2 = [2, -1, 1]
position3 = [0.0, 1, -2]

# Velocity
velocity1 = [0.3, -0.5, 0.8]
velocity2 = [-0.4, 0.9, -0.2]
velocity3 = [0.5, 0.0, -0.6]

initial_conditions = np.array([
    position1, position2, position3,
    velocity1, velocity2, velocity3
]).ravel()
```

In this case, to simplify the simulation, we use dimensionless parameters.

Functions

[5]:
```
def classical_system(t, S, m1, m3, m2):
    r1, r2, r3 = S[0:3], S[3:6], S[6:9]
    dr1_dt, dr2_dt, dr3_dt = S[9:12], S[12:15], S[15:18]

    v1, v2, v3 = dr1_dt, dr2_dt, dr3_dt

    r12 = np.linalg.norm(r2 - r1)
    r13 = np.linalg.norm(r3 - r1)
    r23 = np.linalg.norm(r3 - r2)

    dv1_dt = m3*(r3 - r1)/r13**3 + m2*(r2 - r1)/r12**3
    dv2_dt = m3*(r3 - r2)/r23**3 + m1*(r1 - r2)/r12**3
    dv3_dt = m1*(r1 - r3)/r13**3 + m2*(r2 - r3)/r23**3

    return np.array([v1, v2, v3, dv1_dt, dv2_dt, dv3_dt]).ravel()
```

Results

[22]:
```
time_s, time_e = 0, 10
steps = np.linspace(time_s, time_e, 1001)
time = cla_solution.t
```

Classical Results

[18]:
```
cla_solution = solve_ivp(
    fun=classical_system,
    t_span=(time_s, time_e),
    y0=initial_conditions,
    t_eval=steps,
    args=(m1, m2, m3)
)
cla_solution
```

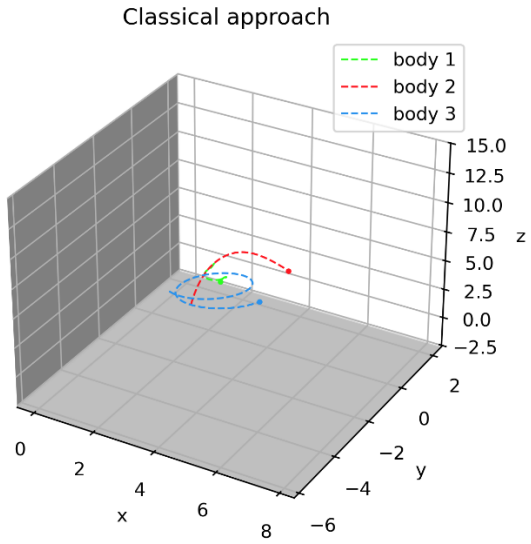
[18]:
```
  message: The solver successfully reached the end of the integration interval.
  success: True
   status: 0
        t: [ 0.000e+00  1.000e-02 ...  9.990e+00  1.000e+01]
        y: [[ 1.000e+00  1.003e+00 ...  3.211e+00  3.212e+00]
            [ 2.000e+00  1.995e+00 ... -1.336e+00 -1.339e+00]
            ...
            [ 0.000e+00  3.233e-03 ... -3.253e-02 -3.499e-02]
            [-6.000e-01 -5.930e-01 ...  1.307e+00  1.313e+00]]
      sol: None
 t_events: None
 y_events: None
     nfev: 230
     njev: 0
      nlu: 0
```

[24]:
```
p1x_cla = cla_solution.y[0]
p1y_cla = cla_solution.y[1]
p1z_cla = cla_solution.y[2]

p2x_cla = cla_solution.y[3]
p2y_cla = cla_solution.y[4]
p2z_cla = cla_solution.y[5]

p3x_cla = cla_solution.y[6]
p3y_cla = cla_solution.y[7]
p3z_cla = cla_solution.y[8]
```

Plotting

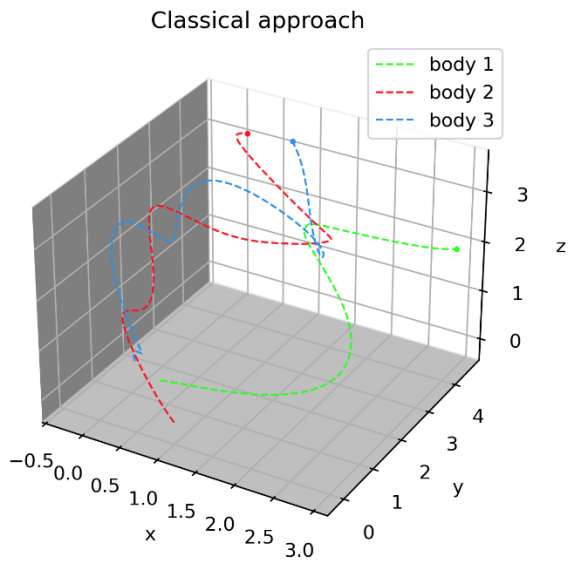
```
[62]: fig, ax = plt.subplots(subplot_kw={"projection":"3d"})

body1, = ax.plot(p1x_cla, p1y_cla, p1z_cla, label='body 1', color = '#2aff1f' ,linestyle='--', linewidth=1)
body2, = ax.plot(p2x_cla, p2y_cla, p2z_cla, label='body 2', color = '#fe151f' ,linestyle='--', linewidth=1)
body3, = ax.plot(p3x_cla, p3y_cla, p3z_cla, label='body 3', color = '#2a92ed' ,linestyle='--', linewidth=1)

body1_dot, = ax.plot([p1x_cla[-1]], [p1y_cla[-1]], [p1z_cla[-1]], 'o', color = '#2aff1f' , markersize=2)
body2_dot, = ax.plot([p2x_cla[-1]], [p2y_cla[-1]], [p2z_cla[-1]], 'o', color = '#fe151f' , markersize=2)
body3_dot, = ax.plot([p3x_cla[-1]], [p3y_cla[-1]], [p3z_cla[-1]], 'o', color = '#2a92ed' , markersize=2)

ax.xaxis.pane.set_facecolor('black')
ax.yaxis.pane.set_facecolor('white')
ax.zaxis.pane.set_facecolor('grey')

ax.set_title("Classical approach")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
plt.grid()
plt.legend()
plt.savefig(f"Classical Three body {mass}.png",dpi=300)
plt.show()
```

Fig 1.1



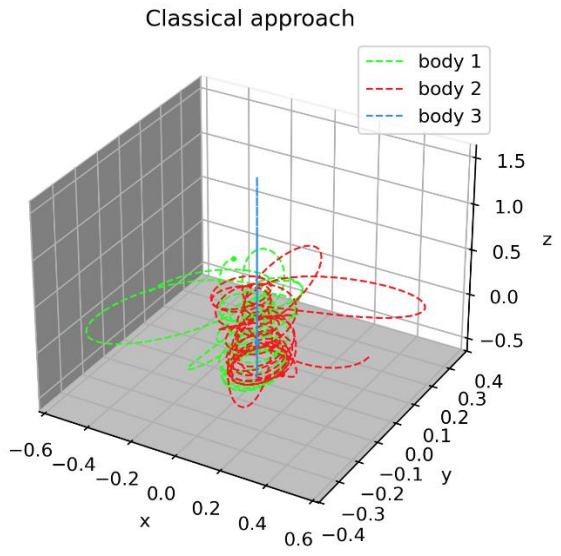
Mass = [5 , 1 , 0.5]
Position 1 = [1 , 2 , 0]
Position 2 = [2 , -1 , 1]
Position 3 = [0 , 1 , -2]
Velocity 1 = [0.3 , -0.5 , 0.8]
Velocity 2 = [-0.4 , 0.9 , -0.2]
Velocity 3 = [0.5 , 0 , -0.6]

Fig 1.2



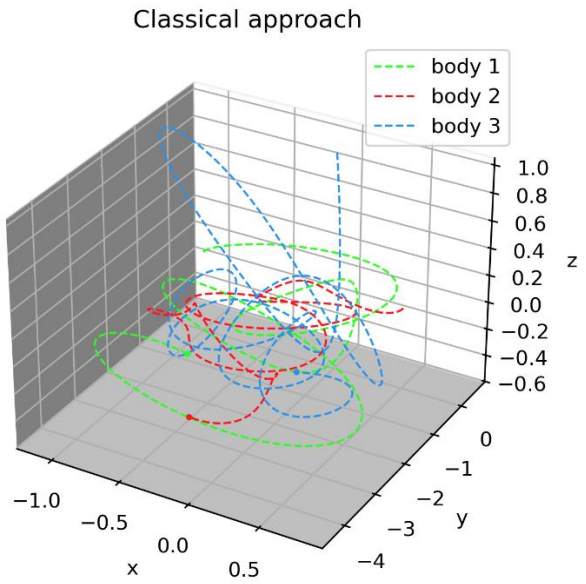
Mass = [2 , 1 , 1]
Position 1 = [0.4 , 1 , 0]
Position 2 = [1 , 0 , 0]
Position 3 = [0.5 , 0 , 1]
Velocity 1 = [0.5 , 1 , -0.5]
Velocity 2 = [-0.6 , 0 , 1]
Velocity 3 = [0.1 , -0.5 , 1]

Fig 1.3



Mass = [1,1,1]
Position 1 = [-0.5 , 0 , 0]
Position 2 = [0.5 , 0 , 0]
Position 3 = [0 , 0 , 1]
Velocity 1 = [0 , 0.347111 , 0]
Velocity 2 = [0 , -0.347111 , 0]
Velocity 3 = [0 , 0 , -0.1]

Fig 1.4



Mass = [3,1,4]
Position 1 = [-1 , 0 , 0]
Position 2 = [0.5 , 0 , 0]
Position 3 = [0 , 0 , 1]
Velocity 1 = [0 , 1 , 0]
Velocity 2 = [0 , -1.5 , 0]
Velocity 3 = [0 , 0 , -0.1]

The four images above are just a sample of the results. We will observe varied outcomes based on different initial conditions, but for the remainder of this discussion, we will focus on these four specific conditions article.

2.2 *relativistic vs. classical approach*

For relativistic correction, we use the formula below:

$$a_i^{rel} = a_i \cdot \left[1 + \sum_{j \neq k} \left(\frac{v_{ij}^2}{c^2} - 4\frac{(m_i + m_j)}{r_{ij}c^2}\right)\right]$$

[60]:
```
def relativistic_system(t, S, m1, m2, m3, c):
    r1, r2, r3 = S[0:3], S[3:6], S[6:9]
    dr1_dt, dr2_dt, dr3_dt = S[9:12], S[12:15], S[15:18]

    v1, v2, v3 = dr1_dt, dr2_dt, dr3_dt

    r12 = np.linalg.norm(r2 - r1)
    r13 = np.linalg.norm(r3 - r1)
    r23 = np.linalg.norm(r3 - r2)

    v12 = np.linalg.norm(v2 - v1)
    v13 = np.linalg.norm(v3 - v1)
    v23 = np.linalg.norm(v3 - v2)

    dv1_dt = m3 * (r3 - r1) / r13**3 + m2 * (r2 - r1) / r12**3
    dv2_dt = m3 * (r3 - r2) / r23**3 + m1 * (r1 - r2) / r12**3
    dv3_dt = m1 * (r1 - r3) / r13**3 + m2 * (r2 - r3) / r23**3

    dv1_dt_rel = dv1_dt * (
        1 + (v13**2 / c**2) - (4 * (m1 + m3) / (r13 * c**2))
        + (v12**2 / c**2) - (4 * (m1 + m2) / (r12 * c**2))
    )
    dv2_dt_rel = dv2_dt * (
        1 + (v23**2 / c**2) - (4 * (m2 + m3) / (r23 * c**2))
        + (v12**2 / c**2) - (4 * (m2 + m1) / (r12 * c**2))
    )
    dv3_dt_rel = dv3_dt * (
        1 + (v13**2 / c**2) - (4 * (m3 + m1) / (r13 * c**2))
        + (v23**2 / c**2) - (4 * (m3 + m2) / (r23 * c**2))
    )

    return np.array([v1, v2, v3, dv1_dt_rel, dv2_dt_rel, dv3_dt_rel]).ravel()
```

```
[ ]: fig, ax = plt.subplots(subplot_kw={"projection":"3d"})

body1, = ax.plot(p1x_rel, p1y_rel, p1z_rel, label='body 1', color = '#2aff1f' ,linestyle='--', linewidth=1)
body2, = ax.plot(p2x_rel, p2y_rel, p2z_rel, label='body 2', color = '#fe151f' ,linestyle='--', linewidth=1)
body3, = ax.plot(p3x_rel, p3y_rel, p3z_rel, label='body 3', color = '#2a92ed' ,linestyle='--', linewidth=1)

body1_dot, = ax.plot([p1x_rel[-1]], [p1y_rel[-1]], [p1z_rel[-1]], 'o', color = '#2aff1f' , markersize=2)
body2_dot, = ax.plot([p2x_rel[-1]], [p2y_rel[-1]], [p2z_rel[-1]], 'o', color = '#fe151f' , markersize=2)
body3_dot, = ax.plot([p3x_rel[-1]], [p3y_rel[-1]], [p3z_rel[-1]], 'o', color = '#2a92ed' , markersize=2)

ax.xaxis.pane.set_facecolor('black')
ax.yaxis.pane.set_facecolor('white')
ax.zaxis.pane.set_facecolor('grey')

ax.set_title("Relativistic approach")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
plt.grid()
plt.legend()
plt.savefig(f"Relativistic Three body {mass}.png",dpi=300)
plt.show()
```

```
[ ]: def update(frame):
    lower_lim = max(0, frame - 300)

    x_current_1 = p1x_rel[lower_lim:frame+1]
    y_current_1 = p1y_rel[lower_lim:frame+1]
    z_current_1 = p1z_rel[lower_lim:frame+1]

    x_current_2 = p2x_rel[lower_lim:frame+1]
    y_current_2 = p2y_rel[lower_lim:frame+1]
    z_current_2 = p2z_rel[lower_lim:frame+1]

    x_current_3 = p3x_rel[lower_lim:frame+1]
    y_current_3 = p3y_rel[lower_lim:frame+1]
    z_current_3 = p3z_rel[lower_lim:frame+1]

    body1.set_data(x_current_1, y_current_1)
    body1.set_3d_properties(z_current_1)

    body1_dot.set_data([x_current_1[-1]], [y_current_1[-1]])
    body1_dot.set_3d_properties([z_current_1[-1]])



    body2.set_data(x_current_2, y_current_2)
    body2.set_3d_properties(z_current_2)

    body2_dot.set_data([x_current_2[-1]], [y_current_2[-1]])
    body2_dot.set_3d_properties([z_current_2[-1]])



    body3.set_data(x_current_3, y_current_3)  -
    body3.set_3d_properties(z_current_3)

    body3_dot.set_data([x_current_3[-1]], [y_current_3[-1]])
    body3_dot.set_3d_properties([z_current_3[-1]])


    return body1, body1_dot, body2, body2_dot, body3, body3_dot
```

```
[ ]: animation = FuncAnimation(fig, update, frames=range(0, len(steps),2), interval=30, blit=True)
animation.save(f"Relativistic Three body {mass}.gif", writer="pillow",fps=30 , dpi=300)
```

Fig 2.1

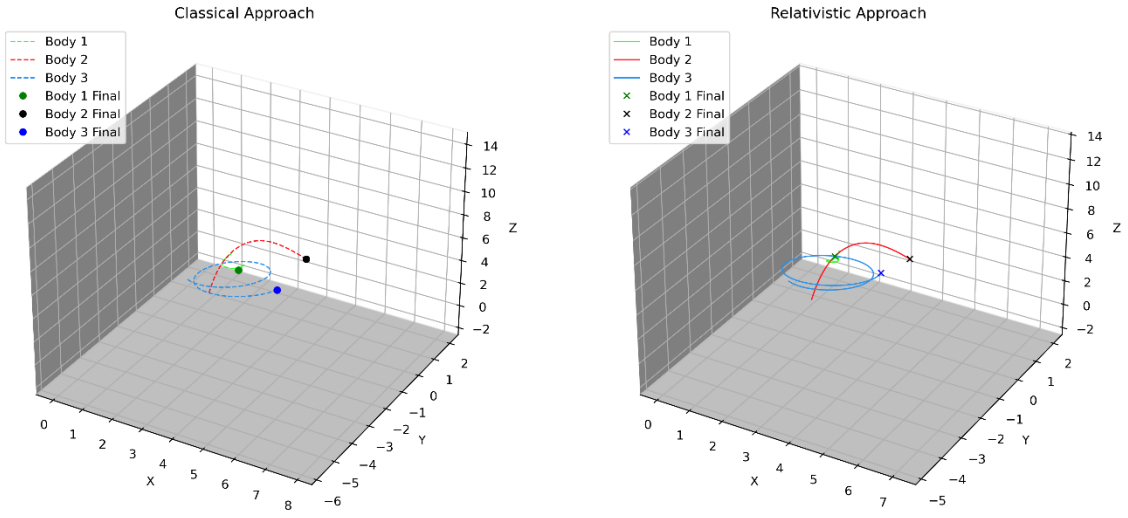


Fig 2.2

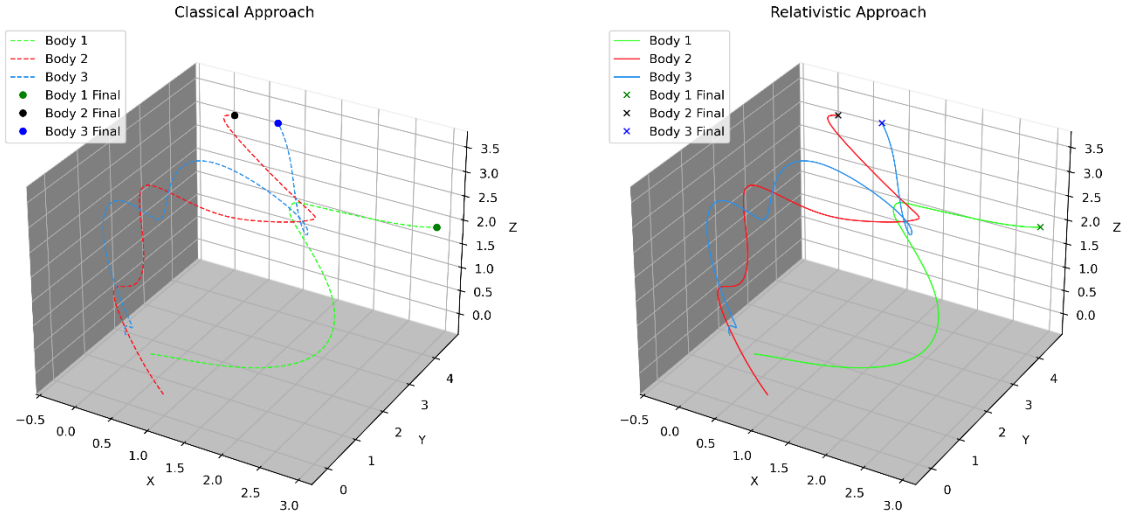


Fig 2.3

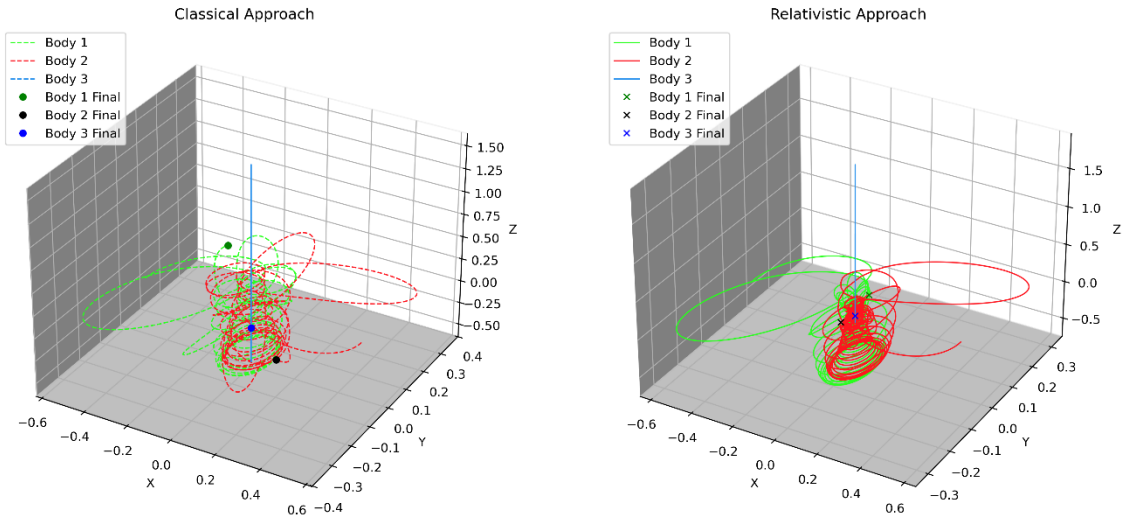
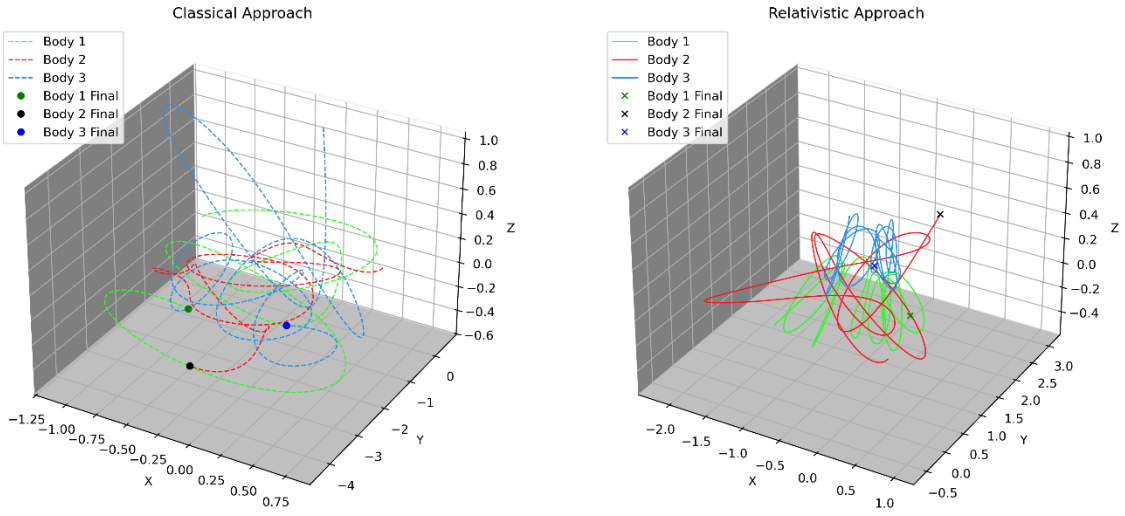


Fig 2.4



As it is clear from the pictures above and the final locations of the particles, relativistic corrections cause changes in the results.

2.3 *Chaotic approach & Lorenz attractor*

Lorenz Attractor

[1643]:
```
x0, y0, z0 = 20.0, 1.0, 21.05
```

[1644]:
```
perturbation = 1e-6
perturbed_position1 = [position1[0] + perturbation, position1[1], position1[2]]
```

[1645]:
```
# Lorenz system parameters
sigma, rho, beta = 10, 28, 1
param = (sigma ,rho ,beta)
```

[1646]:
```
# Initial conditions for both cases
initial_conditions_original = np.array([
    position1, position2, position3,
    velocity1, velocity2, velocity3,
    [x0, y0, z0]
]).ravel()
```

[1647]:
```
initial_conditions_perturbed = np.array([
    perturbed_position1, position2, position3,
    velocity1, velocity2, velocity3,
    [x0, y0, z0]
]).ravel()
```

To solve the Lorenz system of differential equations we need to set some initial parameters like σ, p, β and a perturbation term that shows the sensitivity of the three-body system to initial conditions.

[1648]:
```
def system(t, S, m1, m2, m3, sigma, rho, beta):
    r1, r2, r3 = S[0:3], S[3:6], S[6:9]
    v1, v2, v3 = S[9:12], S[12:15], S[15:18]
    x, y, z = S[18:21]

    r12 = np.linalg.norm(r2 - r1)
    r13 = np.linalg.norm(r3 - r1)
    r23 = np.linalg.norm(r3 - r2)

    dv1_dt = m3*(r3 - r1)/r13**3 + m2*(r2 - r1)/r12**3
    dv2_dt = m3*(r3 - r2)/r23**3 + m1*(r1 - r2)/r12**3
    dv3_dt = m1*(r1 - r3)/r13**3 + m2*(r2 - r3)/r23**3

    # Lorenz system equations
    dx_dt = sigma * (y - x)
    dy_dt = x * (rho - z) - y
    dz_dt = x * y - beta * z

    return np.array([
        v1, v2, v3,
        dv1_dt, dv2_dt, dv3_dt,
        [dx_dt, dy_dt, dz_dt]    # Lorenz equations
    ]).ravel()
```

[1649]:
```
time_s, time_e = 0, 30
steps = np.linspace(time_s, time_e, 1001)
```

[1650]:
```
solution_original = solve_ivp(
    fun=system,
    t_span=(time_s, time_e),
    y0=initial_conditions_original,
    t_eval=steps,
    args=(m1, m2, m3, sigma, rho, beta)
)
```

[1651]:
```
solution_perturbed = solve_ivp(
    fun=system,
    t_span=(time_s, time_e),
    y0=initial_conditions_perturbed,
    t_eval=steps,
    args=(m1, m2, m3, sigma, rho, beta)
)
```

[1652]:
```
p1x_orig, p1y_orig, p1z_orig = solution_original.y[0], solution_original.y[1], solution_original.y[2]
lorenz_x_orig, lorenz_y_orig, lorenz_z_orig = solution_original.y[18], solution_original.y[19], solution_original.y[20]

p1x_pert, p1y_pert, p1z_pert = solution_perturbed.y[0], solution_perturbed.y[1], solution_perturbed.y[2]
lorenz_x_pert, lorenz_y_pert, lorenz_z_pert = solution_perturbed.y[18], solution_perturbed.y[19], solution_perturbed.y[20]
```

```
fig = plt.figure(figsize=(10, 5))

ax1 = fig.add_subplot(121, projection='3d')
ax1.plot(p1x_orig, p1y_orig, p1z_orig, label='Original', color='blue')
ax1.plot(p1x_pert, p1y_pert, p1z_pert, label='Perturbed', color='red')
ax1.set_title('Three-Body Problem')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')
ax1.legend()

ax2 = fig.add_subplot(122, projection='3d')
ax2.plot(lorenz_x_orig, lorenz_y_orig, lorenz_z_orig, label='Original', color='blue')
ax2.plot(lorenz_x_pert, lorenz_y_pert, lorenz_z_pert, label='Perturbed', color='red')
ax2.set_title('Lorenz Attractor')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.set_zlabel('z')
ax2.legend()

plt.tight_layout()
plt.savefig(f"Lorenz {mass} and {param}.png" , dpi = 300)
plt.show()
```

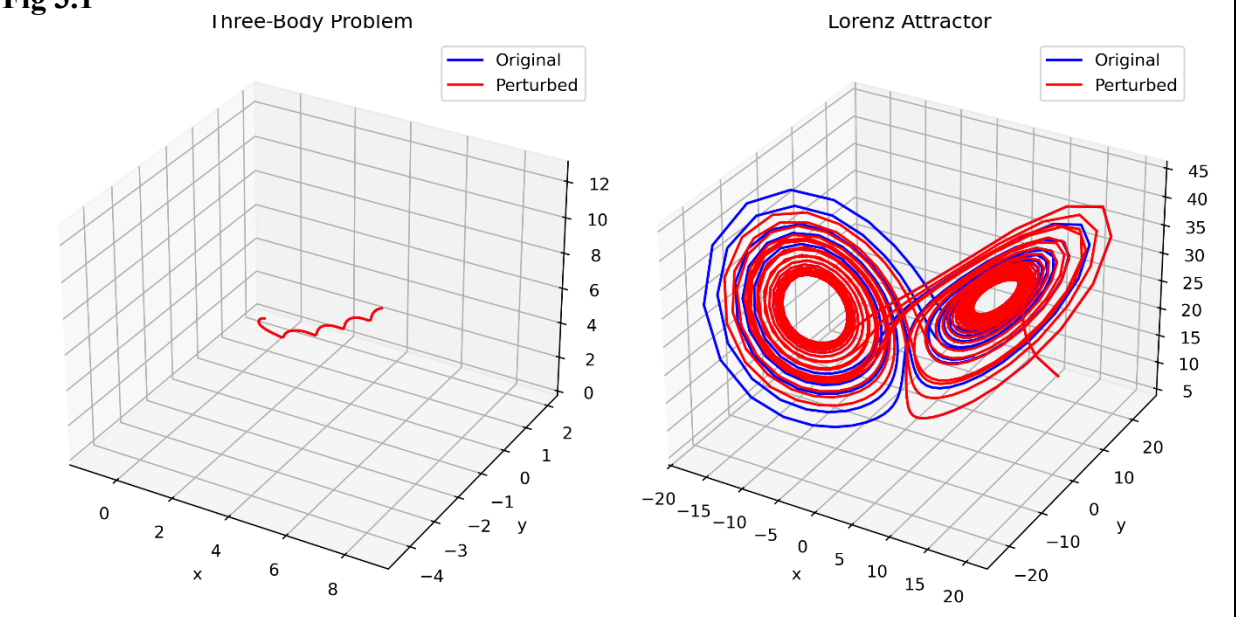
The left plot shows the path of a point mass in the three-body system under the specific initial conditions.

$\sigma = 10$

$p = 28$

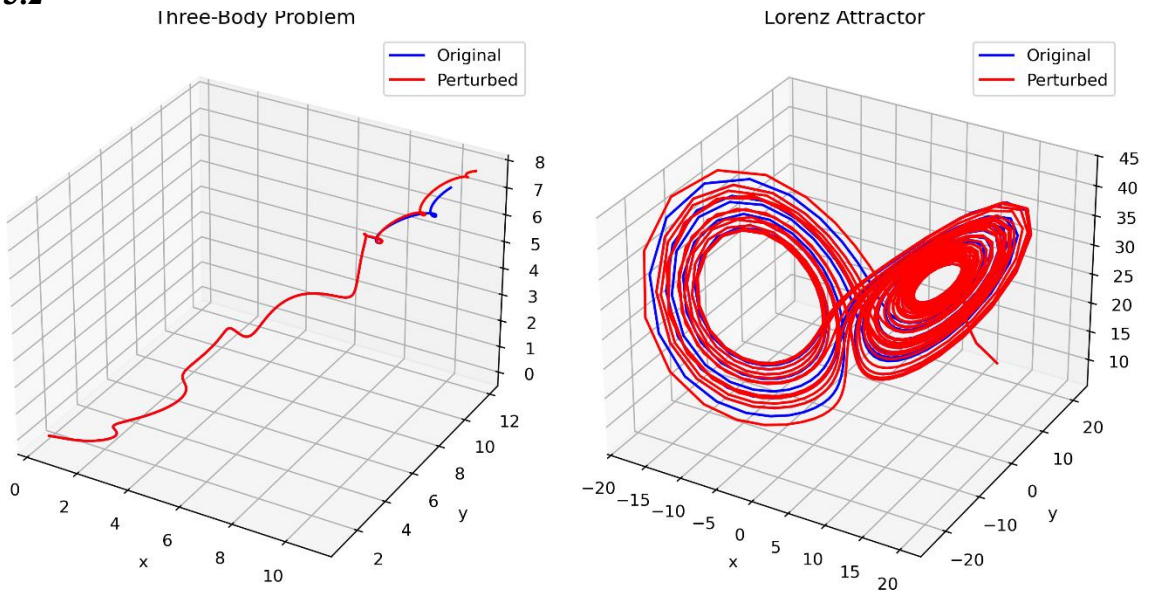
$\beta = 8/3$

Fig 3.1



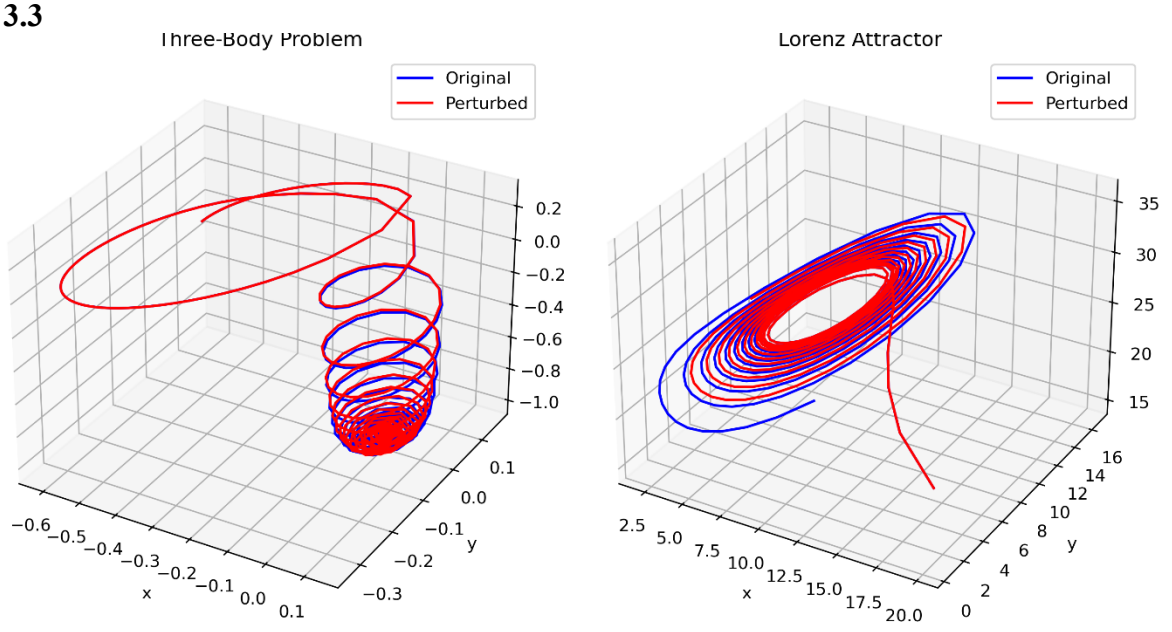
Mass = [5 , 1 , 0.5]
Position 1 = [1 , 2 , 0] /// Position 2 = [2 , -1 , 1] /// Position 3 = [0 , 1 , -2]
Velocity 1 = [0.3 , -0.5 , 0.8] /// Velocity 2 = [-0.4 , 0.9 , -0.2] /// Velocity 3 = [0.5 , 0 , -0.6]

Fig 3.2



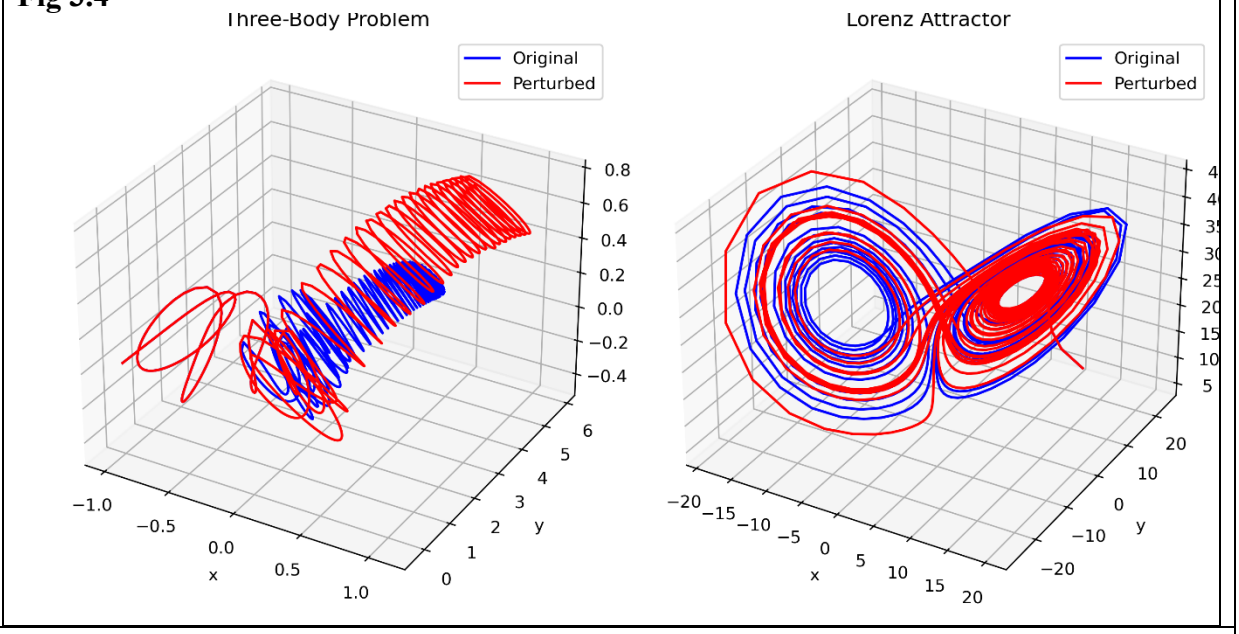
Mass = [2 , 1 , 1]
Position 1 = [0.4 , 1 , 0] /// Position 2 = [1 , 0 , 0] /// Position 3 = [0.5 , 0 , 1]
Velocity 1 = [0.5 , 1 , -0.5] /// Velocity 2 = [-0.6 , 0 , 1] /// Velocity 3 = [0.1 , -0.5 , 1]

Fig 3.3



Mass = [1,1,1]
Position 1 = [-0.5 , 0 , 0] /// Position 2 = [0.5 , 0 , 0] /// Position 3 = [0 , 0 , 1]
Velocity 1 = [0 , 0.347111 , 0] /// Velocity 2 = [0 , -0.347111 , 0] /// Velocity 3 = [0 , 0 , -0.1]

Fig 3.4



Mass = [3,1,4]
Position 1 = [-1 , 0 , 0] /// Position 2 = [0.5 , 0 , 0] /// Position 3 = [0 , 0 , 1]
Velocity 1 = [0 , 1 , 0] /// Velocity 2 = [0 , -1.5 , 0] /// Velocity 3 = [0 , 0 , -0.1]

$\sigma = 20$

$p = 28$

$\beta = 8/3$

Fig 4.1

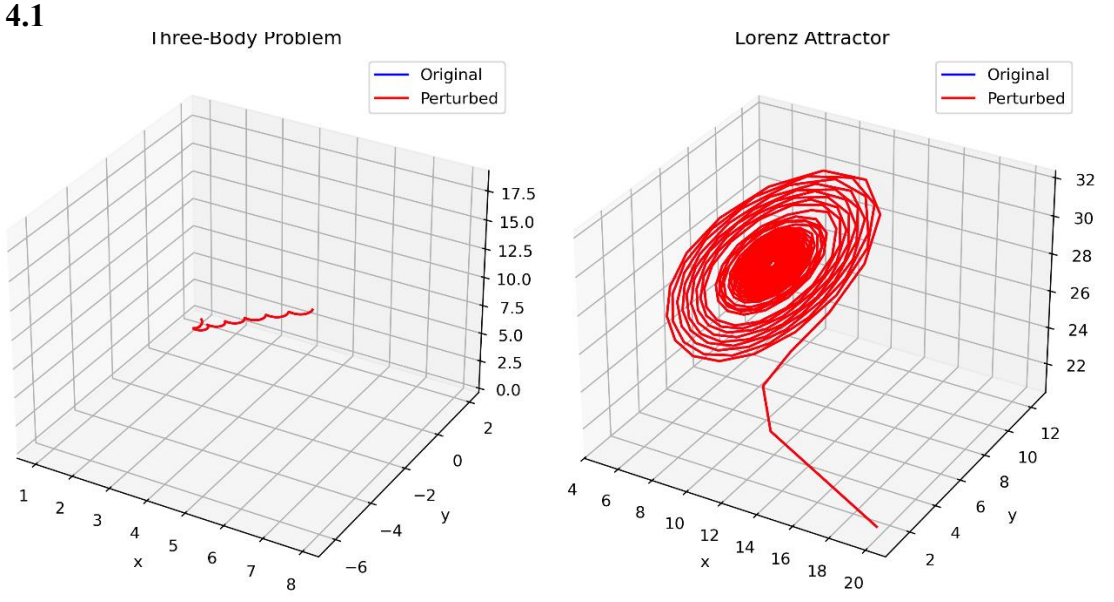


Fig 4.2

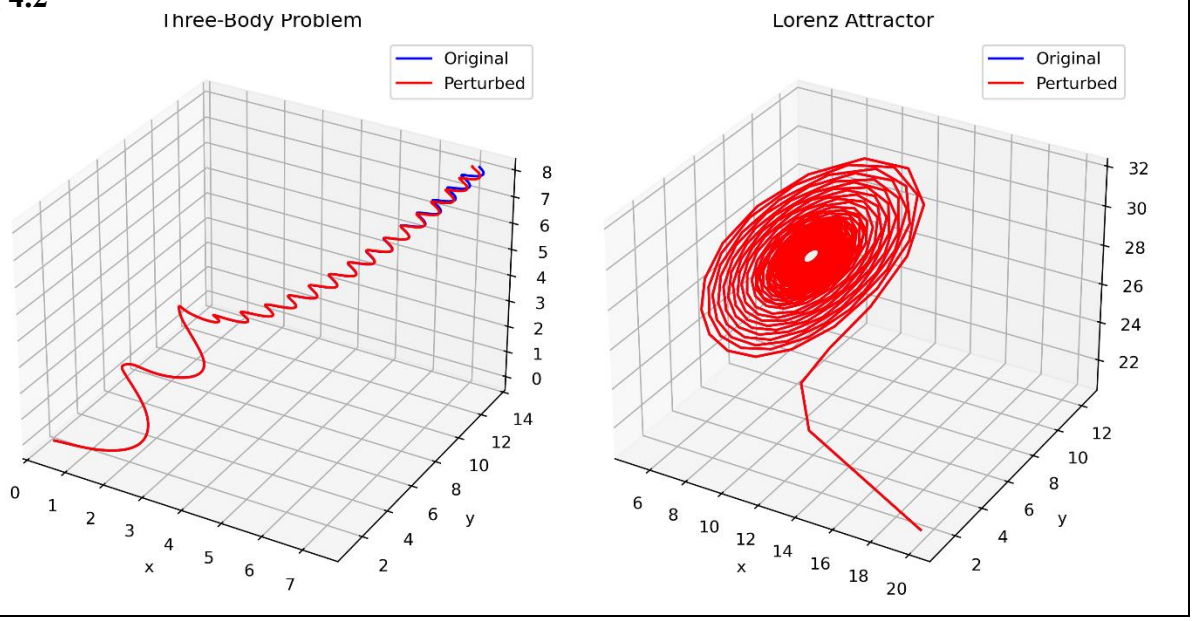


Fig 4.3

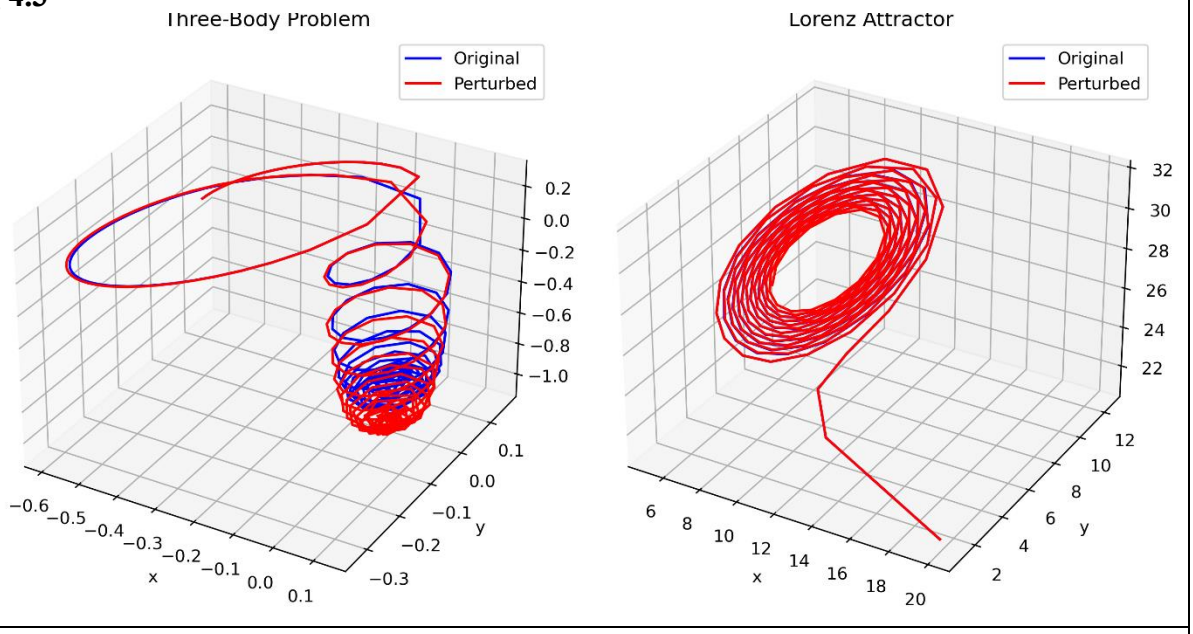
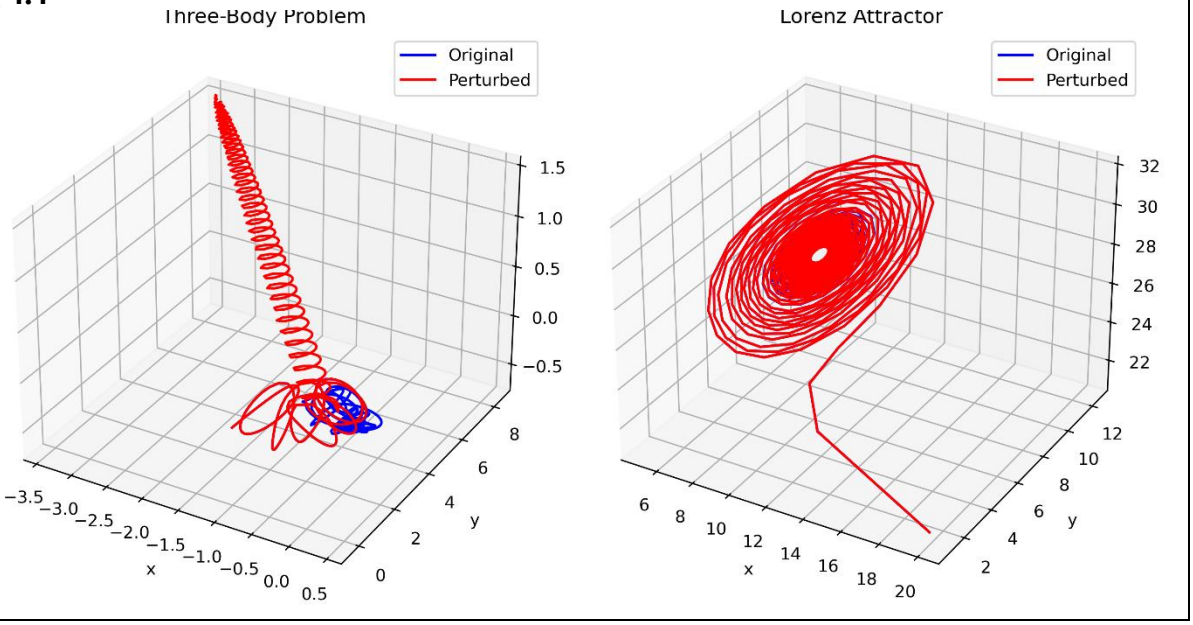


Fig 4.4

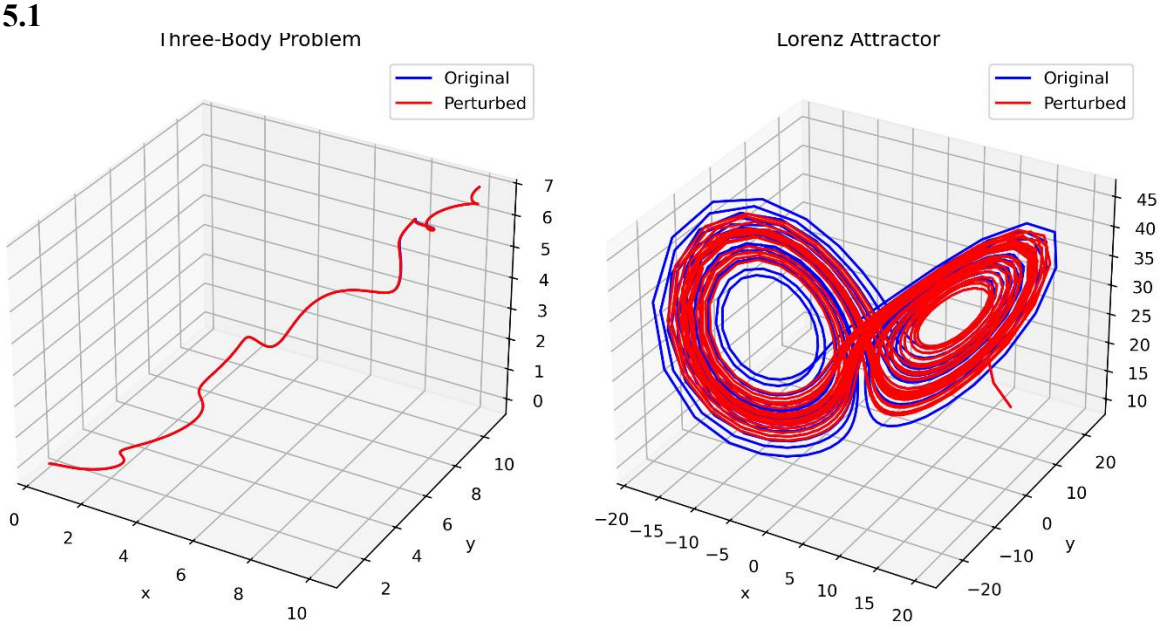


we can confirm if we increase σ or decrease p the system goes to stability.

$\sigma = 10, p = 30$

The system is unstable:

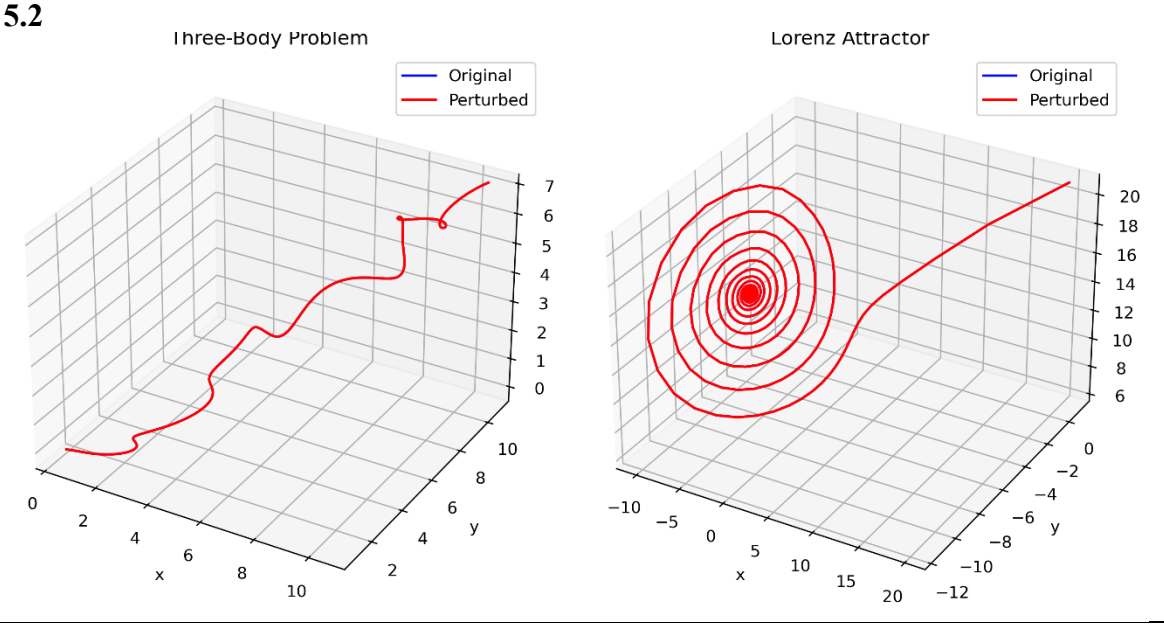
Fig 5.1



$\sigma = 10, p = 15$

The system is stable:

Fig 5.2



As mentioned, these parameters can be anything in the system that changes the system to chaotic.

Stability & instability of the system

Fig 6.1

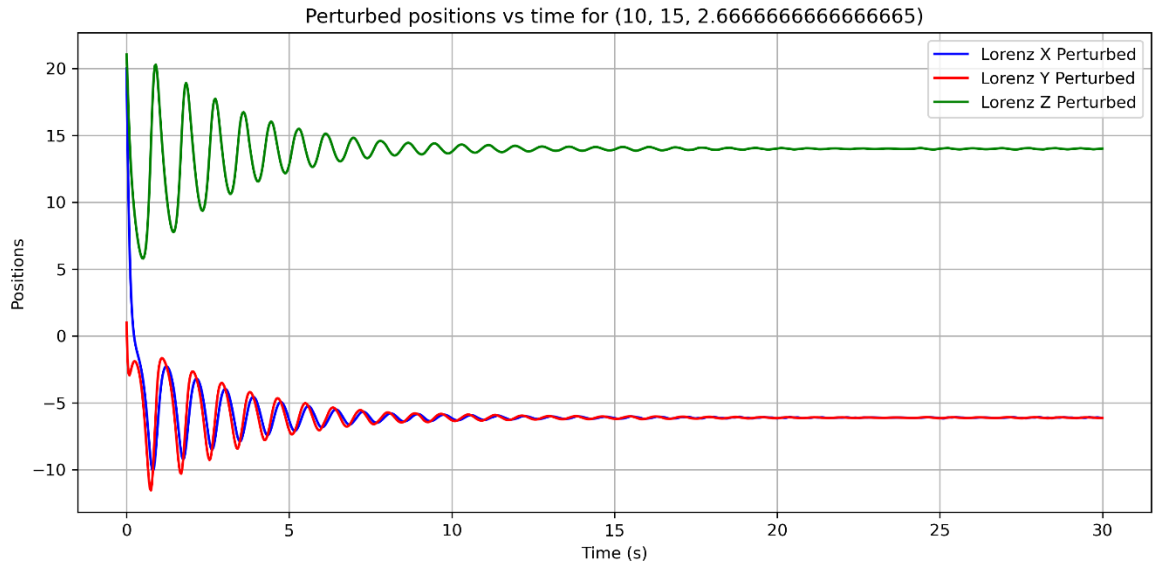


Fig 6.2

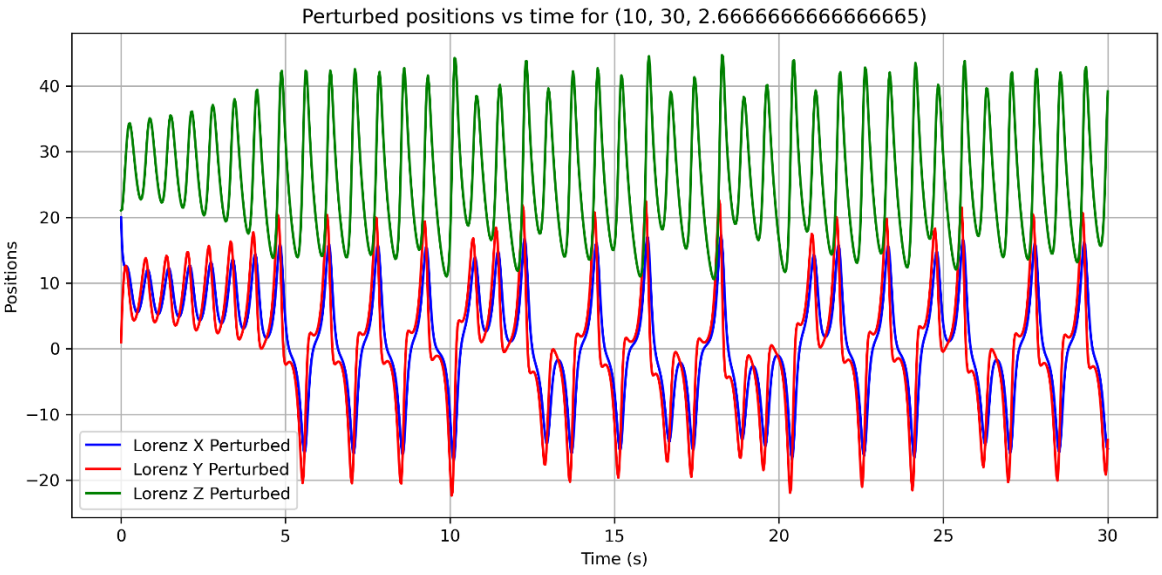


Fig 6.3

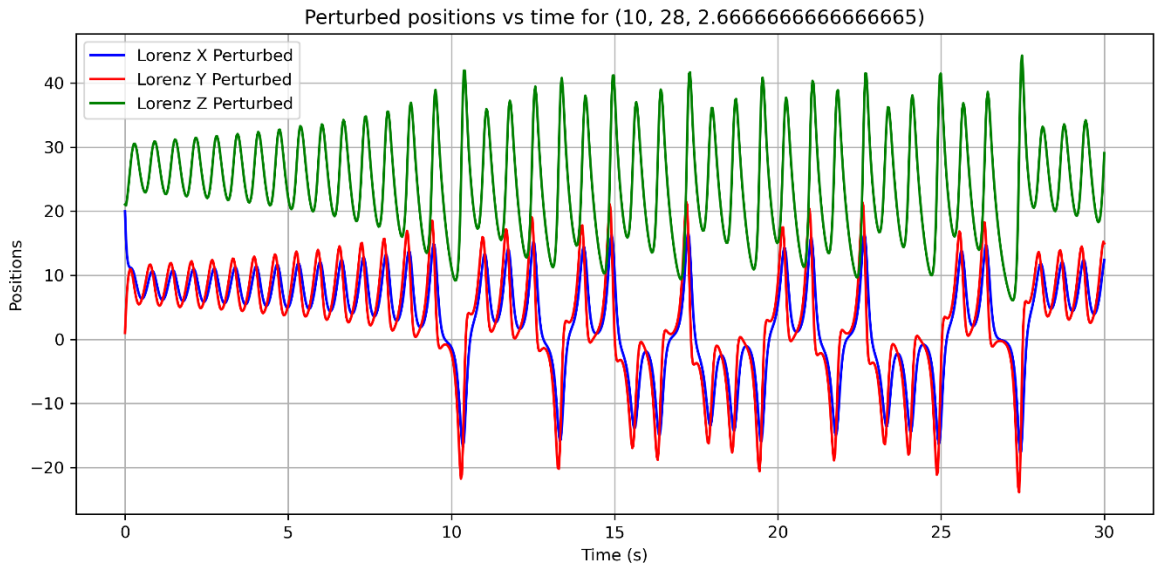
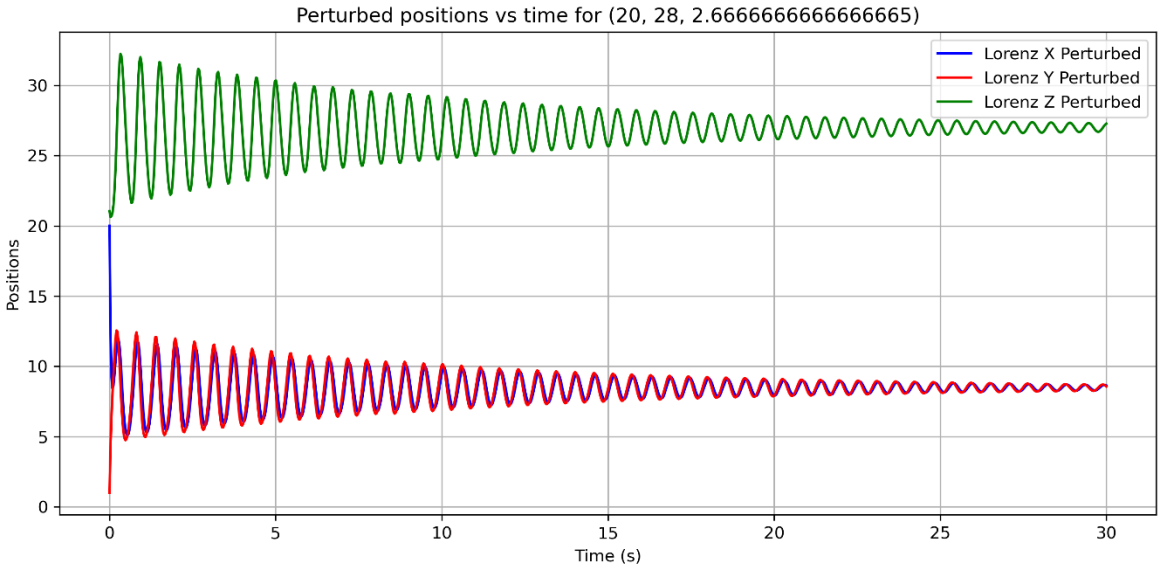

Perturbed positions vs time for (10, 28, 2.6666666666666665)
Lorenz X Perturbed
Lorenz Y Perturbed
Lorenz Z Perturbed
Positions
Time (s)

Fig 6.4


Perturbed positions vs time for (20, 28, 2.6666666666666665)
Lorenz X Perturbed
Lorenz Y Perturbed
Lorenz Z Perturbed
Positions
Time (s)

References

Relativistic Pythagorean three-body problem

Lorenz equations and atmospheric convection

(PDF) Chaos Theory And Its Application