MATH 312 Final Project Paper

Caedan Miller, Kiki Murphy, Will St. John

December 18, 2023

Contents

1 Project Overview

1.1 What is the 3-Body Problem?

The three body problem is a classic application of Newtonian physics wherein the position and velocity of three bodies at some time are determined by their initial positions and velocities, their masses, and the gravitational constant G.

1.2 Equations

Three second-order differential equations can be used to model the three body problem as described by Newtonian mechanics. These equations are as follows:

$$\ddot{r_1} = -Gm_2 \frac{r_1 - r_2}{|r_1 - r_2|^3} - Gm_3 \frac{r_1 - r_3}{|r_1 - r_3|^3} \tag{1}$$

$$\ddot{r_2} = -Gm_2 \frac{r_2 - r_1}{|r_2 - r_1|^3} - Gm_3 \frac{r_2 - r_3}{|r_2 - r_3|^3} \tag{2}$$

$$\ddot{r_3} = -Gm_2 \frac{r_3 - r_1}{|r_3 - r_1|^3} - Gm_3 \frac{r_3 - r_2}{|r_3 - r_2|^3} \tag{3}$$

where r_1, r_2, and r_3 are the positions of the three bodies, m_1, m_2, and m_3 are the masses of the three bodies, and G is the gravitational constant.

1.3 Simplification

To simplify the problem, we assume the following conditions:

- The three bodies only move in two dimensions.

- The three bodies are of equal mass $m_1 = m_2 = m_3 = 1$.

- The gravitational constant $G = 1$.

Equations (1), (2), and (3) can be rewritten as follows:

$$\ddot{r_1} = -\frac{r_1 - r_2}{|r_1 - r_2|^3} - \frac{r_1 - r_3}{|r_1 - r_3|^3} \tag{4}$$

$$\ddot{r_2} = -\frac{r_2 - r_1}{|r_2 - r_1|^3} - \frac{r_2 - r_3}{|r_2 - r_3|^3} \tag{5}$$

$$\ddot{r_3} = -\frac{r_3 - r_1}{|r_3 - r_1|^3} - \frac{r_3 - r_2}{|r_3 - r_2|^3} \tag{6}$$

1.4 Goals

Our intended goal was to model the three body problem in the 2D plane for a range of initial conditions using Heun's Method. Once every approximation is complete, we would create a "moment map" of the result by averaging the amount of times a planet appears at a particular pixel and giving that pixel a corresponding RGB value. For example, a pixel where planet 1 (red) passes through numerous times would appear very dark red in the moment map, while a pixel where planet 1 (red) and 3 (blue) pass through a lot would appear purple.The steps towards this goal fall into two categories: those that we are certain we can achieve and those that are more ambitious.

The minimum viable product (MVP) of our project entails (1) rewritten equations for the three-body problem in two dimensions, (2) code in python to compute Heun's equations for all 12 of the three-body equations, and (3) plots of three-body motion featuring all three bodies and the center of mass in Python.

2 Process

Our more ambitious goals entail the creation of the RGB map synthesizing the motion of the three bodies over a variety of initial conditions in order to achieve a better understanding of how the bodies tend to move.

2.1 Code

2.1.1 Necessary Packages

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random
```

2.1.2 Heun's Method

```
def heun(p0: np.array, N: int, t: float) -> np.array:
    """Calculatue heuns method for the three body problem for N steps with step size t.

    Args:
        p0 (np.array): 12 dimensional vector of initial conditions corresponding to the following:
            p0[0] = x1, p0[1] = x2, p0[2] = x3, p0[3] = y1, p0[4] = y2, p0[5] = y3,
            p0[6] = vx1, p0[7] = vx2, p0[8] = vx3, p0[9] = vy1, p0[10] = vy2, p0[11] = vy3
        N (int): number of steps
        t (float): step size

    Returns:
        np.array: np.array positions and velocities for each x and y of each planet
    """
    data = [p0[0], p0[1], p0[2], p0[3], p0[4], p0[5],
            p0[6], p0[7], p0[8], p0[9], p0[10], p0[11],
            derivative(p0)[6], derivative(p0)[7], derivative(p0)[8],
            derivative(p0)[9], derivative(p0)[10], derivative(p0)[11]]
    df = pd.DataFrame(data=[data], columns=['P1: X Position', 'P2: X Position',
                                'P3: X Position', 'P1: Y Position', 'P2: Y Position',
                                'P3: Y Position', 'P1: X Velocity', 'P2: X Velocity',
                                'P3: X Velocity', 'P1: Y Velocity', 'P2: Y Velocity',
                                'P3: Y Velocity', 'P1: X Acceleration', 'P2: X Acceleration',
                                'P3: X Acceleration', 'P1: Y Acceleration', 'P2: Y Acceleration',
                                'P3: Y Acceleration'])

    for i in range(0, N):
        ptemp = p0 + t * derivative(p0)
        dp0 = derivative(p0)
        dptemp = derivative(ptemp)
        p0 = p0 + t/2 * (dp0 + dptemp)
        data = [p0[0], p0[1], p0[2], p0[3], p0[4], p0[5], p0[6], p0[7], p0[8], p0[9], p0[10], p0[11]
                derivative(p0)[6], derivative(p0)[7], derivative(p0)[8], derivative(p0)[9], derivative(p0
                derivative(p0)[11]]
        df2 = pd.DataFrame(data=[data], columns=['P1: X Position', 'P2: X Position',
                                    'P3: X Position', 'P1: Y Position', 'P2: Y Position',
                                    'P3: Y Position', 'P1: X Velocity', 'P2: X Velocity',
                                    'P3: X Velocity', 'P1: Y Velocity', 'P2: Y Velocity',
                                    'P3: Y Velocity', 'P1: X Acceleration', 'P2: X Acceleration',
                                    'P3: X Acceleration', 'P1: Y Acceleration', 'P2: Y Acceleration',
                                    'P3: Y Acceleration'])
        df2['P1: X Position'] = p0[0]
        df2['P2: X Position'] = p0[1]
        df2['P3: X Position'] = p0[2]
        df2['P1: Y Position'] = p0[3]
        df2['P2: Y Position'] = p0[4]
        df2['P3: Y Position'] = p0[5]
        df2['P1: X Velocity'] = p0[6]
        df2['P2: X Velocity'] = p0[7]
        df2['P3: X Velocity'] = p0[8]
```

```python
        df2['P1: Y Velocity'] = p0[9]
        df2['P2: Y Velocity'] = p0[10]
        df2['P3: Y Velocity'] = p0[11]
        df = pd.concat([df, df2], ignore_index=True)

    df.insert(0, 'Time', [i * t for i in range(0, N + 1)])
    return df
```

2.1.3 Derivative

```python
def derivative(p0: np.array) -> np.array:
    """Calculates the derivative of the 12 dimensional vector p0.

    Args:
        p0 (np.array): 12 dimensional vecotr of initial conditions corresponding to the following:
            p0[0] = x1, p0[1] = x2, p0[2] = x3, p0[3] = y1, p0[4] = y2, p0[5] = y3,
            p0[6] = vx1, p0[7] = vx2, p0[8] = vx3, p0[9] = vy1, p0[10] = vy2, p0[11] = vy3

    Returns:
        np.array: 12 dimensional vector of the derivative of p0
    """
    p1 = 0 * p0
    p1[0] = p0[6]
    p1[1] = p0[7]
    p1[2] = p0[8]
    p1[3] = p0[9]
    p1[4] = p0[10]
    p1[5] = p0[11]
    p1[6] = - ((p0[0] - p0[1])/((p0[0]-p0[1])**2+(p0[3]-p0[4])**2)**(3/2)) - ((p0[0] - p0[2])/((p0[0]
    p1[7] = - ((p0[1] - p0[0])/((p0[1]-p0[0])**2+(p0[4]-p0[3])**2)**(3/2)) - ((p0[1] - p0[2])/((p0[1]
    p1[8] = - ((p0[2] - p0[0])/((p0[2]-p0[0])**2+(p0[5]-p0[3])**2)**(3/2)) - ((p0[2] - p0[1])/((p0[2]
    p1[9] = - ((p0[3] - p0[4])/((p0[0]-p0[1])**2+(p0[3]-p0[4])**2)**(3/2)) - ((p0[3] - p0[5])/((p0[0]
    p1[10] = - ((p0[4] - p0[3])/((p0[1]-p0[0])**2+(p0[4]-p0[3])**2)**(3/2)) - ((p0[4] - p0[5])/((p0[1
    p1[11] = - ((p0[5] - p0[3])/((p0[2]-p0[0])**2+(p0[5]-p0[3])**2)**(3/2)) - ((p0[5] - p0[4])/((p0[2
    return p1
```

2.1.4 Perterbations

```python
def perterbations(N: int) -> list:
    """Randomly generates N positions within the first two quadrants of a circle
        with radius less than or equal to 0.01.

    Args:
        N (int): number of positions to generate

    Returns:
        list: tuples of x and y positions
    """
    positions = []
    while len(positions) < N:
        x = random.randrange(-1000, 1000, 1) / 10000
        y = random.randrange(0, 1000, 1) / 10000
        if x**2 + y**2 <= 0.01:
            positions.append((x, y))
    return positions
```

2.1.5 Making Plots

```python
    def make_vis(pos: int, pert: int, title: str) -> None:
```

```python
    """Creates a visualization of the three body problem with pert perturbations. Indended
    to be used with subplots.

    Args:
        pos (int): subplot position for the visualization
        pert (int): number of perterbations to include in the plot
        title (str): title of the plot
    """
    mid_pos = perterbations(pert)

    p = np.array([-1, 1, 0, 0, 0, 0, 0, 0, 0, -1, 1, 0])
    d = heun(p, 5000, 0.001)
    plt.subplot(pos)

    plt.title(title)
    plt.plot(d['P1: X Position'][0], d['P1: Y Position'][0], c='red', marker='x', label='Earth')
    plt.plot(d['P2: X Position'][0], d['P2: Y Position'][0], c='green', marker='x',label='Mars')
    plt.plot(d['P3: X Position'][0], d['P3: Y Position'][0], c='blue', marker='x', label='Venus')
    plt.plot(d['P1: X Position'], d['P1: Y Position'], c='red', alpha=0.3, label='Earth')
    plt.plot(d['P2: X Position'], d['P2: Y Position'], c='green', alpha=0.3, label='Mars')
    plt.plot(d['P3: X Position'], d['P3: Y Position'], c='blue', alpha=0.3, label='Venus')

    for x, y in mid_pos:
        p = np.array([-1, 1, x, 0, 0, y, 0, 0, 0, -1, 1, 0])
        d = heun(p, 5000, 0.001)
        plt.plot(d['P1: X Position'], d['P1: Y Position'], c='red', alpha=0.5)
        plt.plot(d['P2: X Position'], d['P2: Y Position'], c='green', alpha=0.5)
        plt.plot(d['P3: X Position'], d['P3: Y Position'], c='blue', alpha=0.5)
    return
```

2.1.6 Making Average Plots

```python
def generate_avg(pert: int, pos: int, duration: int, stepsize: float) -> None:
    """Creates an average of pert perterbations using Heun's method for a particular duration and st

    Args:
        pert (int): number of perterbations to plot
        duration (int): how many steps to take
        stepsize (float): step size for heun's method
    """
    x1 = []
    x2 = []
    x3 = []
    y1 = []
    y2 = []
    y3 = []
    mid_pos = perterbations(pert)
    plt.subplot(pos)

    for x, y in mid_pos:
        p = np.array([-1, 1, x, 0, 0, y, 0, 0, 0, -1, 1, 0])
        d = heun(p, duration, stepsize)
        x1.append(d['P1: X Position'])
        x2.append(d['P2: X Position'])
        x3.append(d['P3: X Position'])
        y1.append(d['P1: Y Position'])
        y2.append(d['P2: Y Position'])
        y3.append(d['P3: Y Position'])
        plt.plot(d['P1: X Position'], d['P1: Y Position'], c='red', alpha=0.1)
```

```
        plt.plot(d['P2: X Position'], d['P2: Y Position'], c='green', alpha=0.1)
        plt.plot(d['P3: X Position'], d['P3: Y Position'], c='blue', alpha=0.1)

    x1avg = np.mean(x1, axis=0)
    x2avg = np.mean(x2, axis=0)
    x3avg = np.mean(x3, axis=0)
    y1avg = np.mean(y1, axis=0)
    y2avg = np.mean(y2, axis=0)
    y3avg = np.mean(y3, axis=0)

    plt.title(f"Average of {pert} Perturbations")
    plt.plot(x1avg, y1avg, c='red', alpha=1)
    plt.plot(x2avg, y2avg, c='green', alpha=1)
    plt.plot(x3avg, y3avg, c='blue', alpha=1)
    return
```

2.2 Adding Masses & G

2.3 Exploring Collinear Initial Conditions

3 Results and Discussion

[width=0.25]mesh

Figure 1: a nice plot

4 Conclusion