Computational Modeling
Project 2

Maroua Ouahbi
Constructor University

INTRODUCTION

The project simulates the motion of a planet around the sun using the Newtonian mechanics, while also exploring the impact of the time steps (Δt) on the accuracy of the simulation, in relation to energy conservation.

Since all the planets in the solar system move in the same plane, a 2D model using the x- and y-axes is sufficient for force analysis. The simulation explores the influence of the sun's gravitational force on the planets, using Newton's equations of motion:

- Newton's Law of Gravitation:

$$\overline{F}_{1,2} = -\frac{GMm(\overline{r}_2 - \overline{r}_1)}{(\overline{r}_2 - \overline{r}_1)^3} \quad (1)$$

where G is the gravitational constant ($6.67 \times 10^{-11} Nm^2 kg^{-2}$), M the mass of the sun ($2.0 \times 10^{30} kg$), m the mass of the planet and r_1 and r_2 are their respective positions.

- Newton's second law of motion:

$$\overline{F} = m\overline{a} \quad (2)$$

where $\overline{F}$ is the gravitational force acting on the planet and $\overline{a}$ is the acceleration of the planet.

Then the Euler method is used for iterating over small steps Δt to solve the equations of motion. It updates velocities using:

$$\overline{v}(t + \Delta t) = \overline{v}(t) + \overline{a}(t)\Delta t \quad (3)$$

where $\overline{v}(t)$ is the velocity of the planet at time t, and $\overline{a}(t)$ is the acceleration of the planet at time t.
And it updates the positions:

$$\overline{r}(t + \Delta t) = \overline{r}(t) + \overline{v}(t)\Delta t \quad (4)$$

where $\overline{r}(t)$ the position at time t.

In order to check for the conservation of energy, the kinetic energy (KE) is computed using the equation:

$$KE = \frac{1}{2}mv^2$$

(5)

where v is the velocity magnitude of the planet.
Then the potential energy (PE) is computed using the equation:

$$PE = -\frac{GMm}{r}$$

(6)

where M is the sun's mass, m the mass of the planet and r is the distance between the planet and the sun.
And finally the total mechanical energy is computed using the equation:

$$E = KE + PE$$

(7)

- **Constants**

```
# Constants
G = 6.67e-11  # The gravitational constant in N m^2/kg^2
M_SUN = 1.989e30  # The mass of the Sun in kg
```

All the constants used in the computations through the code are defined above; the gravitational constant as well as the mass of the sun.

- **Dictionary for planetary data**

Planetary Data:

planet	mass(kg)	radius (AU)	eccentricity
Mercury	2.4×10^{23}	0.39	0.206
Venus	4.9×10^{24}	0.72	0.007
Earth	6.0×10^{24}	1.00	0.017
Mars	6.6×10^{23}	1.52	0.093
Jupiter	1.9×10^{27}	5.20	0.048
Saturn	5.7×10^{26}	9.54	0.056
Uranus	8.8×10^{25}	19.19	0.046
Neptune	1.0×10^{26}	30.06	0.010
Pluto	1.3×10^{22}	39.26	0.248
Sun	2.0×10^{30}		

All the planetary data, given in the table above, was set in a dictionary storing the mass, average distance from the Sun, and eccentricity for each planet. In order to retrieve it when needed for the calculations.

```
planet_data = {
    "Mercury": {"mass": 2.4e23, "radius_au": 0.39, "eccentricity": 0.206},
    "Venus": {"mass": 4.9e24, "radius_au": 0.72, "eccentricity": 0.007},
    "Earth": {"mass": 6.0e24, "radius_au": 1.0, "eccentricity": 0.017},
    "Mars": {"mass": 6.6e23, "radius_au": 1.52, "eccentricity": 0.093},
    "Jupiter": {"mass": 1.9e27, "radius_au": 5.2, "eccentricity": 0.048},
    "Saturn": {"mass": 5.7e26, "radius_au": 9.54, "eccentricity": 0.056},
    "Uranus": {"mass": 8.8e25, "radius_au": 19.19, "eccentricity": 0.046},
    "Neptune": {"mass": 1.0e26, "radius_au": 30.06, "eccentricity": 0.010},
    "Pluto": {"mass": 1.3e22, "radius_au": 39.26, "eccentricity": 0.248},
}
```

Then in order to retrieve the info about the planet in needed for other parts of the code this function was defined, which calls out the planetary data for the planet imputed by the user:

```
def get_planet_data(planet_name):
    return planet_data.get(planet_name)
```

- **Initialization of conditions**

```
def initialize_conditions(m_planet, r_au, eccentricity, dt, t_max):
    r_planet = r_au * 1.496e11  # Convert AU to meters.
    v_circular = np.sqrt(G * M_SUN / r_planet)
    v_planet = v_circular * (1 + eccentricity)
    num_steps = int(t_max / dt)

    positions = np.zeros((num_steps, 2))
    velocities = np.zeros((num_steps, 2))

    positions[0] = [r_planet, 0]  # Initial position in x-direction, with y as zero.
    velocities[0] = [0, v_planet]  # Initial velocity is in the y-direction.

    return positions, velocities, num_steps
```

The initial conditions for the planet's position and velocity are set. The distance from the sun is converted from AU to meters, and the initial velocity is calculated using Newton's Law of Gravitation (equation 1) and adjusted for orbital eccentricity. The arrays for storing the positions and velocities of the planet at each time step are initialized. The starting position is along the x-axis, and the velocity is along the y-axis.

- **Gravitational Force Calculations**

```
def compute_gravitational_force(planet_pos):
    r = np.linalg.norm(planet_pos)  # Distance from the sun.
    return -G * M_SUN * planet_pos / r ** 3  # Gravitational force vector.
```

This function calculates the gravitational force using (equation 1) by determining the distance between the planet and the Sun.

- **Energy Calculations**

```
def energy_calc(positions, velocities, dt, num_steps, m_planet):

    kinetic_energy = np.zeros(num_steps)
    potential_energy = np.zeros(num_steps)
    total_energy = np.zeros(num_steps)
```

Euler's method is applied here to update the planet's velocity (equation 3) and position (equation 4) at each time step, based on the gravitational force. Then kinetic energy (equation 5), potential energy (equation 6), and total mechanical energy (equation 7) are calculated at each step to assess energy conservation.

Then the energy change was computed using the defined function below:

```
def energy_change_calc(total_energy):

    initial_energy = total_energy[0]  # Initial energy.
    energy_change = (total_energy - initial_energy) / np.abs(initial_energy)  # Relative energy change
    return energy_change
```

This function computes the relative change in energy to assess how well energy is conserved during the simulation. Ideally, this change should be minimal for smaller time steps.

- **Euler Method Integration**

```
if planet_info:
    m_planet = planet_info["mass"]  # Mass in kg.
    r_au = planet_info["radius_au"]  # Radius in AU.
    eccentricity = planet_info["eccentricity"]  # Eccentricity of the orbit.
    t_max = 365 * 24 * 3600
    dt_values = [60 * 60 * 24 * i for i in range(1, 31)]  # Time steps from 1 day to 30 days.
    energy_changes = []

    for dt in dt_values:
        positions, velocities, num_steps = initialize_conditions(m_planet, r_au, eccentricity, dt, t_max)

        _, _, total_energy = energy_calc(positions, velocities, dt, num_steps, m_planet)

        energy_change = energy_change_calc(total_energy)
        energy_changes.append(energy_change[-1])  # Final energy change.
```

The user is prompted to input the name of a planet, and its corresponding data (mass, radius, eccentricity) is retrieved. The simulation is run for different time steps (from 1 day to 30 days) to observe the effect of time step size on energy conservation.

- **Plotting Results**

```
    # Plotting Results
    plt.figure(figsize=(12, 6))
    plt.plot( *args: dt_values, energy_changes, marker='o', label=f'Energy Change for {planet_name}')

    plt.xscale('log')
    plt.title("Energy Change Over One Orbit vs Time Step (Δt)")
    plt.xlabel("Time Step (s)")
    plt.ylabel("Relative Energy Change")
    plt.axhline( y: 0, color='black', linewidth=0.5, linestyle='--')
    plt.legend()
    plt.grid()
    plt.show()
```

A plot of the relative energy change against the time step size is generated, with a logarithmic scale for the x-axis. This graph provides insight into the accuracy of energy conservation as the time step size varies.

- **Error Handling**

```
else:
    print("Planet not found. Please check the name and try again.")
```

This line acts as a safety net, making sure users get a response even if they enter an invalid planet name. It enhances the program's reliability and makes it easier for users to navigate.

RESULTS & ANALYSIS

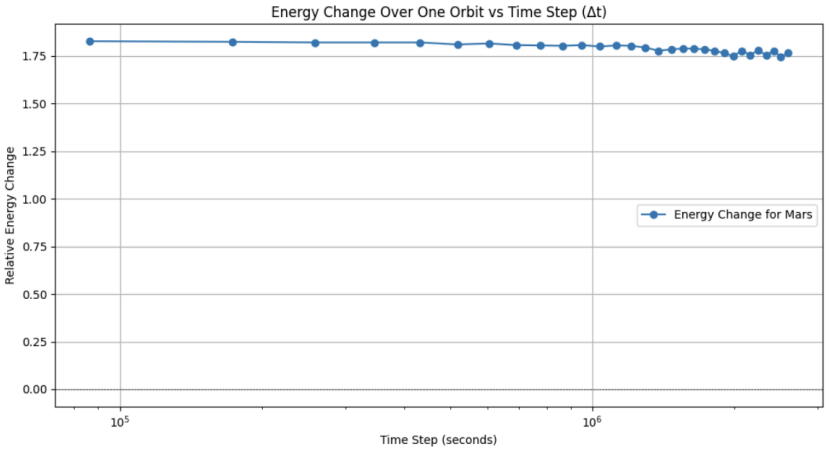

Figure 1: *Plot of energy change over* Δt *for Mars*

The Mars energy change plot shows that relative energy deviation increases with larger time step sizes (Δt). Smaller time steps, particularly around 1 day, improve energy conservation and numerical stability in the simulation, highlighting the importance of selecting appropriate time steps in orbital dynamics.

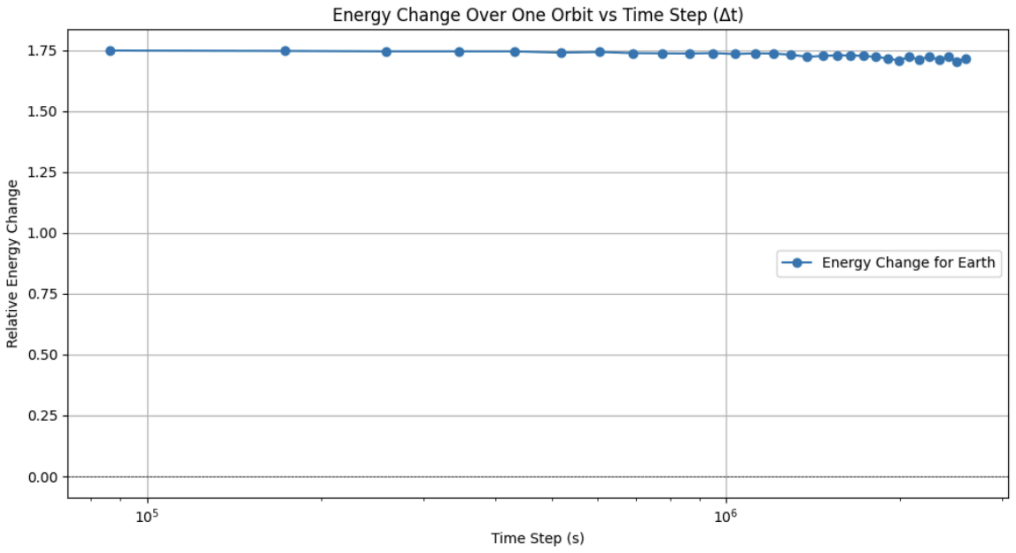

Figure 2: *Plot of energy change over* Δt *for Earth*

The Earth energy change plot reflects a similar trend to Mars, with less energy deviation at smaller time steps. Compared to Mars, Earth may demonstrate slightly different sensitivity to time steps, emphasizing the need for careful selection to balance energy conservation and computational efficiency, particularly around 1-2 days.

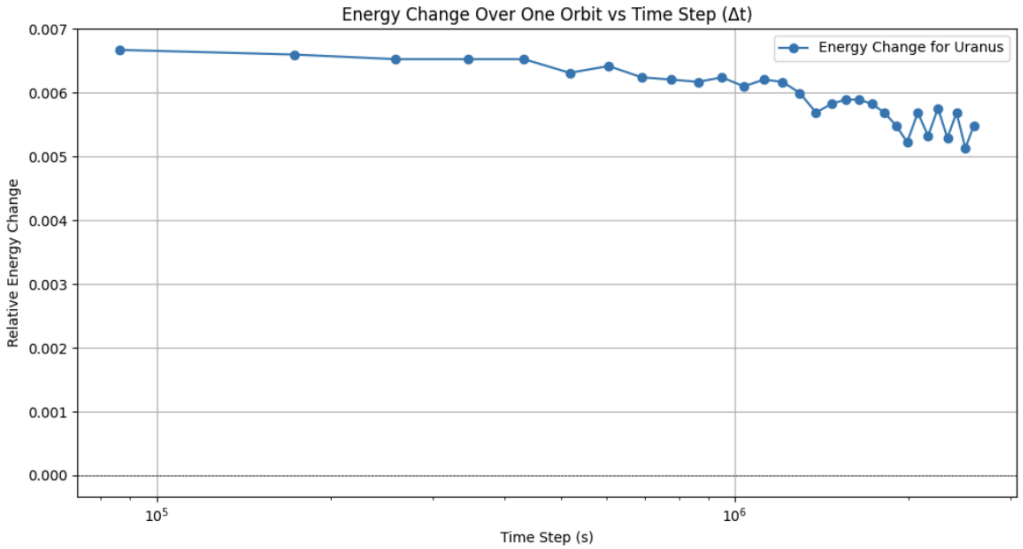

Figure 3: *Plot of energy change over* Δt *for Uranus*

The Uranus energy change plot indicates more significant energy deviations with larger time steps, suggesting that Uranus's distance and longer orbital period make it more sensitive to numerical errors. Similar to Mars and Earth, smaller time steps, ideally close to 1 day, are essential for maintaining energy conservation in simulations of outer planets.

CONCLUSION

In this project, we simulated how planets move around the Sun using Newtonian mechanics, and we found that choosing the right time step is crucial for keeping the energy stable and the calculations accurate. When we analyzed the energy changes for Mars, Earth, and Uranus, we noticed a clear trend: larger time steps resulted in bigger energy deviations, while smaller time steps improved energy conservation.

Using the Euler method, we discovered that keeping time steps to about one day or less is particularly important for planets with longer orbits or those that are farther from the Sun. This highlights how essential it is to carefully calibrate time steps in simulations of orbital dynamics to get reliable results.

Overall, this project offers valuable insights into the computational modeling of celestial mechanics, emphasizing the importance of precision in numerical methods and their effects on our understanding of how planets behave. Looking ahead, future work could involve using more advanced techniques, like the Runge-Kutta methods, to enhance energy conservation and accuracy in our simulations.