Solving the Three-Body Problem Using Python

The three-body problem is a classical problem in physics and mathematics that involves predicting the motion of three bodies under their mutual gravitational influence. Since there is no general closed-form solution, numerical simulations are the most practical approach for solving it. Python, with libraries like numpy, matplotlib, and scipy, provides an accessible and powerful platform for such simulations.

1. Setting Up Your Environment

To get started, ensure you have Python installed, along with the required libraries. You can install the dependencies using the following command:

```
pip install numpy matplotlib scipy
```

2. Defining the Physics of the Problem

The core of the three-body problem is Newton's law of gravitation:

F = G * (m1 * m2) / r^2

Where:
- F is the gravitational force between two bodies,
- G is the gravitational constant,
- m1, m2 are the masses of the bodies,
- r is the distance between them.

We start by implementing this in Python:

```
import numpy as np

def gravitational_force(m1, m2, r1, r2):
    G = 6.67430e-11  # Gravitational constant (m^3 kg^-1 s^-2)
    r = np.linalg.norm(r2 - r1)
    force = G * m1 * m2 / r**2
    direction = (r2 - r1) / r
    return force * direction
```

3. Modeling the Equations of Motion

The motion of the three bodies is governed by their mutual gravitational forces. We define a function to compute the derivatives of position and velocity, which will be used for numerical integration.

```
def three_body_equations(t, state, masses):
    # Extract positions and velocities
    x1, y1, x2, y2, x3, y3, vx1, vy1, vx2, vy2, vx3, vy3 = state
    r1 = np.array([x1, y1])
```

```
    r2 = np.array([x2, y2])
    r3 = np.array([x3, y3])

    # Masses
    m1, m2, m3 = masses

    # Compute forces
    F12 = gravitational_force(m1, m2, r1, r2)
    F13 = gravitational_force(m1, m3, r1, r3)
    F23 = gravitational_force(m2, m3, r2, r3)

    # Compute accelerations
    a1 = (F12 + F13) / m1
    a2 = (-F12 + F23) / m2
    a3 = (-F13 - F23) / m3

    # Derivatives of position = velocity
    dxdt = [vx1, vy1, vx2, vy2, vx3, vy3]
    dvdt = [a1[0], a1[1], a2[0], a2[1], a3[0], a3[1]]

    return dxdt + dvdt
```

4. Setting Initial Conditions

We need to specify the masses, positions, and velocities of the three bodies at the start of the simulation.

```
# Masses (in kilograms)
masses = [1.989e30, 5.972e24, 7.348e22]  # Sun, Earth, Moon

# Initial positions (x, y) in meters
positions = [
    [0, 0],  # Sun
    [1.496e11, 0],  # Earth
    [1.496e11 + 3.844e8, 0]  # Moon
]

# Initial velocities (vx, vy) in meters/second
velocities = [
    [0, 0],  # Sun
    [0, 29783],  # Earth
    [0, 29783 + 1022]  # Moon
]

# Combine into a single state vector
```

```
initial_state = []
for i in range(3):
    initial_state.extend(positions[i])
    initial_state.extend(velocities[i])
```

5. Solving the System of Differential Equations

Using scipy.integrate.solve_ivp, we solve the equations of motion for a given time span.

```
from scipy.integrate import solve_ivp

# Time span for the simulation (one year, in seconds)
t_span = (0, 3.154e7)
t_eval = np.linspace(0, 3.154e7, 1000)  # Evaluate at 1000 time points

# Solve the equations
solution = solve_ivp(
    three_body_equations,
    t_span,
    initial_state,
    t_eval=t_eval,
    args=(masses,),
    method='RK45'
)
```

6. Visualizing the Results

Extract the positions of the three bodies from the solution and plot their trajectories.

```
import matplotlib.pyplot as plt

# Extract positions
x1, y1 = solution.y[0], solution.y[1]
x2, y2 = solution.y[2], solution.y[3]
x3, y3 = solution.y[4], solution.y[5]

# Plot the trajectories
plt.figure(figsize=(10, 8))
plt.plot(x1, y1, label='Body 1 (Sun)', lw=2)
plt.plot(x2, y2, label='Body 2 (Earth)', lw=2)
plt.plot(x3, y3, label='Body 3 (Moon)', lw=2)

# Add labels and legend
plt.xlabel('X Position (m)')
plt.ylabel('Y Position (m)')
plt.title('Three-Body Problem Simulation')
```

```
plt.legend()
plt.axis('equal')
plt.grid()
plt.show()
```

7. Extending the Simulation

Here are some ideas to extend the simulation:
- Explore different initial conditions to observe how the system behaves under various setups.
- Simulate different three-body systems, such as binary stars with a planet.
- Verify that the total energy (kinetic + potential) remains constant over time to validate the simulation's accuracy.
- Use smaller time steps in t_eval for more accurate solutions.

8. Conclusion

By combining Python's scientific libraries, we can effectively solve and visualize the three-body problem. This simulation showcases the beauty of gravitational dynamics and highlights the chaotic nature of three-body systems. With small tweaks, you can simulate various configurations, explore periodic orbits, and gain deeper insights into celestial mechanics.