Simulación planetaria en Python

Universidad Nacional de La Pampa
Licenciatura en Física
Física Computacional

Tisiano Martinez
Julio 2025

Resumen

En este trabajo se desarrolló una simulación del movimiento del sistema solar utilizando la ley de gravitación de Newton y el método numérico de Verlet para integrar las ecuaciones del movimiento. El objetivo principal es ilustrar las trayectorias orbitales de los planetas en torno al Sol, considerando la interacción gravitatoria entre los cuerpos celestes. La simulación se basa en una versión adaptada de la constante de gravitación universal en unidades astronómicas y días, lo que permite trabajar con escalas naturales del sistema solar. Se modelan seis cuerpos (el Sol, Mercurio, Venus, Tierra, Marte y Júpiter), cuyas posiciones y velocidades iniciales se especifican según datos astronómicos aproximados.

Palabras claves: Gravitación**,** Sistema solar, Python, Orbitas planetarias, Método de Verlet.

*In*troducción

El estudio del movimiento de los cuerpos celestes ha sido una de las motivaciones históricas más importantes para el desarrollo de la física. Desde las leyes de Kepler hasta la formulación de la gravitación universal por Newton, la comprensión del sistema solar ha requerido herramientas tanto teóricas como empíricas. En la actualidad, la programación y la simulación computacional permiten abordar estos fenómenos desde un enfoque dinámico y visual, facilitando su análisis y comprensión.

Este trabajo se centra en modelar un sistema simplificado del sistema solar, utilizando la integración por el método de Verlet, apropiado para sistemas conservativos como éste.

Desarrollo

1. Problema físico

   El sistema solar está compuesto por una serie de cuerpos celestes que interactúan gravitacionalmente. El problema físico abordado consiste en modelar y simular el movimiento de varios planetas

alrededor del Sol, considerando la interacción gravitatoria mutua entre ellos. Se asume que los cuerpos obedecen la ley de gravitación universal de Newton, y que su movimiento se rige por la segunda ley de Newton en un sistema bidimensional simplificado.

2. Ecuaciones del movimiento

Las fuerzas que actúan sobre cada planeta están determinadas por la ley de gravitación universal:

$$\overrightarrow{F_{ij}} = -G\frac{m_i \cdot m_j}{r_{ij}^2} \cdot \hat{r}_{ij}$$

Segunda ley de Newton, relaciona la fuerza con el cambio en el movimiento de un cuerpo:

$$\overrightarrow{F_{ij}} = m\vec{a}_i \quad \rightarrow \quad \overrightarrow{a_i} = \frac{\overrightarrow{F_{ij}}}{m}$$

Para obtener la aceleración que experimenta un planeta debido a la gravedad, se iguala la segunda ley de Newton con la fuerza gravitatoria:

$$\vec{a}_i = -G\sum_{i\neq j}\frac{m_j\left(\overrightarrow{r_i} - \overrightarrow{r_j}\right)}{\left|\overrightarrow{r_i} - \overrightarrow{r_j}\right|^3}$$

Este resultado muestra que la aceleración sobre un cuerpo depende solo de la masa del cuerpo que lo atrae, la distancia entre ellos, y la constante $\vec{G}$.

Para resolver numéricamente la evolución temporal, se aplicó el método de Verlet, que actualiza las posiciones con la siguiente fórmula:

$$\overrightarrow{r_{n+1}} = 2\overrightarrow{r_n} - \overrightarrow{r_{n-1}} + \vec{a}_n \cdot \Delta t^2$$

Este método es particularmente adecuado para simular sistemas conservativos, como el sistema solar, por las siguientes razones:

- **Conserva mejor la energía total** del sistema a lo largo del tiempo, evitando errores acumulativos que podrían hacer que los planetas se alejen o colapsen artificialmente.
- **Es estable** para sistemas en los que las aceleraciones cambian lentamente.
- **Es reversible en el tiempo**, lo que refleja una propiedad fundamental de las leyes físicas en sistemas sin disipación.
- **No requiere cálculo directo de velocidades** en cada paso, lo que reduce el error numérico.

3. Programación del modelo

   El modelo fue implementado en Python mediante la creación de una clase cuerpo, que representa a cada cuerpo celeste con atributos de posición, velocidad, masa y listas para registrar su trayectoria histórica. Los métodos definidos permiten calcular las aceleraciones (por interacción gravitatoria con otros cuerpos) y actualizar las posiciones usando el método de Verlet.

   Se simulan seis cuerpos: el Sol, Mercurio, Venus, Tierra, Marte y Júpiter. Las condiciones iniciales (posición y velocidad) fueron establecidas en unidades astronómicas (UA) para la distancia, masas relativas al Sol y días terrestres como unidad de tiempo.

4. Lenguajes y librerías utilizadas

   El lenguaje elegido fue Python y las bibliotecas utilizadas fueron:

   - **numpy:** permite hacer operaciones matemáticas rápidas y vectorizadas.
   - **matplotlib.pyplot:** genera los gráficos estáticos que muestran la evolución de las órbitas y los planetas cuadro a cuadro.
   - **matplotlib.animation:** convierte las gráficas individuales en una secuencia animada que muestra el movimiento orbital en el tiempo.

5. Animación y visualización

   Para visualizar el resultado de la simulación se empleó FuncAnimation, que permite representar en tiempo real la evolución del sistema. En cada fotograma se muestran las posiciones actuales de los planetas y su trayectoria acumulada.

Cada planeta se representa con un color y tamaño distintivo, y el Sol se ubica en el centro como cuerpo dominante. La animación abarca un período de dos años terrestres, con un paso temporal de un día, lo que permite visualizar varias órbitas de los planetas interiores y parte de la órbita de Júpiter.

6. Bibliografía

https://numpy.org/

https://matplotlib.org/stable/

Resnick, R., Halliday, D., & Krane, K. (2004). *Física (Vol. 1, 5ª ed.)*. Editorial Patria.

https://es.wikipedia.org/wiki/Ley_de_gravitaci%C3%B3n_universal

https://en.wikipedia.org/wiki/Verlet_integration

https://youtu.be/d5R8odwgceM?si=WFDCeFz9k9ZQcpLP

7. Anexo

```python
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np

G = (6.67 * (10 ** -11) * (6.68 * (10 ** -12))**3) / (5 * (10 ** -31) * (1.15 * (10 ** -5))**2)
dt = 1          #Salto temporal de un dia
N_A = 2         #Numero de años terrestres
N = N_A*365     #Numero de iteraciones

class cuerpo:
    """Clase para los planetas: el objeto creado es un planeta"""

    def __init__(self, x, y, vx, vy, masa):
        #Atributos del objeto
        self.x=x                        #Coordenada x inicial del planeta
        self.y=y                        #Coordenada y inicial del planeta
        self.masa=masa                  #Masa del planeta
        self.orbitax=[x, x+vx*dt]       #Orbita en x del planeta
        self.orbitay=[y, y+vy*dt]       #Orbita en y del planeta
        self.vx=vx                      #Velocidad inicial en x del planeta
        self.vy=vy                      #Velocidad inicial en y del planeta
```

```
    #Metodos
    def atrac(self, ocuerpo):

        """Calcula la atraccion hacia otro cuerpo."""

        r = np.sqrt((self.x-ocuerpo.x)**2+(self.y-ocuerpo.y)**2)
        ax = -G*ocuerpo.masa*(self.x-ocuerpo.x)/r**3
        ay = -G*ocuerpo.masa*(self.y-ocuerpo.y)/r**3
        a = np.array([ax, ay])
        return a

    def posicion(self, Cuerpos):

        """Calcula la nueva posicion de un cuerpo usando Verlet."""

        #Calculo de la aceleracion total
        ax = ay = 0
        for cuerpo in Cuerpos:
            if self == cuerpo:
                continue #No calculamos la atraccion de un planeta a si mismo
            a = self.atrac(cuerpo)
            ax += a[0]
            ay += a[1]
        # Verlet
        self.x=2*self.orbitax[-1]-self.orbitax[-2]+ax*dt**2
        self.y=2*self.orbitay[-1]-self.orbitay[-2]+ay*dt**2
        self.orbitax.append(self.x)
        self.orbitay.append(self.y)

#Instanciamos cuerpos
Sol = cuerpo(x=0, y=0, vx=-0.000, vy=0.000, masa=1)

Tierra = cuerpo(x=1, y=0, vx=0, vy=0.01724, masa=3.00273*10**-6)

Marte = cuerpo(x=1.524, y=0, vx=0, vy=0.013929, masa=3.22700*10**-7)

Venus = cuerpo(x=0.723, y=0, vx=0, vy=0.020248, masa=2.4470*10**-6)

Mercurio = cuerpo(x=0.389, y=0, vx=0, vy=0.02956, masa=1.65158*10**-7)

Jupiter = cuerpo(x=5.20, y=0, vx=0, vy=0.0075611, masa=0.00095525)

#Lista de cuerpos celestes
Cuerpos = [Sol, Tierra, Mercurio, Venus, Marte, Jupiter]

#Dinamica de movimiento
for t in range(N):
    for planeta in Cuerpos:
        planeta.posicion(Cuerpos)
```

```
#Animacion

#Coordenadas de movimiento
xMe = np.array(Mercurio.orbitax)
yMe = np.array(Mercurio.orbitay)

xVe = np.array(Venus.orbitax)
yVe = np.array(Venus.orbitay)

xTi = np.array(Tierra.orbitax)
yTi = np.array(Tierra.orbitay)

xMa = np.array(Marte.orbitax)
yMa = np.array(Marte.orbitay)

xJu = np.array(Jupiter.orbitax)
yJu = np.array(Jupiter.orbitay)

xS = np.array(Sol.orbitax)
yS = np.array(Sol.orbitay)

#Grafica
fig = plt.figure()  #Figura vacia
ax = fig.gca()      #Ejes sobre los que se va a dibujar

#Actualizar
def actualizar(i):
    ax.clear()

    #Movimiento de la Tierra
    ax.plot(xTi[:i], yTi[:i], color="#0000FF")
    ax.plot(xTi[i], yTi[i], 'o', markersize=12, color="#00008B")

    #Movimiento de Mercurio
    ax.plot(xMe[:i], yMe[:i], color="#EE2C2C")
    ax.plot(xMe[i], yMe[i], 'o', markersize=6, color="#FF3030")

    #Movimiento de Venus
    ax.plot(xVe[:i], yVe[:i], color="#EE3A8C")
    ax.plot(xVe[i], yVe[i], 'o', markersize=8, color="#FF3E96")

    #Movimiento de Marte
    ax.plot(xMa[:i], yMa[:i], color="#CD3333")
    ax.plot(xMa[i], yMa[i], 'o', markersize=10, color="#8B2323")

    #Movimiento de Jupiter
    ax.plot(xJu[:i], yJu[:i], color="#CD3333")
    ax.plot(xJu[i], yJu[i], 'o', markersize=14, color="#8B2323")

    #Movimiento del Sol
    ax.plot(xS[:i], yS[:i], color='y')
    ax.plot(xS[i], yS[i], 'o', markersize=20, color='y')

    plt.xlim(-1.6, 1.6)  #Limites del eje x
    plt.ylim(-1.6, 1.6)  #Limites del eje y
    plt.grid()           #Grilla
    plt.title('Movimiento planetario: '+str(N_A)+"años terrestres", fontweight="bold")
    plt.xlabel("x", fontweight="bold", fontsize=10)
    plt.ylabel("y", fontweight="bold", fontsize=10)
    #ax.set_aspect('equal', adjustable='box')

ani=animation.FuncAnimation(fig, actualizar, range(0, N, 20), repeat=False)
#ani.save('Video.mp4)
plt.show()
```