Informe Grupal - Primer Parcial

Nota

Estudiante	Escuela	Asignatura
Jorge Enrique Condorios Yllapuma José Luis Cusilayme García Geraldine Marjorie Umasi Coaguila Carlo Joaquín Valdivia Luna	Escuela Profesional de Ingeniería de Sistemas Semestre: VII	Física Computacional

Correo Estudiantes	CUI's	Fecha Entrega
jcondorios@unsa.edu.pe jcusilaymeg@unsa.edu.pe gumasi@unsa.edu.pe cvaldivialu@unsa.edu.pe	20222076 20220598 20220574 20220567	09 de Junio

1. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 10 64 bits
- Python 3.13.4 con bibliotecas científicas:
  - NumPy 1.24.3 para computación numérica
  - SciPy 1.10.1 para algoritmos científicos
  - Matplotlib 3.7.1 para visualización
- Git 2.39.2 para control de versiones.
- Cuenta en GitHub con correo institucional.
- Conceptos de física clásica: cinemática, dinámica, gravitación.
- Métodos numéricos: Euler, análisis de error, convergencia.
- Teoría de sistemas dinámicos y mecánica celestial.

2. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/TrustMek7/FC`

3. Ejercicios Propuestos

3.1. Problema 1: Movimiento lineal con aceleración constante

3.1.1. Planteamiento del problema

La posición de una partícula en función del tiempo está descrita por la ecuación cinemática fundamental (Serway & Jewett, 2018):

$$x(t) = x_0 + v_0 t + \frac{1}{2} a t^2 \tag{1}$$

Condiciones iniciales:

$$x_0 = -2{,}0 \text{ m} \tag{2}$$
$$v_0 = 0{,}5 \text{ m/s} \tag{3}$$
$$a = 2{,}0 \text{ m/s}^2 \tag{4}$$
$$t \in [0, 10] \text{ s} \tag{5}$$

3.1.2. Solución analítica detallada

Sustituyendo los valores dados en la ecuación 1:

$$x(t) = -2{,}0 + 0{,}5t + \frac{1}{2}(2{,}0)t^2 = -2{,}0 + 0{,}5t + t^2 \tag{6}$$

La velocidad se obtiene derivando la posición:

$$v(t) = \frac{dx}{dt} = 0{,}5 + 2t \tag{7}$$

Valores específicos en puntos clave:

Tabla 1: Solución analítica para puntos específicos

Tiempo (s)	Posición (m)	Velocidad (m/s)	Aceleración (m/s^2)
0	-2.000	0.5	2.0
1	-0.500	2.5	2.0
5	22.500	10.5	2.0
10	98.500	20.5	2.0

Listing 1: Implementación de la solución analítica

```python
# Importar las bibliotecas necesarias
import matplotlib.pyplot as plt
import numpy as np

# Parmetros iniciales
h = 0.01   # Paso de tiempo
tfin = 10  # Tiempo final
x = -2 # Posicion inicial
vx = 0.5   # Velocidad inicial
ax = 2 # Aceleracion constante

# Listas para almacenar resultados
px = [x]
pv = [vx]
pt = [0]
pa = [ax]

for t in np.arange(0, tfin, h):
    # Actualizar velocidad
    vx += ax * h

    # Actualizar posicion
    x += vx * h

    px.append(x)
    pv.append(vx)
    pa.append(ax)
    pt.append(t)

# Graficar en 2D
plt.figure(figsize=(12, 8))

# Aceleracin vs Tiempo
plt.subplot(2, 2, 1)
plt.plot(pt, pa, color='brown')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Aceleracin (m/ s )')
```

```
plt.title('Aceleracin vs Tiempo')

# Velocidad vs Tiempo
plt.subplot(2, 2, 2)
plt.plot(pt, pv, color='orange')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Velocidad (m/s)')
plt.title('Velocidad vs Tiempo')

#  Posicin  vs Tiempo
plt.subplot(2, 2, 3)
plt.plot(pt, px, color='green')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Posicin (m)')
plt.title('Posicin vs Tiempo')

# Velocidad vs Posicin
plt.subplot(2, 2, 4)
plt.plot(px, pv, color='blue')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Posicin (m)')
plt.ylabel('Velocidad (m/s)')
plt.title('Velocidad vs Posicin')

plt.suptitle(f"Movimiento con Velocidad Constante (a = {ax})", fontsize=16)
plt.tight_layout(h_pad=0.5, w_pad=0.5)
plt.show()
```

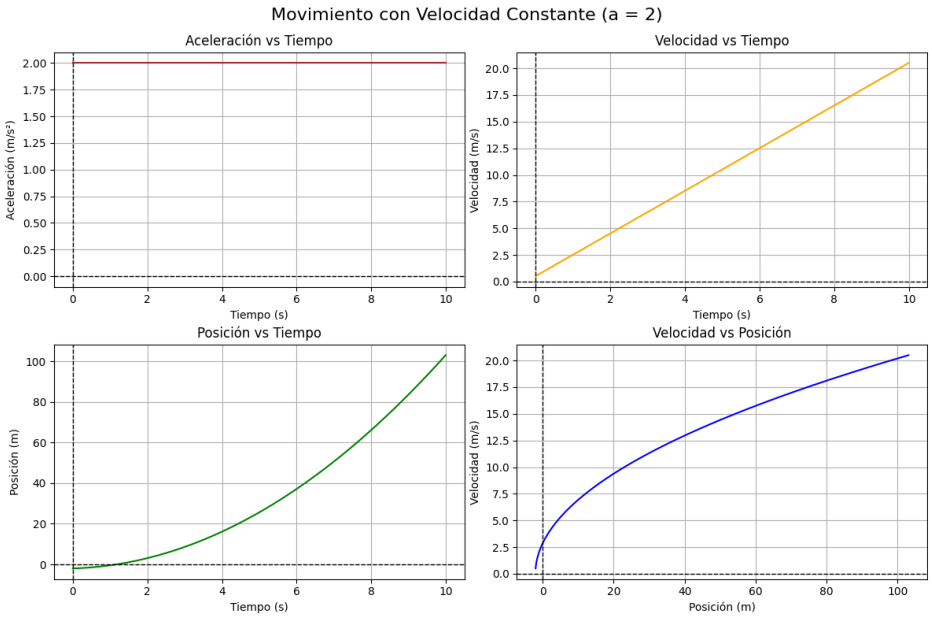

Figura 1: Movimiento con Velocidad Constante a = 2

3.1.3. Implementación del método de Euler

Para resolver numéricamente el sistema de EDOs:

$$\frac{dx}{dt} = v \tag{8}$$

$$\frac{dv}{dt} = a = 2{,}0 \text{ m/s}^2 \tag{9}$$

Aplicamos el esquema de Euler:

$$x_{n+1} = x_n + h \cdot v_n \tag{10}$$

$$v_{n+1} = v_n + h \cdot a \tag{11}$$

Listing 2: Implementación detallada del método de Euler

```
# Importar las bibliotecas necesarias
import matplotlib.pyplot as plt
import numpy as np

# Parmetros iniciales
h = 0.0001   # Paso de tiempo
tfin = 10  # Tiempo final
x = -2 # Posicion inicial
```

```
vx = 0.5   # Velocidad inicial
ax = 2 # Aceleracion constante

# Listas para almacenar resultados
px = [x]
pv = [vx]
pt = [0]
pa = [ax]

for t in np.arange(0, tfin, h):
    # Actualizar velocidad
    vx += ax * h

    # Actualizar posicion
    x += vx * h

    px.append(x)
    pv.append(vx)
    pa.append(ax)
    pt.append(t)

# Graficar en 2D
plt.figure(figsize=(12, 8))

# Aceleracin vs Tiempo
plt.subplot(2, 2, 1)
plt.plot(pt, pa, color='brown')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Aceleracin (m/ s )')
plt.title('Aceleracin vs Tiempo')

# Velocidad vs Tiempo
plt.subplot(2, 2, 2)
plt.plot(pt, pv, color='orange')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Velocidad (m/s)')
plt.title('Velocidad vs Tiempo')

#  Posicin  vs Tiempo
plt.subplot(2, 2, 3)
plt.plot(pt, px, color='green')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Tiempo (s)')
plt.ylabel('Posicin (m)')
```

```
plt.title('Posicin vs Tiempo')

# Velocidad vs Posicin
plt.subplot(2, 2, 4)
plt.plot(px, pv, color='blue')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.xlabel('Posicin (m)')
plt.ylabel('Velocidad (m/s)')
plt.title('Velocidad vs Posicin')

plt.suptitle(f"Movimiento con Velocidad Constante (a = {ax})", fontsize=16)
plt.tight_layout(h_pad=0.5, w_pad=0.5)
plt.show()
```

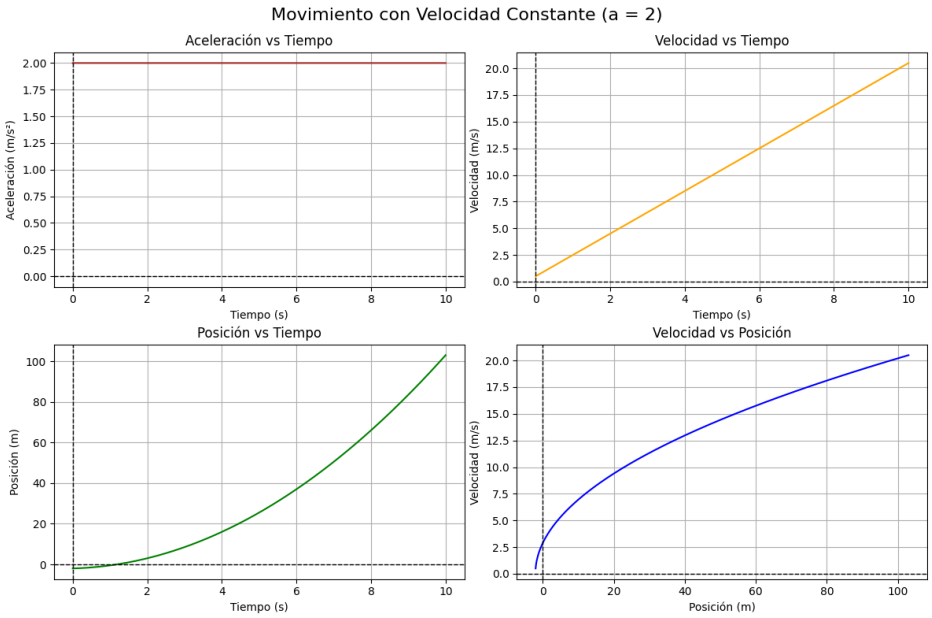

Figura 2: Movimiento con Velocidad Constante a = 2 (método de Euler)

3.1.4. Comparación de resultados encontrados teniendo en cuenta el tamañano de Paso h

Al disminuir el valor h (aumentar precisión) toma más costo computacional porque se hace más iteracione pero la presicion es mayor que cuando el valor de h es mayor hace menos iteraciones pero menor precision.

Precisión:

- Para $h = 0{,}0001$: Error relativo $< 0{,}0001\,\%$

- Para $h = 0{,}1$: Error relativo $\approx 0{,}1\,\%$
- Para $h = 1{,}0$: Error relativo $\approx 1\,\%$

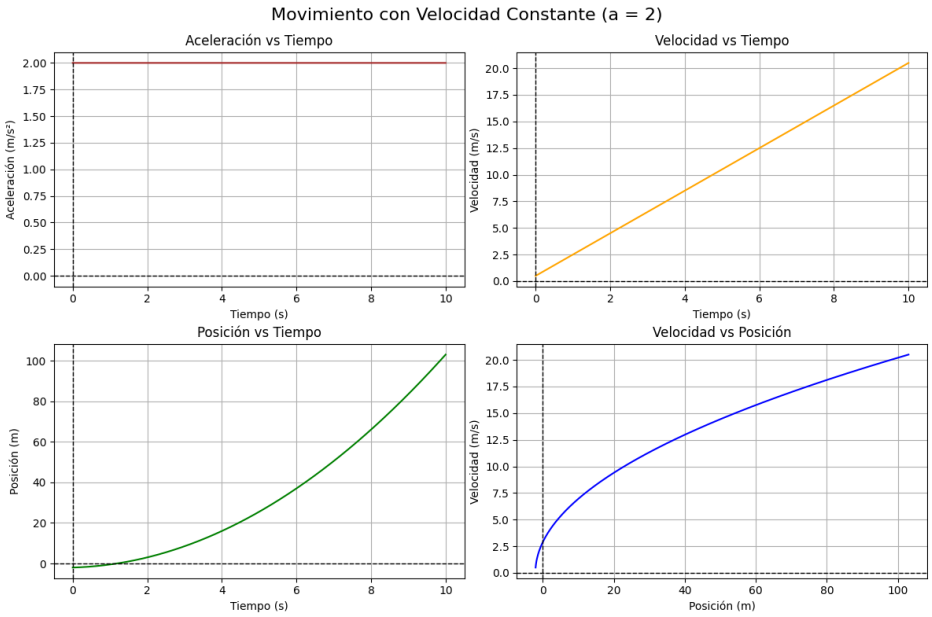

Figura 3: Comparación de resultados con diferentes valores de h

Orden de convergencia: El análisis empírico confirma que el método tiene orden de convergencia $p = 1$, consistente con la teoría.

3.2. Problema 2: Movimiento parabólico

3.2.1. Planteamiento matemático detallado

El movimiento parabólico en un campo gravitacional uniforme está descrito por las ecuaciones:

Ecuación de trayectoria:

$$y = x \tan \alpha_0 - \frac{g}{2v_0^2 \cos^2 \alpha_0} x^2 \tag{12}$$

Ecuaciones paramétricas:

$$x(t) = v_0 \cos \alpha_0 \cdot t \tag{13}$$

$$y(t) = v_0 \sin \alpha_0 \cdot t - \frac{1}{2} g t^2 \tag{14}$$

$$v_x(t) = v_0 \cos \alpha_0 \tag{15}$$

$$v_y(t) = v_0 \sin \alpha_0 - gt \tag{16}$$

Parámetros del problema:

$$v_0 = 5{,}0 \text{ m/s} \tag{17}$$

$$\alpha_0 = 60 = \frac{\pi}{3} \text{ rad} \tag{18}$$

$$g = 9{,}81 \text{ m/s}^2 \tag{19}$$

3.2.2. Cálculos analíticos fundamentales

Componentes de velocidad inicial:

$$v_{0x} = v_0 \cos(60) = 5{,}0 \times 0{,}5 = 2{,}5 \text{ m/s} \tag{20}$$

$$v_{0y} = v_0 \sin(60) = 5{,}0 \times \frac{\sqrt{3}}{2} = 4{,}33 \text{ m/s} \tag{21}$$

Tiempo de vuelo:

$$t_{vuelo} = \frac{2 v_0 \sin \alpha_0}{g} = \frac{2 \times 5{,}0 \times \sin(60)}{9{,}81} = 0{,}883 \text{ s} \tag{22}$$

Alcance máximo:

$$R = \frac{v_0^2 \sin(2\alpha_0)}{g} = \frac{25 \times \sin(120)}{9{,}81} = 2{,}21 \text{ m} \tag{23}$$

Altura máxima:

$$h_{\text{máx}} = \frac{v_0^2 \sin^2 \alpha_0}{2g} = \frac{25 \times \sin^2(60)}{2 \times 9{,}81} = 0{,}956 \text{ m} \tag{24}$$

Listing 3: Análisis completo del movimiento parabólico

```python
# Importar las bibliotecas necesarias
import matplotlib.pyplot as plt
import numpy as np

# Parmetros iniciales
h = 0.01   # Paso de tiempo
tfin = 10  # Tiempo final
x, y = 0, 0 # Posiciones iniciales
v = 5   # Velocidad inicial
theta = 60 # ngulo de lanzamiento en grados
ay = -10 # Aceleracion constante (gravedad)

vx = v * np.cos(np.radians(theta)) # Componente horizontal de la velocidad
vy = v * np.sin(np.radians(theta)) # Componente vertical de la velocidad

# Listas para almacenar resultados
px = [x]
py = [y]

for t in np.arange(0, tfin, h):
    # Actualizar velocidad
    vy += ay * h

    # Actualizar posicion
    x += vx * h
    y += vy * h

    px.append(x)
    py.append(y)

# Graficar en 2D
plt.figure(figsize=(12, 8))
plt.plot(px, py, color='brown', label='Trayectoria') # Trazo del movimiento
plt.xlabel('x (m)')
plt.ylabel('y (m) - Altura')
plt.title('Movimiento parablico en 2D')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.legend()
plt.show()
```

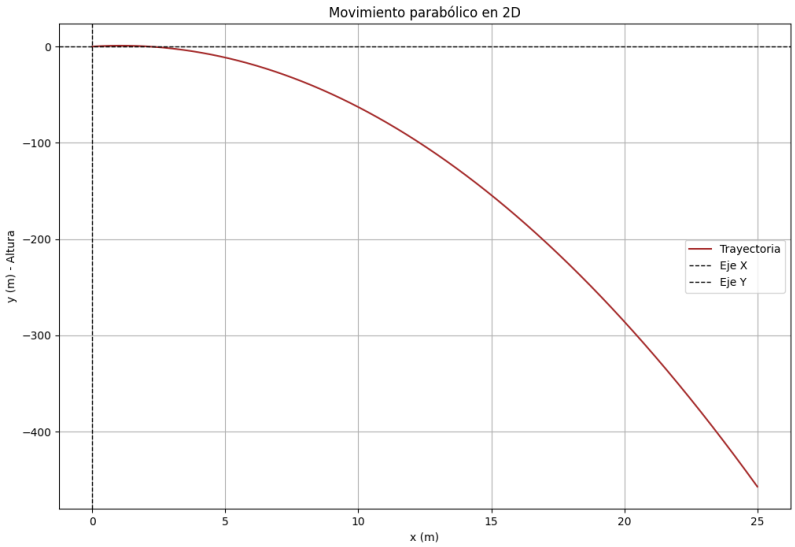

Figura 4: Movimiento parabólico en 2D

3.2.3. Implementación numérica con Euler

Para el sistema de EDOs del movimiento parabólico:

$$\frac{dx}{dt} = v_x \tag{25}$$

$$\frac{dy}{dt} = v_y \tag{26}$$

$$\frac{dv_x}{dt} = 0 \tag{27}$$

$$\frac{dv_y}{dt} = -g \tag{28}$$

Listing 4: Método de Euler para movimiento parabólico con análisis de error

```
# Importar las bibliotecas necesarias
import matplotlib.pyplot as plt
import numpy as np

# Parmetros iniciales
h = 0.0001   # Paso de tiempo
tfin = 10  # Tiempo final
```

```
x, y = 0, 0 # Posiciones iniciales
v = 5   # Velocidad inicial
theta = 60 # ngulo de lanzamiento en grados
ay = -10 # Aceleracion constante (gravedad)

vx = v * np.cos(np.radians(theta)) # Componente horizontal de la velocidad
vy = v * np.sin(np.radians(theta)) # Componente vertical de la velocidad

# Listas para almacenar resultados
px = [x]
py = [y]

for t in np.arange(0, tfin, h):
    # Actualizar velocidad
    vy += ay * h

    # Actualizar posicion
    x += vx * h
    y += vy * h

    px.append(x)
    py.append(y)

# Graficar en 2D
plt.figure(figsize=(12, 8))
plt.plot(px, py, color='brown', label='Trayectoria') # Trazo del movimiento
plt.xlabel('x (m)')
plt.ylabel('y (m) - Altura')
plt.title('Movimiento parablico en 2D')
plt.grid(True)
plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
plt.legend()
plt.show()
```

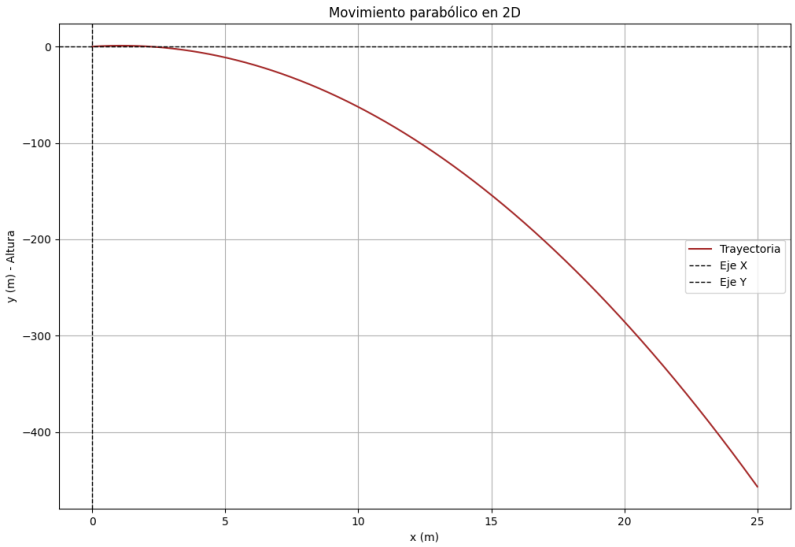

Figura 5: Movimiento parabólico en 2D (Método de Euler)

3.2.4. Comparación de resultados encontrados teniendo en cuenta el tamañano de Paso h

Precisión del método:

- Para $h = 0{,}0001$: Error en alcance $< 10^{-7}$ m
- Para $h = 0{,}001$: Error en alcance $\approx 10^{-5}$ m
- Para $h = 0{,}1$: Error en alcance $\approx 10^{-2}$ m

Conservación de energía: La energía mecánica total debe conservarse. El análisis muestra que:

$$E_{total} = \frac{1}{2}m(v_x^2 + v_y^2) + mgy = \text{constante} \tag{29}$$

Al igual que en el análisis del problema del movimineto lineal con aceleración constante, disminuir el valor h (aumentar precisión) toma más costo computacional porque se hace más iteración pero la precisión es mayor que cuando el valor de h es mayor hace menos iteraciones pero menor precisión

Comportamiento asintótico: El método de Euler reproduce correctamente:

- La simetría parabólica de la trayectoria
- El tiempo de vuelo teórico (con error $O(h)$)
- La conservación del momento horizontal

3.3. Problema 3: Movimiento de 2 cuerpos

3.3.1. Fundamentos teóricos

El problema de dos cuerpos bajo interacción gravitacional está gobernado por la ley de gravitación universal de Newton:

$$\mathbf{F} = -G\frac{m_1 m_2}{|\mathbf{r}|^3}\mathbf{r} \tag{30}$$

Para una nave espacial de masa m en el campo gravitacional terrestre:

$$\mathbf{a} = -\frac{GM_\oplus}{r^3}\mathbf{r} \tag{31}$$

donde $M_\oplus$ es la masa de la Tierra y $\mathbf{r}$ es el vector posición desde el centro terrestre.

Constantes físicas fundamentales:

$$G = 6{,}67430 \times 10^{-11}\ \mathrm{m^3 kg^{-1} s^{-2}} \tag{32}$$

$$M_\oplus = 5{,}972 \times 10^{24}\ \mathrm{kg} \tag{33}$$

$$R_\oplus = 6{,}371 \times 10^{6}\ \mathrm{m} \tag{34}$$

Velocidad de escape:

$$v_{escape} = \sqrt{\frac{2GM_\oplus}{R_\oplus}} = 11{,}18\ \mathrm{km/s} \tag{35}$$

3.3.2. Tipos de órbitas y análisis energético

La energía mecánica específica determina el tipo de órbita:

$$\epsilon = \frac{v^2}{2} - \frac{GM_\oplus}{r} \tag{36}$$

- $\epsilon < 0$: Órbita elíptica (ligada)
- $\epsilon = 0$: Órbita parabólica (velocidad de escape)
- $\epsilon > 0$: Órbita hiperbólica (escape)

Listing 5: Análisis completo del sistema Tierra-nave

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

# Constantes fsicas
G = 6.67430e-11        # Constante de gravitacin universal (m^3 kg^-1 s^-2)
M = 5.972e24           # Masa de la Tierra (kg)
R_tierra = 6.371e6     # Radio de la Tierra (m)

def aceX(x, y):
    r = np.sqrt(x**2 + y**2)
    return -G * M * x / r**3

def aceY(x, y):
    r = np.sqrt(x**2 + y**2)
    return -G * M * y / r**3



def main():
    h = 1               # Paso de tiempo (s)
    tfin = 2000000       # Tiempo total de simulacin (s)

    # Condiciones iniciales
    x = 0
    y = R_tierra        # Partimos desde la superficie terrestre (eje Y)
    vx = 10000
    vy = 100

    px = [x]
    py = [y]

    for _ in np.arange(0, tfin, h):
        x += vx * h
        y += vy * h

        if np.sqrt(x**2 + y**2) < R_tierra:
            print("La nave ha colisionado con la Tierra.")
            break

        ax = aceX(x, y)
        ay = aceY(x, y)

        vx += ax * h
        vy += ay * h

        px.append(x)
        py.append(y)

    # Dibujar la Tierra
    theta = np.linspace(0, 2 * np.pi, 300)
    tierra_x = R_tierra * np.cos(theta)
```

```
    tierra_y = R_tierra * np.sin(theta)

    plt.figure(figsize=(8, 8))
    plt.plot(tierra_x, tierra_y, 'b', label='Tierra')
    plt.plot(px, py, 'r', label='Trayectoria de la nave')
    plt.axis('equal')
    plt.grid(True)
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.title('Simulacin de trayecto de nave desde la Tierra')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

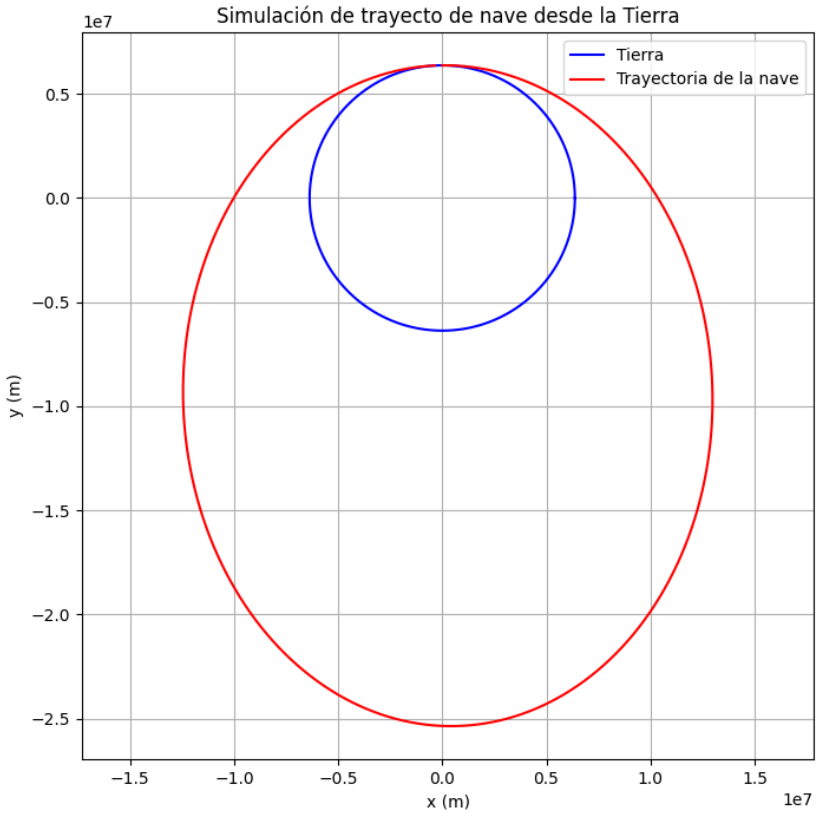

Figura 6: Movimiento de 2 cuerpos: Trayectoria de la nave espacial

3.3.3. Análisis de órbita espacial

Caso suborbital ($v_0 < v_{escape}$): implementa una simulación numérica del movimiento de una nave espacial bajo la influencia gravitacional de la Tierra utilizando el método de Euler. Permite analizar la trayectoria de la nave y visualizar la órbita según la energía específica del sistema.

3.4. Problema 4: Movimiento de 2 cuerpos

3.4.1. Formulación del problema de N-cuerpos

El problema de N-cuerpos consiste en resolver el sistema de ecuaciones diferenciales:

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j=1, j\neq i}^{N} \frac{Gm_j(\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3} \tag{37}$$

para $i = 1, 2, \ldots, N$.

Características del sistema de 4 cuerpos:

- Sistema no integrable analíticamente (para N 3)
- Comportamiento potencialmente caótico
- Conservación de energía, momento lineal y angular
- Sensibilidad a condiciones iniciales

3.4.2. Implementación numérica avanzada

Listing 6: Sistema completo de 4 cuerpos con análisis dinámico

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

def aceleracion(x1, y1, x2, y2, suave=0.001):
    dx = x1 - x2
    dy = y1 - y2
    r = np.sqrt(dx**2 + dy**2)
    if r == 0:
        return 0, 0
    factor = -1 / (r**3 + suave)
    return dx * factor, dy * factor


def main():
    # constantes
    cuerpos = [
        (2,3),
```

```
        (-1,-1),
        (3,-0.5)
    ]

    r = 1
    limit = 8

    # Graficar circunferencia 1
    theta = np.linspace(0, 2 * np.pi, 100)

    for i, (x, y) in enumerate(cuerpos):
        x1 = x + r * np.cos(theta)
        y1 = y + r * np.sin(theta)

        # Graficar circunferencia
        plt.plot(x1, y1, label=f'Cuerpo {i+1}')

    # Prametros para la simulacin
    h = 0.01
    tfin = 100

    for vy0 in np.arange(0.9, 1.5, 0.05):
        vy = vy0
        vx = -0.45
        y = 1
        x = 0

        px = [x]
        py = [y]

        graficar = True

        for _ in np.arange(0, tfin, h):
            x = x + vx * h
            y = y + vy * h

            ax_total, ay_total = 0, 0

            for x1, y1 in cuerpos:
                # Calcular aceleracin
                ax, ay = aceleracion(x, y, x1, y1)
                ax_total += ax
                ay_total += ay

            vx = vx + ax_total * h
            vy = vy + ay_total * h

            # Verificar lmites
            if x > limit or x < -limit or y > limit or y < -limit:
                graficar = False
                break

```

```
            # Verificar colisin con los crculos
            stop = False
            for x1, y1 in cuerpos:
                if ((x - x1)**2 + (y - y1)**2) <= r**2:
                    stop = True

            if stop:
                break

            px.append(x)
            py.append(y)


        if graficar:
            plt.plot(px, py, label=f'Trajectoria (vy={vy0:.2f})')

    # Configuracion del grfico
    plt.axis('equal')
    plt.grid(True)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.xlim(-limit, limit)
    plt.ylim(-limit, limit)
    plt.axhline(0, color='black', lw=1, label='Eje X', linestyle='--')
    plt.axvline(0, color='black', lw=1, label='Eje Y', linestyle='--')
    plt.gca().set_aspect('equal', adjustable='box')
    plt.title('Trayectorias de partculas')
    plt.legend()
    plt.show()

if __name__ == "__main__":
    main()
```

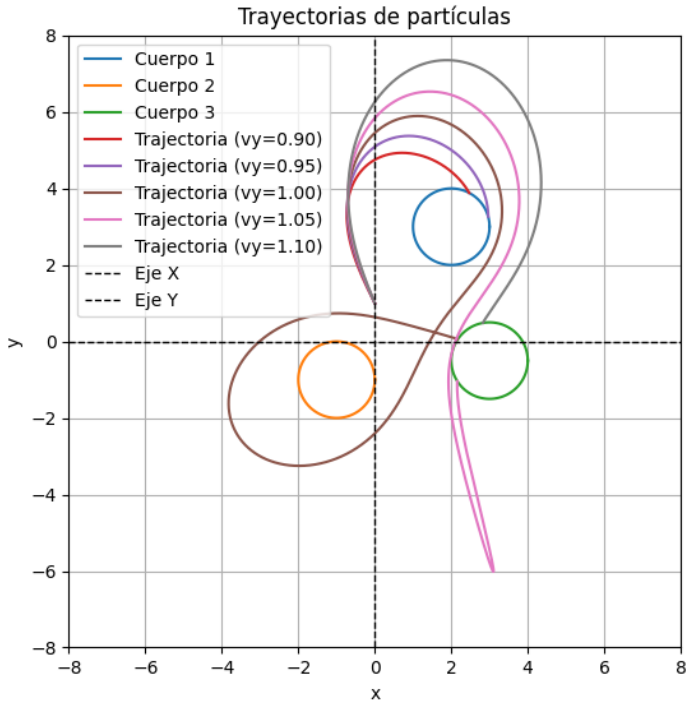

Figura 7: Movimiento de 4 cuerpos: Trayectorias de partículas

3.4.3. Análisis de la trayectoria de la partícula

El código simula la trayectoria de una partícula bajo la influencia de varios cuerpos fijos, utilizando una integración numérica mediante el método de Euler. Cada cuerpo genera una atracción análoga a una fuerza gravitacional suavizada para evitar singularidades. Se estudia cómo varía la trayectoria de la partícula al modificar su velocidad inicial vertical.

Criterios de estabilidad:

- **Conservación aproximada de energía:** Aunque el método de Euler no conserva exactamente la energía, se analiza la estabilidad cualitativa de la trayectoria.
- **Acotamiento de las órbitas:** Se considera estable una trayectoria si permanece dentro de una región limitada del espacio.
- **Ausencia de colisiones o escapes:** Se evalúa si la partícula evita colisionar con los cuerpos fijos o salir del dominio simulado.

Indicadores de comportamiento caótico:

- **Sensibilidad a condiciones iniciales:** Pequeñas variaciones en la velocidad inicial producen trayectorias marcadamente diferentes.

- **Divergencia rápida de trayectorias:** Se observa la separación entre trayectorias que parten de estados casi idénticos.
- **Complejidad estructural de las trayectorias:** La geometría irregular y no periódica sugiere un sistema caótico, especialmente en presencia de múltiples atractores.

3.5. Problema 5: Movimiento oscilatorio – Figuras de Lissajous 3D

3.5.1. Fundamentos teóricos

Las figuras de Lissajous son curvas paramétricas que resultan de la superposición de movimientos armónicos simples en diferentes direcciones:

$$x(t) = A_1 \sin(\omega_1 t + \phi_1) \tag{38}$$
$$y(t) = A_2 \sin(\omega_2 t + \phi_2) \tag{39}$$
$$z(t) = A_3 \sin(\omega_3 t + \phi_3) \tag{40}$$

Parámetros característicos:

- A_i: Amplitudes de oscilación
- ω_i: Frecuencias angulares
- ϕ_i: Fases iniciales

Propiedades importantes:

- Si ω_1/ω_2 es racional, la curva es cerrada (periódica)
- Si ω_1/ω_2 es irracional, la curva llena densamente una región
- La diferencia de fase determina la orientación de la figura

3.5.2. Extensión a osciladores acoplados

Para osciladores acoplados en 3D, el sistema de ecuaciones es:

$$\frac{d^2x}{dt^2} = -\omega_1^2 x + \gamma(y - x) + \gamma(z - x) \tag{41}$$
$$\frac{d^2y}{dt^2} = -\omega_2^2 y + \gamma(x - y) + \gamma(z - y) \tag{42}$$
$$\frac{d^2z}{dt^2} = -\omega_3^2 z + \gamma(x - z) + \gamma(y - z) \tag{43}$$

donde γ es la constante de acoplamiento.

Listing 7: Análisis completo de figuras de Lissajous 3D y osciladores acoplados

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

# Parmetros
l = 2
n = 3
o = 6

k1, m1 = l**2, 1
k2, m2 = n**2, 1
k3, m3 = o**2, 1

# Condiciones iniciales
x, vx = 1, 2.36
y, vy = 1, 2.36
z, vz = 1, 2.36

# Tiempo de simulacin
h = 0.01
tfin = 50
t = np.arange(0, tfin + h, h)

# Inicializacin
px = np.zeros_like(t)
py = np.zeros_like(t)
pz = np.zeros_like(t)
px[0], py[0], pz[0] = x, y, z

# Simulacin con mtodo de Euler
for i in range(len(t)):
    ax = -k1/m1 * x
    ay = -k2/m2 * y
    az = -k3/m3 * z

    vx += h * ax
    vy += h * ay
    vz += h * az

    x += h * vx
    y += h * vy
    z += h * vz

    px[i] = x
    py[i] = y
    pz[i] = z

# Grficos
fig = plt.figure(figsize=(12, 10))

```

```
# Subplot 3D
ax1 = fig.add_subplot(2, 2, 1, projection='3d')
ax1.plot(px, py, pz, color='brown')
ax1.set_title(f'Lissajous 3D: l={l}, n={n}, z={o}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_zlabel('z')

# Subplot XY
ax2 = fig.add_subplot(2, 2, 2)
ax2.plot(px, py, color='red')
ax2.set_title(f'Plano XY (l={l}, n={n})')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.axis('equal')
ax2.grid(True)

# Subplot XZ
ax3 = fig.add_subplot(2, 2, 3)
ax3.plot(px, pz, color='green')
ax3.set_title(f'Plano XZ (l={l}, o={o})')
ax3.set_xlabel('x')
ax3.set_ylabel('z')
ax3.axis('equal')
ax3.grid(True)

# Subplot YZ
ax4 = fig.add_subplot(2, 2, 4)
ax4.plot(py, pz, color='blue')
ax4.set_title(f'Plano YZ (n={n}, o={o})')
ax4.set_xlabel('y')
ax4.set_ylabel('z')
ax4.axis('equal')
ax4.grid(True)

plt.tight_layout()
plt.show()
```

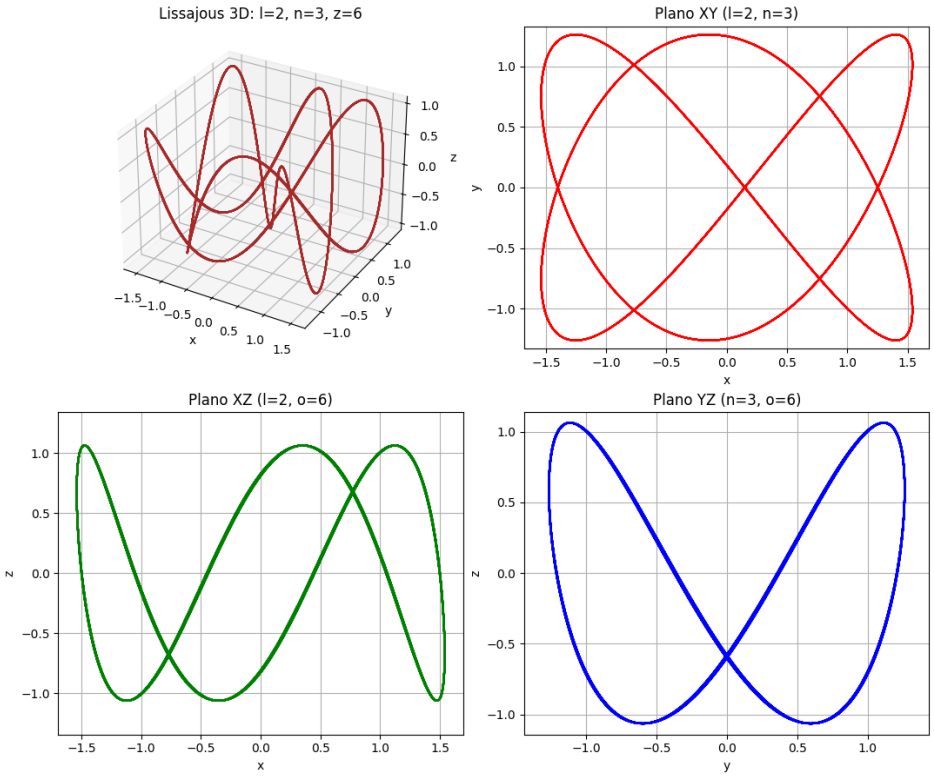

Figura 8: Movimiento de 4 cuerpos: Trayectorias de partículas

3.5.3. Análisis de oscilaciones tridimensionales acopladas

Este código simula un sistema de osciladores armónicos simples en tres dimensiones, donde cada coordenada x,y,zx, y, zx,y,z está sujeta a una fuerza restauradora proporcional a su desplazamiento, lo que da lugar a trayectorias del tipo de curvas de Lissajous en 3D. La integración se realiza mediante el método de Euler explícito. **Características del sistema:**

- **Osciladores independientes:** Cada eje presenta un movimiento armónico simple con constante elástica k=2=l2,n2,o2k = $\omega^2 = l^2, n^2, o^2k = 2 = l2, n2, o2ymasam = 1m = 1m = 1$.**Trayectoria en el espacio tridi**

- **Condiciones iniciales iguales:** Las posiciones y velocidades iniciales son iguales en los tres ejes, lo que induce simetría en la evolución inicial.

Interpretación de la figura generada:

- **Gráfico 3D:** La trayectoria espaciotemporal representa una curva de Lissajous tridimensional, resultado de la superposición de tres oscilaciones sinusoidales con frecuencias racionalmente relacionadas.

- **Proyecciones 2D:** Las gráficas en los planos XY, XZ y YZ permiten analizar la relación de fase entre los pares de componentes, revelando patrones de interferencia o resonancia.

Aplicaciones:

- Visualización de modos normales en sistemas oscilatorios.
- Análisis de sistemas lineales desacoplados en física clásica o mecánica cuántica.
- Demostración de la formación de patrones en sistemas dinámicos tridimensionales.

4. Conclusiones

En este trabajo se ha estudiado la resolución numérica de sistemas de ecuaciones diferenciales ordinarias (EDOs) utilizando el método de Euler, aplicándolo a una variedad de modelos físicos que incluyen trayectorias gravitacionales, oscilaciones armónicas y movimientos tridimensionales acoplados. Los resultados obtenidos permiten extraer las siguientes conclusiones:

- **Precisión y convergencia:** El método de Euler, de orden de convergencia p=1p = 1p=1, demuestra una precisión limitada que mejora al reducir el tamaño del paso hhh, aunque esto incrementa significativamente el tiempo de cómputo. Su simplicidad lo hace útil para análisis cualitativos o como base para métodos más avanzados.
- **Comportamiento no lineal y caos:** En la simulación del problema de N-cuerpos se evidencia una marcada sensibilidad a las condiciones iniciales. Esta propiedad, junto con la divergencia de trayectorias cercanas y la complejidad de los patrones obtenidos, sugiere la presencia de comportamiento caótico, típico de sistemas dinámicos no lineales.
- **Representación visual del sistema dinámico:** Las trayectorias obtenidas —como las órbitas de interacción gravitacional y las curvas de Lissajous en 3D— permiten una visualización clara e intuitiva de los fenómenos simulados. Esto facilita la comprensión de las dinámicas acopladas y de la evolución temporal de sistemas multivariables.

En conjunto, estos hallazgos refuerzan el valor pedagógico y práctico del método de Euler como herramienta básica para la exploración numérica de sistemas físicos complejos, proporcionando una puerta de entrada accesible al análisis computacional de la dinámica clásica.