Física I

Tarea 2 - Ejercicio 1

Nombre del Estudiante

September 26, 2025

Movimiento de Proyectiles en 2D (sin arrastre)

1. Ecuaciones de posición y velocidad

Configuración del problema: Consideramos un proyectil lanzado desde el origen ($x_0 = 0, y_0 = 0$) con velocidad inicial v_0 y ángulo θ respecto a la horizontal. La única fuerza que actúa es la gravedad, por lo que la aceleración es $\vec{a} = (0, -g)$, donde $g \approx 9.8\,\mathrm{m/s^2}$.

Deducción de las ecuaciones: Partimos de las ecuaciones básicas del movimiento con aceleración constante:

$$\begin{aligned} v_x(t) &= v_{0x} + a_x t \\ v_y(t) &= v_{0y} + a_y t \\ x(t) &= x_0 + v_{0x} t + \tfrac{1}{2} a_x t^2 \\ y(t) &= y_0 + v_{0y} t + \tfrac{1}{2} a_y t^2 \end{aligned}$$

Sustituyendo las condiciones iniciales y las componentes de la aceleración:

$$\begin{aligned} v_{0x} &= v_0 \cos\theta, \quad a_x = 0 \\ v_{0y} &= v_0 \sin\theta, \quad a_y = -g \end{aligned}$$

Obtenemos las ecuaciones finales:

$$\begin{aligned} v_x(t) &= v_0 \cos\theta, \\ v_y(t) &= v_0 \sin\theta - gt, \\ x(t) &= v_0 \cos\theta \; t, \\ y(t) &= v_0 \sin\theta \; t - \tfrac{1}{2} g t^2. \end{aligned}$$

Interpretación física: La velocidad horizontal permanece constante porque no hay aceleración en esa dirección (componente $a_x = 0$), mientras que la velocidad vertical cambia linealmente con el tiempo debido a la aceleración gravitacional.

2. Alcance horizontal

Tiempo de vuelo: El proyectil regresa al suelo cuando $y(T) = 0$. Resolvemos:

$$y(T) = v_0 \sin\theta \, T - \tfrac{1}{2} g T^2 = 0$$
$$T(v_0 \sin\theta - \tfrac{1}{2} g T) = 0$$

Las soluciones son $T = 0$ (instante inicial) y:

$$T = \frac{2 v_0 \sin\theta}{g}.$$

Alcance horizontal: Sustituimos el tiempo de vuelo en la ecuación de $x(t)$:

$$R(\theta) = x(T) = v_0 \cos\theta \cdot T = v_0 \cos\theta \cdot \frac{2 v_0 \sin\theta}{g} = \frac{2 v_0^2 \sin\theta \cos\theta}{g}.$$

Usando la identidad trigonométrica $\sin(2\theta) = 2 \sin\theta \cos\theta$, obtenemos:

$$R(\theta) = \frac{v_0^2}{g} \sin(2\theta).$$

Maximización del alcance: Para encontrar el ángulo que maximiza $R(\theta)$, derivamos respecto a θ:

$$\frac{dR}{d\theta} = \frac{v_0^2}{g} \cdot 2\cos(2\theta) = 0$$
$$\cos(2\theta) = 0 \Rightarrow 2\theta = 90^\circ \Rightarrow \theta = 45^\circ.$$

El alcance máximo es:

$$R_{\max} = \frac{v_0^2}{g} \sin(90^\circ) = \frac{v_0^2}{g}.$$

3. Altura máxima

Tiempo hasta la altura máxima: La altura máxima ocurre cuando la velocidad vertical se hace cero ($v_y = 0$):

$$v_y(t_{\text{pico}}) = v_0 \sin\theta - g t_{\text{pico}} = 0$$
$$t_{\text{pico}} = \frac{v_0 \sin\theta}{g}.$$

Altura máxima: Sustituimos t_{pico} en la ecuación de $y(t)$:

$$\begin{aligned} H(\theta) = y(t_{\text{pico}}) &= v_0 \sin\theta \cdot t_{\text{pico}} - \tfrac{1}{2} g t_{\text{pico}}^2 \\ &= v_0 \sin\theta \cdot \frac{v_0 \sin\theta}{g} - \tfrac{1}{2} g \left(\frac{v_0 \sin\theta}{g} \right)^2 \\ &= \frac{v_0^2 \sin^2\theta}{g} - \frac{v_0^2 \sin^2\theta}{2g} = \frac{v_0^2 \sin^2\theta}{2g}. \end{aligned}$$

Maximización de la altura: La altura máxima se alcanza cuando $\sin^2\theta$ es máximo:

$$\sin^2\theta_{\max} = 1 \Rightarrow \theta = 90^\circ \quad \text{(lanzamiento vertical)}$$
$$H_{\max} = \frac{v_0^2}{2g}.$$

4. Relación entre tiempo de vuelo y tiempo de subida

Simetría del movimiento: Notamos que $t_{\text{pico}} = \frac{T}{2}$, lo que significa que el tiempo de subida es igual al tiempo de bajada. Esto ocurre porque la aceleración es constante y no hay fuerzas disipativas.

Demostración:

$$t_{\text{pico}} = \frac{v_0 \sin\theta}{g}, \quad T = \frac{2v_0 \sin\theta}{g} \Rightarrow t_{\text{pico}} = \frac{T}{2}.$$

5. Observaciones y conclusiones

- **Alcance máximo:** Ocurre a 45° debido a que $\sin(2\theta)$ alcanza su valor máximo de 1 cuando $2\theta = 90^\circ$.
- **Altura máxima:** Se maximiza con lanzamiento vertical (90°) porque toda la velocidad inicial se dirige contra la gravedad.
- **Relación entre $R_{\max}$ y $H_{\max}$:**

  $$\frac{R_{\max}}{H_{\max}} = \frac{v_0^2/g}{v_0^2/(2g)} = 2$$

  Para un ángulo de 45°, el alcance máximo es exactamente el doble de la altura máxima alcanzable con lanzamiento vertical.
- **Trade-off:** Existe una compensación entre altura y alcance. Ángulos mayores a 45° aumentan la altura pero reducen el alcance, y viceversa.
- **Complementariedad de ángulos:** Para un mismo alcance R, existen dos ángulos complementarios (θ y $90^\circ - \theta$) que producen el mismo resultado, excepto cuando $\theta = 45^\circ$.

Planteamiento del problema

El movimiento parabólico en dos dimensiones sin resistencia del aire está gobernado por el sistema de ecuaciones diferenciales:

$$\frac{dx}{dt} = v_x, \qquad \frac{dv_x}{dt} = 0,$$
$$\frac{dy}{dt} = v_y, \qquad \frac{dv_y}{dt} = -g,$$

con condiciones iniciales:

$$x(0) = 0, \quad y(0) = 0, \quad v_x(0) = v_0 \cos\theta, \quad v_y(0) = v_0 \sin\theta.$$

La solución analítica para la trayectoria es:

$$y(x) = x \tan\theta - \frac{g}{2v_0^2 \cos^2\theta}\, x^2.$$

Método de Euler para Movimiento de Proyectiles

Fundamento Matemático

El método de Euler es un procedimiento numérico de primer orden para resolver ecuaciones diferenciales ordinarias. Para un sistema de la forma:

$$\frac{dy}{dt} = f(t, y)$$

la aproximación discreta es:

$$y_{n+1} = y_n + hf(t_n, y_n),$$

donde h es el paso temporal (tamaño del intervalo de discretización).

Aplicación al movimiento de proyectiles: Nuestro sistema tiene cuatro variables: posición (x, y) y velocidad (v_x, v_y). Las ecuaciones diferenciales son:

$$\begin{aligned}
\frac{dx}{dt} &= v_x \\
\frac{dy}{dt} &= v_y \\
\frac{dv_x}{dt} &= 0 \quad \text{(no hay aceleración horizontal)} \\
\frac{dv_y}{dt} &= -g \quad \text{(aceleración gravitacional)}
\end{aligned}$$

Aplicando el método de Euler a cada ecuación:

$$\begin{aligned}
x_{n+1} &= x_n + hv_{x,n}, \\
y_{n+1} &= y_n + hv_{y,n}, \\
v_{x,n+1} &= v_{x,n} + h \cdot 0 = v_{x,n}, \\
v_{y,n+1} &= v_{y,n} + h(-g) = v_{y,n} - hg.
\end{aligned}$$

Interpretación física: Cada paso del método corresponde a:

- **Posición:** Movemos la partícula según su velocidad actual
- **Velocidad horizontal:** Permanece constante (conservación del momentum)
- **Velocidad vertical:** Disminuye linealmente debido a la gravedad

Implementación en Python: Explicación Paso a Paso

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

# ============================================
# PAR METROS F SICOS Y NUM RICOS
# ============================================

v0 = 20.0      # velocidad inicial [m/s] - t pico lanzamiento de
    proyectil
theta = np.pi/4  #  ngulo  de lanzamiento (45 grados =  /4
   radianes)
g = 9.81       # aceleraci n gravitacional [m/s ] - valor
   est ndar
h = 0.1        # paso de integraci n [s] - determina la
   precisi n
t_max = 3.0    # tiempo m ximo de simulaci n [s] - para evitar
   loops infinitos

# Explicaci n: Estos par metros controlan tanto la f sica del
   problema como la
# precisi n del m todo num rico. Un h m s peque o da mayor
   precisi n pero mayor costo computacional.

# ============================================
# CONDICIONES INICIALES
# ============================================

# Descomposici n de la velocidad inicial en componentes x e y
# vx0 = v0 * cos(theta): componente horizontal (coseno =
   adyacente/hipotenusa)
# vy0 = v0 * sin(theta): componente vertical (seno = opuesto/
   hipotenusa)
vx0 = v0 * np.cos(theta)
vy0 = v0 * np.sin(theta)

# Listas para almacenar la evoluci n temporal de las variables
# Iniciamos en el origen (0,0) con las velocidades calculadas
x, y = [0.0], [0.0]          # Posiciones iniciales
vx, vy = [vx0], [vy0]        # Velocidades iniciales
t_values = [0.0]             # Tiempos (para tracking temporal)

print(f"Condiciones iniciales:")
print(f"Velocidad inicial: {v0:.2f} m/s")
print(f" ngulo : {np.degrees(theta):.1f}  ")
print(f"vx0: {vx0:.2f} m/s, vy0: {vy0:.2f} m/s")

# ====================================
# M TODO DE EULER - INTEGRACI N NUM RICA
# ====================================
```

```
t = 0.0
step = 0
print("\nIniciando integraci n num rica...")

# El loop contin a mientras el proyectil est  arriba del suelo
   (y >= 0)
while y[-1] >= 0 and t < t_max:

    # --- PASO 1: Calcular nuevas posiciones usando velocidades
       actuales ---
    # x_new = x_actual +  t  * vx_actual (movimiento horizontal
       uniforme)
    # y_new = y_actual +  t  * vy_actual (movimiento vertical
       acelerado)
    x_new = x[-1] + h * vx[-1]
    y_new = y[-1] + h * vy[-1]

    # --- PASO 2: Calcular nuevas velocidades ---
    # vx permanece constante (no hay fuerza horizontal)
    # vy disminuye debido a la gravedad: vy_new = vy_actual - g*
       t
    vx_new = vx[-1]
    vy_new = vy[-1] - h * g

    # --- PASO 3: Almacenar los nuevos valores ---
    x.append(x_new)
    y.append(y_new)
    vx.append(vx_new)
    vy.append(vy_new)
    t += h
    t_values.append(t)
    step += 1

    # Informaci n de progreso cada 10 pasos
    if step % 10 == 0:
        print(f"Paso {step}: t = {t:.2f}s, x = {x_new:.2f}m, y =
           {y_new:.2f}m")

print(f"Simulaci n completada: {step} pasos, tiempo final = {t
   :.2f}s")
print(f"Alcance num rico: {x[-1]:.2f} m")

# ==================================
# SOLUCI N ANAL TICA EXACTA (PARA COMPARACI N)
# ==================================

# La soluci n exacta para movimiento parab lico sin rozamiento:
# x(t) = v0*cos(  )*t
# y(t) = v0*sin(  )*t - 0.5*g*t
```

```
# Eliminando el tiempo: y(x) = x*tan(  ) - (g*x  )/(2*v0  *cos (
    ))

print("\nCalculando soluci n anal tica...")

# Crear array de posiciones x para evaluar la trayectoria
   anal tica
x_analytic = np.linspace(0, max(x), 200)

# Calcular las y correspondientes usando la ecuaci n de la
   trayectoria
# y(x) = x*tan(  ) - (g*x  )/(2*v0  *cos (  ))
tan_theta = np.tan(theta)
cos2_theta = np.cos(theta)**2
denominador = 2 * v0**2 * cos2_theta

y_analytic = x_analytic * tan_theta - (g * x_analytic**2) /
   denominador

# Tambi n calculamos el alcance te rico exacto
alcance_teorico = (v0**2 * np.sin(2*theta)) / g
altura_max_teorica = (v0**2 * np.sin(theta)**2) / (2*g)

print(f"Alcance te rico: {alcance_teorico:.2f} m")
print(f"Altura m xima te rica: {altura_max_teorica:.2f} m")

# ============================
# VISUALIZACI N Y COMPARACI N
# ============================

plt.figure(figsize=(10, 6))

# Graficar soluci n num rica (m todo de Euler)
plt.plot(x, y, 'bo-', markersize=3, linewidth=1, label=f'Euler (h
   ={h}s)')

# Graficar soluci n anal tica exacta
plt.plot(x_analytic, y_analytic, 'r-', linewidth=2, label='
   Soluci n anal tica')

# Configuraci n del gr fico
plt.xlabel("Distancia horizontal, x [m]", fontsize=12)
plt.ylabel("Altura, y [m]", fontsize=12)
plt.title("Trayectoria de Proyectil: M todo de Euler vs
   Soluci n Anal tica", fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.axis('equal')  # Misma escala en ambos ejes para ver la forma
    real

# A adir informaci n adicional en el gr fico
```

```
plt.text(0.05, 0.95, f' v    = {v0} m/s,     = {np.degrees(theta)
   :.0f}   ',
         transform=plt.gca().transAxes, fontsize=10,
            verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha
            =0.8))

plt.tight_layout()
plt.show()

# ====================
# AN LISIS DEL ERROR
# ====================

print("\n" + "="*50)
print("AN LISIS DE PRECISI N")
print("="*50)

# Calcular error en el alcance
error_alcance = abs(x[-1] - alcance_teorico)
error_relativo = (error_alcance / alcance_teorico) * 100

print(f"Alcance num rico: {x[-1]:.4f} m")
print(f"Alcance te rico:  {alcance_teorico:.4f} m")
print(f"Error absoluto:   {error_alcance:.4f} m")
print(f"Error relativo:   {error_relativo:.2f}%")

# Sugerencia para mejorar la precisi n
if error_relativo > 1:
    print("\nRecomendaci n: Reducir el paso h para mayor
       precisi n")
    print(f"Pruebe con h = 0.01 o h = 0.001")
else:
    print("\n Buena precisi n! El m todo funciona bien para
       este paso h")
```

Análisis Detallado del Método

- **Precisión del método:** El método de Euler es de primer orden, lo que significa que el error es proporcional a h. Para pasos grandes ($h = 0.1$ s), el error acumulativo es significativo.
- **Estabilidad:** El método es condicionalmente estable. Para nuestro problema, la estabilidad requiere que h sea suficientemente pequeño comparado con las escalas de tiempo características del sistema.
- **Convergencia:** Al reducir h ($h = 0.01$ s, $h = 0.001$ s), la solución numérica converge a la analítica, pero el costo computacional aumenta.
- **Ventajas:**

  - Simple implementación
  - Fácil de entender
  - Bajo costo computacional por paso
- **Desventajas:**
  - Baja precisión para pasos grandes
  - Error acumulativo
  - Puede volverse inestable para sistemas rígidos
- **Mejoras posibles:** Métodos de Runge-Kutta (especialmente RK4) ofrecen mayor precisión con el mismo paso, o la misma precisión con pasos mayores.

Método de Euler para Movimiento de Proyectiles

Fundamento Matemático

El método de Euler es un procedimiento numérico de primer orden para resolver ecuaciones diferenciales ordinarias. Para un sistema de la forma:

$$\frac{dy}{dt} = f(t, y)$$

la aproximación discreta es:

$$y_{n+1} = y_n + hf(t_n, y_n),$$

donde h es el paso temporal (tamaño del intervalo de discretización).

Aplicación al movimiento de proyectiles: Nuestro sistema tiene cuatro variables: posición (x, y) y velocidad (v_x, v_y). Las ecuaciones diferenciales son:

$$\begin{aligned}
\frac{dx}{dt} &= v_x \\
\frac{dy}{dt} &= v_y \\
\frac{dv_x}{dt} &= 0 \quad \text{(no hay aceleración horizontal)} \\
\frac{dv_y}{dt} &= -g \quad \text{(aceleración gravitacional)}
\end{aligned}$$

Aplicando el método de Euler a cada ecuación:

$$\begin{aligned}
x_{n+1} &= x_n + hv_{x,n}, \\
y_{n+1} &= y_n + hv_{y,n}, \\
v_{x,n+1} &= v_{x,n} + h \cdot 0 = v_{x,n}, \\
v_{y,n+1} &= v_{y,n} + h(-g) = v_{y,n} - hg.
\end{aligned}$$

Interpretación física: Cada paso del método corresponde a:

- **Posición:** Movemos la partícula según su velocidad actual
- **Velocidad horizontal:** Permanece constante (conservación del momentum)
- **Velocidad vertical:** Disminuye linealmente debido a la gravedad

Implementación en Python: Explicación Paso a Paso

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

# ================================================
# PAR METROS  F SICOS  Y  NUM RICOS
# ================================================

v0 = 20.0       # velocidad inicial [m/s] - t pico lanzamiento de
    proyectil
theta = np.pi/4  #  ngulo  de lanzamiento (45 grados =  /4
   radianes)
g = 9.81        # aceleraci n gravitacional [m/s  ] - valor
   est ndar
h = 0.1         # paso de integraci n [s] - determina la
   precisi n
t_max = 3.0     # tiempo m ximo de simulaci n [s] - para evitar
   loops infinitos

# Explicaci n: Estos par metros controlan tanto la f sica del
   problema como la
# precisi n del m todo num rico. Un h m s peque o da mayor
   precisi n pero mayor costo computacional.

# ================================================
# CONDICIONES INICIALES
# ================================================

# Descomposici n de la velocidad inicial en componentes x e y
# vx0 = v0 * cos(theta): componente horizontal (coseno =
   adyacente/hipotenusa)
# vy0 = v0 * sin(theta): componente vertical (seno = opuesto/
   hipotenusa)
vx0 = v0 * np.cos(theta)
vy0 = v0 * np.sin(theta)

# Listas para almacenar la evoluci n temporal de las variables
# Iniciamos en el origen (0,0) con las velocidades calculadas
x, y = [0.0], [0.0]          # Posiciones iniciales
vx, vy = [vx0], [vy0]        # Velocidades iniciales
t_values = [0.0]             # Tiempos (para tracking temporal)
```

```
print(f"Condiciones iniciales:")
print(f"Velocidad inicial: {v0:.2f} m/s")
print(f" ngulo : {np.degrees(theta):.1f}  ")
print(f"vx0: {vx0:.2f} m/s, vy0: {vy0:.2f} m/s")

# ==================================================
# M TODO DE EULER - INTEGRACI N NUM RICA
# ==================================================

t = 0.0
step = 0
print("\nIniciando integraci n num rica...")

# El loop contin a mientras el proyectil est  arriba del suelo
   (y >= 0)
while y[-1] >= 0 and t < t_max:

    # --- PASO 1: Calcular nuevas posiciones usando velocidades
       actuales ---
    # x_new = x_actual +  t  * vx_actual (movimiento horizontal
       uniforme)
    # y_new = y_actual +  t  * vy_actual (movimiento vertical
       acelerado)
    x_new = x[-1] + h * vx[-1]
    y_new = y[-1] + h * vy[-1]

    # --- PASO 2: Calcular nuevas velocidades ---
    # vx permanece constante (no hay fuerza horizontal)
    # vy disminuye debido a la gravedad: vy_new = vy_actual - g*
        t
    vx_new = vx[-1]
    vy_new = vy[-1] - h * g

    # --- PASO 3: Almacenar los nuevos valores ---
    x.append(x_new)
    y.append(y_new)
    vx.append(vx_new)
    vy.append(vy_new)
    t += h
    t_values.append(t)
    step += 1

    # Informaci n de progreso cada 10 pasos
    if step % 10 == 0:
        print(f"Paso {step}: t = {t:.2f}s, x = {x_new:.2f}m, y =
           {y_new:.2f}m")

print(f"Simulaci n completada: {step} pasos, tiempo final = {t
   :.2f}s")
print(f"Alcance num rico: {x[-1]:.2f} m")
```

```
# ================================================
# SOLUCI N ANAL TICA EXACTA (PARA COMPARACI N)
# ================================================

# La soluci n exacta para movimiento parab lico sin rozamiento:
# x(t) = v0*cos(  )*t
# y(t) = v0*sin(  )*t - 0.5*g*t
# Eliminando el tiempo: y(x) = x*tan(  ) - (g*x  )/(2*v0  *cos  (
    ))

print("\nCalculando soluci n anal tica...")

# Crear array de posiciones x para evaluar la trayectoria
   anal tica
x_analytic = np.linspace(0, max(x), 200)

# Calcular las y correspondientes usando la ecuaci n de la
   trayectoria
# y(x) = x*tan(  ) - (g*x  )/(2*v0  *cos  (  ))
tan_theta = np.tan(theta)
cos2_theta = np.cos(theta)**2
denominador = 2 * v0**2 * cos2_theta

y_analytic = x_analytic * tan_theta - (g * x_analytic**2) /
   denominador

# Tambi n calculamos el alcance te rico exacto
alcance_teorico = (v0**2 * np.sin(2*theta)) / g
altura_max_teorica = (v0**2 * np.sin(theta)**2) / (2*g)

print(f"Alcance te rico: {alcance_teorico:.2f} m")
print(f"Altura m xima te rica: {altura_max_teorica:.2f} m")

# ================================================
# VISUALIZACI N Y COMPARACI N
# ================================================

plt.figure(figsize=(10, 6))

# Graficar soluci n num rica (m todo de Euler)
plt.plot(x, y, 'bo-', markersize=3, linewidth=1, label=f'Euler (h
   ={h}s)')

# Graficar soluci n anal tica exacta
plt.plot(x_analytic, y_analytic, 'r-', linewidth=2, label='
   Soluci n anal tica')

# Configuraci n del gr fico
plt.xlabel("Distancia horizontal, x [m]", fontsize=12)
plt.ylabel("Altura, y [m]", fontsize=12)
```

```
plt.title("Trayectoria de Proyectil: M todo de Euler vs
    Soluci n Anal tica", fontsize=14)
plt.legend(fontsize=11)
plt.grid(True, alpha=0.3)
plt.axis('equal')  # Misma escala en ambos ejes para ver la forma
    real

# A adir informaci n adicional en el gr fico
plt.text(0.05, 0.95, f' v   = {v0} m/s,    = {np.degrees(theta)
    :.0f}  ',
         transform=plt.gca().transAxes, fontsize=10,
            verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha
            =0.8))

plt.tight_layout()
plt.show()

# ==================================================
# AN LISIS DEL ERROR
# ==================================================

print("\n" + "="*50)
print("AN LISIS DE PRECISI N")
print("="*50)

# Calcular error en el alcance
error_alcance = abs(x[-1] - alcance_teorico)
error_relativo = (error_alcance / alcance_teorico) * 100

print(f"Alcance num rico: {x[-1]:.4f} m")
print(f"Alcance te rico:  {alcance_teorico:.4f} m")
print(f"Error absoluto:   {error_alcance:.4f} m")
print(f"Error relativo:   {error_relativo:.2f}%")

# Sugerencia para mejorar la precisi n
if error_relativo > 1:
    print("\nRecomendaci n: Reducir el paso h para mayor
       precisi n")
    print(f"Pruebe con h = 0.01 o h = 0.001")
else:
    print("\n Buena precisi n! El m todo funciona bien para
       este paso h")
```

Análisis Detallado del Método

- **Precisión del método:** El método de Euler es de primer orden, lo que significa que el error es proporcional a h. Para pasos grandes ($h = 0.1$ s), el error acumulativo es significativo.
- **Estabilidad:** El método es condicionalmente estable. Para nuestro problema, la

estabilidad requiere que h sea suficientemente pequeño comparado con las escalas de tiempo características del sistema.

- **Convergencia:** Al reducir h ($h = 0.01$ s, $h = 0.001$ s), la solución numérica converge a la analítica, pero el costo computacional aumenta.
- **Ventajas:**
  - Simple implementación
  - Fácil de entender
  - Bajo costo computacional por paso
- **Desventajas:**
  - Baja precisión para pasos grandes
  - Error acumulativo
  - Puede volverse inestable para sistemas rígidos
- **Mejoras posibles:** Métodos de Runge-Kutta (especialmente RK4) ofrecen mayor precisión con el mismo paso, o la misma precisión con pasos mayores.