Tarea #4 Física numérica

Oscar Andrés Valencia Magaña

27 de octubre de 2025

Índice

Índice de figuras

Introducción

Buscamos presentar la solución a tres problemas físicos: el movimiento de un proyectil en un medio viscoso, un sistema de osciladores acoplados y una cuerda vibrante. En este trabajo se abordarán tanto las soluciones analíticas (si las hay, sino solo se deducirán las ecuaciones) como las soluciones numéricas y las gráficas obtenidas mediante programas desarrollados en Python.

1. Lanzamiento del martillo

Planteamiento:

El récord mundial para hombres en lanzamiento de martillo es de 86,74 m, establecido por Yuri Sedykh y vigente desde 1986. El martillo tiene una masa de 7,26 kg, es esférico y posee un radio de $R = 6$ cm.

La fricción sobre el martillo puede considerarse proporcional al cuadrado de la velocidad relativa al aire:

$$F_D = \frac{1}{2}\rho A C_D v^2$$

donde ρ es la densidad del aire (1,2 kg/m^3) y $A = \pi R^2$ es la sección transversal del martillo.

El martillo puede experimentar, en principio, un flujo laminar con coeficiente de rozamiento $C_D = 0{,}5$ o un flujo inestable oscilante con $C_D = 0{,}75$.

(a) Resuelva la ecuación de movimiento para el lanzamiento oblicuo del martillo. Deberá transformar las EDO correspondientes a los movimientos en x y y en un sistema de cuatro ecuaciones de primer orden. Considere lanzamientos desde una posición inicial $x_0 = 0$ y $y_0 = 2$ m, para un ángulo ideal $\theta = 45°$, y determine la velocidad que produce la distancia del lanzamiento del récord mundial.
Solución:

Consideremos $\vec{r} = (x, y)$ y $\dot{\vec{r}} = \vec{v} = (v_x, v_y)$. Según la mecánica newtoniana, la ecuación de movimiento está dada por:

$$m\vec{\ddot{r}} = m\vec{g} - \vec{F_D} = m\vec{g} - \frac{1}{2}\rho A C_D v^2 \hat{v} = m\vec{g} - \frac{1}{2}\rho A C_D v\,\vec{v},$$

donde $\vec{g} = (0, -g)$ y $v = |\vec{v}| = \sqrt{v_x^2 + v_y^2}$.
Desarrollando las componentes obtenemos:

$$m\dot{v}_x = -\frac{1}{2}\rho A C_D\, v\, v_x,$$

$$m\dot{v}_y = -mg - \frac{1}{2}\rho A C_D\, v\, v_y.$$

Por lo tanto, el sistema de ecuaciones diferenciales de primer orden queda expresado como:

$$\begin{aligned}
\dot{x} &= v_x, \\
\dot{y} &= v_y, \\
\dot{v}_x &= -\frac{1}{2m}\rho A C_D\, v\, v_x, \\
\dot{v}_y &= -g - \frac{1}{2m}\rho A C_D\, v\, v_y.
\end{aligned}$$

Con las condiciones iniciales:

$$\begin{aligned}
x(0) &= 0, \\
y(0) &= 2 \text{ m}, \\
v_x(0) &= v_0 \cos\theta, \\
v_y(0) &= v_0 \sin\theta,
\end{aligned}$$

donde v_0 es la velocidad inicial que se debe determinar para alcanzar la distancia del récord mundial, y $\theta = 45° = \frac{\pi}{4}$.
Este sistema es no lineal debido a la presencia del término $v = \sqrt{v_x^2 + v_y^2}$ en las ecuaciones para $\dot{v}_x$ y $\dot{v}_y$. Por lo tanto, no existe una solución analítica cerrada, y es necesario recurrir a métodos numéricos para resolverlo.
Proponemos utilizar la librería `scipy` para resolver el sistema de ecuaciones diferenciales, en particular, la función `odeint` para realizar la integración temporal del sistema.
La estrategia para encontrar la velocidad inicial v_0 que produce la distancia del récord mundial es la siguiente:

- Definir una función que resuelva el sistema de ecuaciones diferenciales para un valor dado de v_0 y que retorne la distancia horizontal del lanzamiento.
- Emplear un método de búsqueda de raíces (Newton-Raphson) para hallar el valor de v_0 que hace que la distancia obtenida sea igual a 86,74 m.

Implementando este procedimiento en `Python`, se obtiene que la velocidad inicial necesaria para alcanzar la distancia del récord mundial es aproximadamente:

$$\begin{aligned}
v_0 &\approx 28{,}84 \text{ m/s} && \text{(sin fricción, } C_D = 0), \\
v_0 &\approx 29{,}31 \text{ m/s} && \text{(flujo laminar, } C_D = 0{,}5), \\
v_0 &\approx 29{,}54 \text{ m/s} && \text{(flujo inestable oscilante, } C_D = 1{,}0).
\end{aligned}$$

El código utilizado para estos cálculos se presenta en los apéndices de esta tarea.

(b) Calcule y grafique la dependencia temporal de la altitud del martillo y su trayectoria $y = y(x)$ en los tres regímenes:
   i. Sin fricción

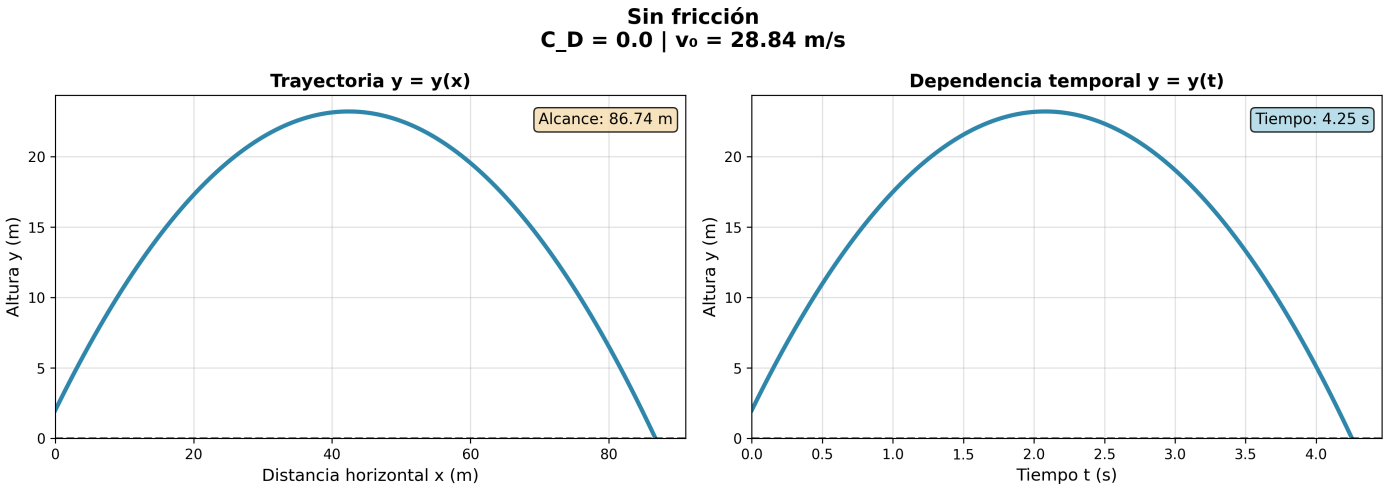

Figura 1: Trayectoria del martillo sin fricción

II. Flujo laminar

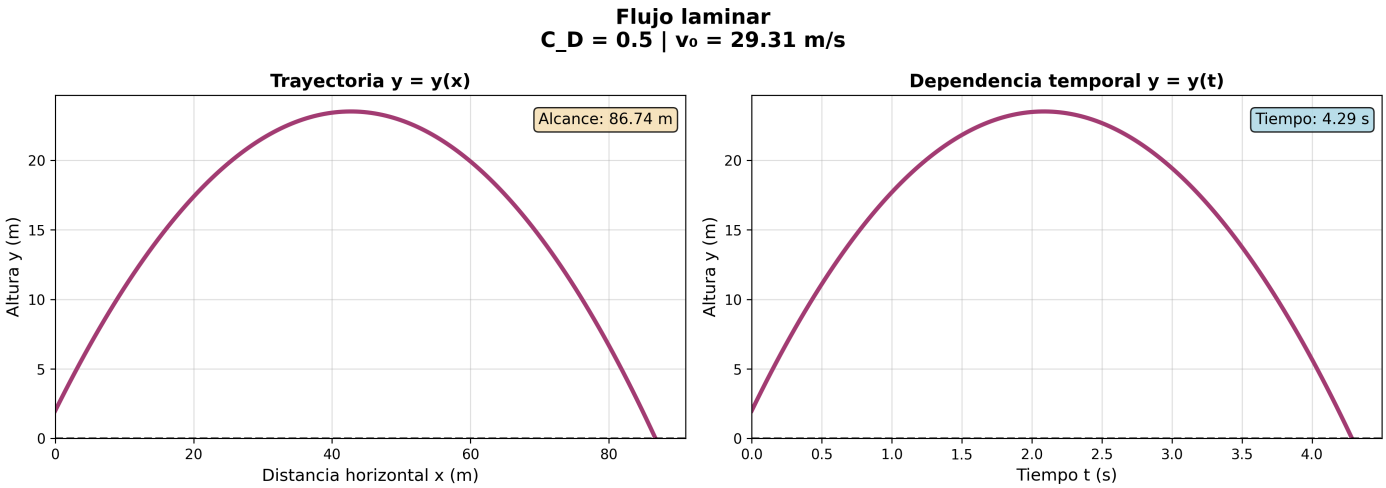

Figura 2: Trayectoria del martillo en flujo laminar

III. Flujo inestable oscilante

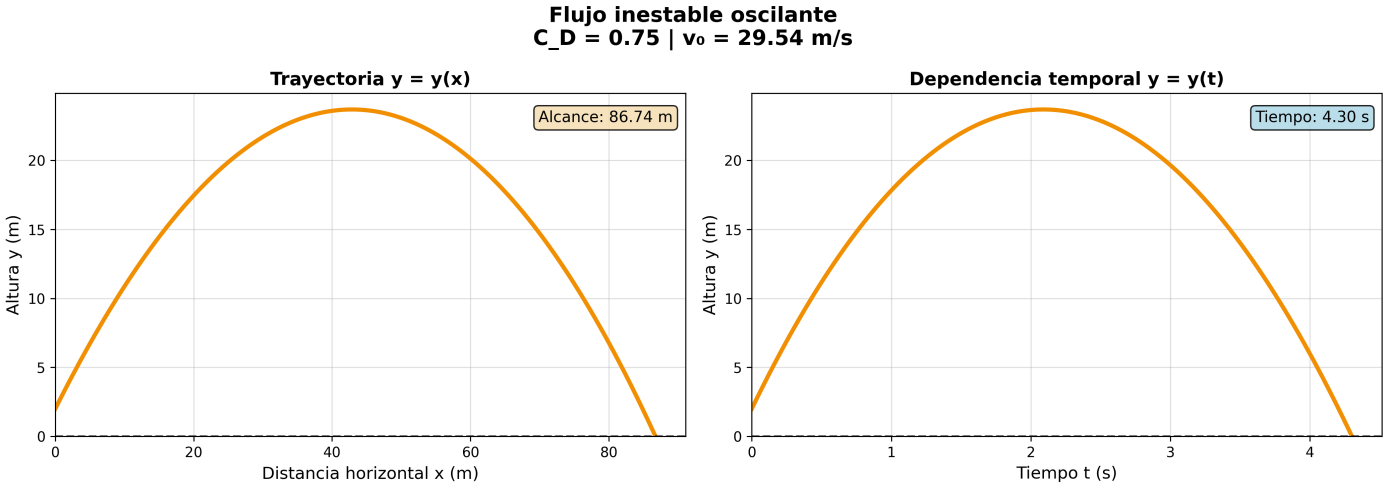

Figura 3: Trayectoria del martillo en flujo inestable oscilante

(c) En el inciso anterior, estime en qué medida la fricción influye en la distancia del lanzamiento.
Solución:
La fricción del aire tiene un impacto significativo en la distancia del lanzamiento del martillo. A continuación, se presenta un análisis detallado de la influencia de la fricción en los diferentes regímenes de flujo:

Régimen	Distancia (m)	Pérdida (m)	Pérdida (%)
Sin fricción	86.74	0.00	0.00 %
Flujo laminar	84.15	2.59	2.99 %
Flujo inestable oscilante	82.92	3.82	4.41 %

Cuadro 1: Influencia de la fricción en la distancia del lanzamiento del martillo

Como se observa en la Tabla 1, la fricción del aire reduce la distancia del lanzamiento en ambos regímenes considerados. En el caso del flujo laminar, la distancia se reduce en aproximadamente 2,59 m, lo que representa una pérdida del 2,99 % respecto al caso sin fricción. En el régimen de flujo inestable oscilante, la reducción es aún mayor, con una pérdida de 3,82 m o un 4,41 %. Esto indica que la fricción del aire tiene un efecto considerable en la trayectoria del martillo, especialmente en condiciones de flujo más turbulento.
Por lo que si resumimos nuestros resultados obtenidos tenemos que:

Régimen	Velocidad inicial (m/s)	Distancia (m)	Perdida (m)
Sin fricción	28.84	86.74	0.00
Flujo laminar	29.31	84.15	2.59
Flujo inestable oscilante	29.54	82.92	3.82

Cuadro 2: Velocidades iniciales y distancias alcanzadas en diferentes regímenes de fricción

De la Tabla 2, podemos concluir que: la fricción del aire reduce el alcance hasta en 4.4 %, lo que representa una pérdida máxima de 3.82 metros.

2. Oscilador armónico acoplado

Considere el sistema de resortes que se muestra en la figura 4.

Sea m la masa de cada bloque (ambas iguales) y supóngase que los resortes lineales tienen constantes elásticas k (resortes exteriores) y k_c (resorte central de acoplamiento), salvo que se indique lo contrario. Denote por $x_1(t)$ y $x_2(t)$ los desplazamientos horizontales de las masas respecto a sus posiciones de equilibrio.

(a) Escriba las ecuaciones de movimiento acopladas para los desplazamientos $x_1(t)$ y $x_2(t)$. Exprese las EDOs en su forma habitual y, a continuación, transforme el sistema a un conjunto equivalente de cuatro ecuaciones de primer orden adecuado para integración numérica.

(b) Calcule las frecuencias de los modos normales de vibración del sistema (modo simétrico y modo antisimétrico), y obtenga las correspondientes relaciones entre amplitudes X_1 y X_2 para cada modo.

(c) Grafique las posiciones de las masas en función del tiempo para las condiciones iniciales siguientes:

i. Ambas masas parten del reposo habiendo sido desplazadas la misma cantidad hacia la derecha: $x_1(0) = x_2(0) = A, \quad \dot{x}_1(0) = \dot{x}_2(0) = 0$.

ii. Ambas masas parten del reposo habiendo sido desplazadas la misma cantidad en sentidos opuestos: $x_1(0) = A,\ x_2(0) = -A, \quad \dot{x}_1(0) = \dot{x}_2(0) = 0$.

iii. Una masa parte de su posición de equilibrio y la otra de una posición desplazada hacia la derecha: $x_1(0) = 0,\ x_2(0) = A, \quad \dot{x}_1(0) = \dot{x}_2(0) = 0$.

Para cada caso, muestre las curvas $x_1(t)$ y $x_2(t)$ y, cuando sea útil, represente la combinación en coordenadas normales.

(d) Suponga ahora que los resortes no son lineales y que la fuerza restauradora de cada resorte tiene la forma

$$F = -k\big(x + 0{,}1\,x^3\big).$$

Repita el procedimiento del inciso (b): determine (o estime) las frecuencias / comportamientos de oscilación y compare las respuestas del sistema lineal con las del sistema no lineal. Discuta las diferencias cualitativas y cuantitativas entre ambos casos (desplazamiento- dependiente de la frecuencia, aparición de armónicos, etc.).

Figura 4: Diagrama del sistema de dos masas acopladas por resortes

3. Vibración de una cuerda

Oscilaciones de una cuerda

Considere una cuerda de longitud L y densidad lineal $\rho(x)$ (masa por unidad de longitud), sujeta en ambos extremos y bajo una tensión $T(x)$. Suponga que el desplazamiento transversal de la cuerda respecto a su posición de equilibrio, $y(x,t)$, es pequeño y que la pendiente $\partial y/\partial x$ también es pequeña.

(a) Considere una sección infinitesimal de la cuerda entre x y $x + \Delta x$. Notando que la diferencia en las componentes horizontales y verticales de las tensiones produce una fuerza restauradora, demuestre que, aplicando la segunda ley de Newton a esta sección, se obtiene la ecuación

$$\frac{dT(x)}{dx}\,\frac{\partial y(x,t)}{\partial x} + T(x)\,\frac{\partial^2 y(x,t)}{\partial x^2} \;=\; \rho(x)\,\frac{\partial^2 y(x,t)}{\partial t^2}.$$

(b) ¿Qué condiciones sobre $T(x)$ y $\rho(x)$ son necesarias para recuperar la ecuación de onda estándar

$$\frac{\partial^2 y(x,t)}{\partial x^2} \;=\; \frac{1}{c^2}\,\frac{\partial^2 y(x,t)}{\partial t^2}, \qquad c = \sqrt{\frac{T}{\rho}}\,?$$

Explique claramente las hipótesis de homogeneidad y constancia que se requieren.

(c) ¿Qué condiciones deben imponerse (condiciones iniciales y de frontera) para que la EDP de segundo orden tenga una única solución?

(d) Utilice una malla en tiempo y espacio con pasos Δt y Δx. Denote

$$y(x_i, t_j) = y_{i,j}, \qquad x_i = i\,\Delta x, \quad t_j = j\,\Delta t.$$

Escriba la aproximación finita correspondiente para obtener una solución numérica $y_{i,j}$.

(e) Exprese las segundas derivadas temporales y espaciales en términos de diferencias finitas centradas y demuestre que, para el caso homogéneo (constantes T y ρ), esto conduce a la ecuación en diferencias

$$y_{i,j+1} - 2y_{i,j} + y_{i,j-1} \;=\; \frac{c^2(\Delta t)^2}{(\Delta x)^2}\,\big(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\big),$$

donde $c = \sqrt{T/\rho}$.

(f) De la ecuación anterior, despeche $y_{i,j+1}$ y escriba el algoritmo de obtención del paso temporal siguiente en la forma

$$y_{i,j+1} \;=\; 2y_{i,j} - y_{i,j-1} \;+\; \lambda^2\big(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\big),$$

donde $\lambda = c\,\dfrac{\Delta t}{\Delta x}$ es el número de Courant reducido (velocidad de la malla $c_0 = \Delta x/\Delta t$ implica $\lambda = c/c_0$).

(g) ¿Cómo se implementan las condiciones iniciales y de frontera en el esquema numérico? Especifique la forma de:

- condiciones de frontera fijas (extremos sujetos: $y_{0,j} = y_{N,j} = 0$),
- condiciones de frontera libres o de radiación (si procede),
- condiciones iniciales $y_{i,0}$ y $\dot{y}_{i,0}$ (desplazamiento y velocidad inicial).

Indique además cómo calcular $y_{i,1}$ (el primer paso en tiempo) a partir de $y_{i,0}$ y $\dot{y}_{i,0}$.

(h) La condición de Courant para la estabilidad del esquema explícito es

$$\lambda = \frac{c\,\Delta t}{\Delta x} \le 1.$$

Explique qué significa esto en términos de los pasos Δt y Δx (interpretación física y numérica).

(i) Escriba un programa (por ejemplo en Python) que implemente el esquema explicito anterior y produzca una animación del movimiento de la cuerda $y(x,t)$. Indique las decisiones prácticas importantes (elección de Δx, Δt, duración de la simulación, condiciones de frontera, normalización de ejes para la animación).

(j) Varíe los pasos Δt y Δx en su programa de modo que en algunos casos se cumpla la condición de Courant y en otros no. Describa y explique qué ocurre en cada caso (estabilidad numérica, propagación correcta de ondas cuando $\lambda \le 1$; crecimiento no físico e inestabilidad cuando $\lambda > 1$).

A. Código para el lanzamiento del martillo

Tarea 4: Resolución del problema del martillo

```
from pylab import *
import numpy as np
from scipy.integrate import odeint
from matplotlib.animation import FuncAnimation
import os

# Parámetros físicos
g = 9.81  # Aceleración debido a la gravedad (m/s^2)
rho = 1.2  # Densidad del aire (kg/m^3)
```

```
R = 0.06  # Radio del martillo (m)
A = np.pi * R**2  # Área de sección transversal del martillo (m^2)
m = 7.26  # Masa del martillo (kg)
record_distance = 86.74  # Distancia del récord mundial (m)
theta = np.radians(45)  # Ángulo de lanzamiento (radianes)
x0, y0 = 0,  2  # Posición inicial (m)
it_max = 500  # Número máximo de iteraciones
dt = 0.01  # Paso de tiempo (s)
N= 500  # Número de pasos de tiempo
tol=1e-3  # Tolerancia para la convergencia
drag_coeffs = [0.0, 0.5, 0.75]  # Coeficientes de arrastre para los tres regímenes

# Crear carpeta para resultados
output_dir = 'resultados_martillo' # Carpeta para guardar resultados
if not os.path.exists(output_dir): # Crear carpeta si no existe
    os.makedirs(output_dir) # Crear carpeta si no existe


#definición de las ecuaciones de movimiento (EDOs)
def equations_of_motion(state, t, k):
    """Devuelve las derivadas de las variables de estado."""
    f0= state[1]
    f1= -k/m*state[1]*np.sqrt(state[1]**2 + state[3]**2)
    f2= state[3]
    f3 = -g - k/m*state[3]*np.sqrt(state[1]**2 + state[3]**2)
    return array([f0, f1, f2, f3])

#buscamos la distancia alcanzada para una velocidad inicial dada
def distance_reached(initial, v0, k):
    """Calcula la distancia alcanzada para una velocidad inicial dada y coeficiente de arrastre
    ↪ k."""
    #como theta = 45 grados, las componentes x e y de la velocidad inicial son iguales
    v= v0 * np.sin(pi/4)
    finaltime = 10.0 # Tiempo final para la simulación
    r0= array([initial[0],v , initial[1], v])  # Estado inicial: [x0, vx0, y0, vy0]
    r=r0 # variable para almacenar el estado actual
    t = linspace(0, it_max * dt, N)  # Vector de tiempo
    groundtime= 0.0

    #inicia lo dificil, jugar a adivinar con algo numerico
    s=0
    while s < it_max:
        # Integración numérica de las EDOs
        sol = odeint(equations_of_motion, r, t, args=(k,))
        n=len(sol)-1
        # Verificar si el martillo no ha tocado el suelo
        if sol[n, 2] > 0:
            finaltime += 1.0
            t= linspace(0., finaltime, N)
        else:
           for j in range(n):
               # verificamos si la fisica nos falla y fuimos capaces de atravesar el suelo
               if sol[j, 2] <= 0:
                   groundtime += t[j-1]-t[0]
                   #checamos la tolerancia
                   if abs(sol[j,2])<= tol/2.:
                       # buscamos que en caso de ser preciso, reacemos todo pero usando el tiempo
                       ↪  para que toque el suelo.
                       t = linspace(0, groundtime+t[j]-t[j-1], N*50*s)
                       sol = odeint(equations_of_motion, r0, t, args=(k,))
                       n=len(sol)-1
                       return (sol[n,0]+sol[n-1,0])/2., groundtime
                   else: #en caso de no ser preciso, resolvemos la ED con los valores inciales,
                    ↪  que seran las condiciones que el martillo llevaba antes de tocar el piso.
```

```
                        r = array([sol[j-1][0], sol[j-1][1],
                                   sol[j-1][2], sol[j-1][3]])
                        t=linspace(t[j-1], t[j], 50)
                        break
        s += 1
    print("Distancia recorrida antes de llegar al suelo no encontrada\
          dentro de las iteraciones permitidas. Regresamos 0.")
    return 0.,0.


# nos interesa hallar la velocidad inicial que produce la distancia del récord mundial
def find_initial_velocity(distance, initial, v0, k):
    """Encuentra la velocidad inicial que produce la distancia del récord mundial."""
    f= lambda v: distance_reached(initial, v, k)[0]- distance
    dv= 1.0e-3

    i=0

    #vamos a hacer Newton-Raphson para encontrar la velocidad inicial
    while i < it_max:
        fv = f(v0)
        if abs(fv) <= tol:
            return v0
        # Derivada numérica
        df = (f(v0 + dv/2.) - f(v0 - dv/2.)) / dv
        dv= -fv/df
        v0 += dv

        i += 1
    print("Velocidad inicial no encontrada dentro de las iteraciones permitidas.\
          Regresamos 0.")
    return 0.
#Tambien nos interesa la trayectoria :
def get_full_trajectory(initial, v0, k, tground):
    """Obtiene la trayectoria completa para graficar."""
    v = v0 * np.sin(np.pi/4)
    r0 = array([initial[0], v, initial[1], v])

    if tground == 0.:
        tground = 5.0

    t = linspace(0., tground, 500)
    sol = odeint(equations_of_motion, r0, t, args=(k,))

    return sol, t
#imprimmos las condiciones iniciales
print("\nCondiciones iniciales:")
print(f"  x0 = {x0} m, y0 = {y0} m, theta = 45")
print("\nEncontrando velocidad inicial para alcanzar récord mundial...")
print("="*70)

v_aux = [] # Lista para almacenar velocidades iniciales
for i, cd in enumerate(drag_coeffs):
    print(f"  * Régimen {i+1} (C_D = {cd}):")
    k = rho * A * cd / 2.0
    v0_record = find_initial_velocity(record_distance, array([x0, y0]), 28.0, k)
    v_aux.append(v0_record) # Agregar velocidad inicial encontrada a la lista --- nos serviran
    ↪ más adelante
    if v0_record == 0.:
        print("ERROR: No se pudo calcular la velocidad inicial.")
        exit(1)
    else:
        print(f"Velocidad inicial necesaria: v0 = {v0_record:.2f} m/s")
        print(f"(Para alcanzar {record_distance} m con C_D = {cd})")
```

```
        print("="*70)

print("\n" + "="*70)
print("TRAYECTORIAS EN LOS TRES REGÍMENES")
print("="*70)

solutions = [] # Almacenar soluciones para cada régimen
times_list = [] # Almacenar tiempos para cada régimen
distances_list = [] # Almacenar distancias alcanzadas
labels = ['Sin fricción', 'Flujo laminar', 'Flujo inestable oscilante'] # Etiquetas para los
↪  regímenes
cd_labels = ['C_D = 0.0', 'C_D = 0.5', 'C_D = 0.75'] # Etiquetas para los coeficientes de
↪  arrastre
colors = ['#2E86AB', '#A23B72', '#F18F01'] # Colores para las gráficas

print("\nCalculando trayectorias...")
for i, cd in enumerate(drag_coeffs): # Iterar sobre los coeficientes de arrastre
    k = rho * A * cd / 2.0
    distance, time = distance_reached(array([x0, y0]), v_aux[i], k) # Calcular distancia
    ↪  alcanzada
    distances_list.append(distance)     # Almacenar distancia alcanzada

    print(f"\n{labels[i]} ({cd_labels[i]}):")
    print(f"  Distancia alcanzada: {distance:.2f} m") # Almacenar distancia alcanzada
    print(f"  Tiempo de vuelo: {time:.2f} s") # Almacenar tiempo de vuelo
    print(f"  Velocidad inicial: v0 = {v_aux[i]:.2f} m/s")

    sol, t = get_full_trajectory(array([x0, y0]), v_aux[i], k, time) # Obtener trayectoria
    ↪  completa
    solutions.append(sol) # Almacenar solución
    times_list.append(t) # Almacenar tiempos
print("="*70)
print("Calculo de las trayectorias con la velocidad inicial dada con C_D = 0.0: v0 =",v_aux[0])
print("\nCalculando trayectorias...")
sols=[] # Almacenar soluciones para cada régimen
tims=[] # Almacenar tiempos para cada régimen
dists=[] # Almacenar distancias alcanzadas
for i, cd in enumerate(drag_coeffs): # Iterar sobre los coeficientes de arrastre
    k = rho * A * cd / 2.0
    distance, time = distance_reached(array([x0, y0]), v_aux[0], k) # Calcular distancia
    ↪  alcanzada
    dists.append(distance)     # Almacenar distancia alcanzada

    print(f"\n{labels[i]} ({cd_labels[i]}):")
    print(f"  Distancia alcanzada: {distance:.2f} m") # Almacenar distancia alcanzada
    print(f"  Tiempo de vuelo: {time:.2f} s") # Almacenar tiempo de vuelo

    sol, t = get_full_trajectory(array([x0, y0]), v_aux[0], k, time) # Obtener trayectoria
    ↪  completa
    sols.append(sol) # Almacenar solución
    tims.append(t) # Almacenar tiempos
print("="*70)
print(" GENERANDO GRÁFICAS POR RÉGIMEN...")
print("="*70)

for i, (sol, t, cd, label, cd_label, color, distance) in enumerate(
    zip(solutions, times_list, drag_coeffs, labels, cd_labels, colors, distances_list)):

    # Crear figura con 2 subplots
    fig, (ax1, ax2) = subplots(1, 2, figsize=(14, 5))
    fig.suptitle(f'{label}\n{cd_label} | v₀ = {v_aux[i]:.2f} m/s',
                 fontsize=15, fontweight='bold')

    # Subplot 1: Trayectoria y = y(x)
```

```
    ax1.plot(sol[:, 0], sol[:, 2], color=color, linewidth=3, label='Trayectoria')
    ax1.axhline(y=0, color='black', linewidth=1.5, linestyle='--', alpha=0.7)
    ax1.set_xlabel('Distancia horizontal x (m)', fontsize=12)
    ax1.set_ylabel('Altura y (m)', fontsize=12)
    ax1.set_title('Trayectoria y = y(x)', fontsize=13, fontweight='bold')
    ax1.text(0.98, 0.95, f'Alcance: {distance:.2f} m',
             transform=ax1.transAxes, fontsize=11,
             verticalalignment='top', horizontalalignment='right',
             bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.85))
    ax1.grid(True, alpha=0.4)
    ax1.set_xlim(left=0)
    ax1.set_ylim(bottom=0)

    # Subplot 2: Altura vs tiempo y = y(t)
    ax2.plot(t, sol[:, 2], color=color, linewidth=3, label='Altura')
    ax2.axhline(y=0, color='black', linewidth=1.5, linestyle='--', alpha=0.7)
    ax2.set_xlabel('Tiempo t (s)', fontsize=12)
    ax2.set_ylabel('Altura y (m)', fontsize=12)
    ax2.set_title('Dependencia temporal y = y(t)', fontsize=13, fontweight='bold')
    ax2.text(0.98, 0.95, f'Tiempo: {t[-1]:.2f} s',
             transform=ax2.transAxes, fontsize=11,
             verticalalignment='top', horizontalalignment='right',
             bbox=dict(boxstyle='round', facecolor='lightblue', alpha=0.85))
    ax2.grid(True, alpha=0.4)
    ax2.set_xlim(left=0)
    ax2.set_ylim(bottom=0)

    tight_layout()
    filename = f'{output_dir}/regimen_{i+1}_CD_{cd:.2f}.png'
    savefig(filename, dpi=300, bbox_inches='tight')
    print(f"Guardado: {filename}")
    show()
    close(fig)

print("\n" + "="*70)
print("INFLUENCIA DE LA FRICCIÓN")
print("="*70)

print("\n" + "-"*70)
print(f"{'Régimen':<35} {'Distancia (m)':<15} {'Pérdida (m)':<15} {'Pérdida (%)'}")
print("-"*70)

losses_m = []
losses_pct = []

for cd, dist, label in zip(drag_coeffs, dists, labels):
    if dist > 0:
        loss_m = record_distance - dist
        loss_pct = (loss_m / record_distance) * 100
        losses_m.append(loss_m)
        losses_pct.append(loss_pct)

        print(f"{label:<35} {dist:<15.2f} {loss_m:<15.2f} {loss_pct:>6.2f}%")
    else:
        losses_m.append(0)
        losses_pct.append(0)
        print(f"{label:<35} {'N/A':<15} {'N/A':<15} {'N/A'}")

print("-"*70)

# Resumen final
print("\n" + "="*70)
print(" RESUMEN DE RESULTADOS")
print("="*70)
```

```
print(f"\nVelocidad inicial (sin fricción): v0 = {v0_record:.2f} m/s")
print(f"Distancia récord objetivo: {record_distance:.2f} m\n")

if losses_m[1] > 0:
    print(f"Flujo laminar (C_D = 0.5):")
    print(f"  ->Reduce {losses_m[1]:.2f} m ({losses_pct[1]:.1f}%)")
    print(f"  -> Alcance: {distances_list[1]:.2f} m\n")

if losses_m[2] > 0:
    print(f"Flujo inestable oscilante (C_D = 0.75):")
    print(f"  -> Reduce {losses_m[2]:.2f} m ({losses_pct[2]:.1f}%)")
    print(f"  -> Alcance: {distances_list[2]:.2f} m\n")

print("CONCLUSIÓN:")
print(f"La fricción del aire reduce el alcance hasta en {max(losses_pct):.1f}%,")
print(f"lo que representa una pérdida máxima de {max(losses_m):.2f} metros.")

print("\n" + "="*70)
print(f" PROCESO COMPLETADO")
print("="*70)
print(f"\nArchivos generados en '{output_dir}/':")
print("  * regimen_1_CD_0.00.png - Sin fricción")
print("  * regimen_2_CD_0.50.png - Flujo laminar")
print("  * regimen_3_CD_0.75.png - Flujo inestable oscilante")
print("="*70 + "\n")
```

B. Código para el oscilador acoplado

C. Código para la vibración de una cuerda

D. Consideraciones a futuro