ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS
ARQUITECTURA DE COMPUTADORES

Taller Script Python

Ricardo León

2025-01-19

1. Objetivos

- Desarrollar un script de Python con paradigma estructurado

2. Problema

Usted es el científico de datos para un sistema de lanzamiento de cohetes. Para esto, se le solicita que determine gráficamente el alcance del cohete según al menos tres ángulos de entrada, un tiempo máximo de vuelo `t`. Se considera que existe una fuerza de resistencia al movimiento debido al viento y representada mediante un coeficiente `b`.

Escribir un programa que permita obtener el alcance del cohete gráficamente, mediante la resolución de ecuaciones diferenciales ordinarias. Los parámetros como Velocidad inicial `V`, coeficiente de resistencia `B`, tiempo máximo `tmax` y ángulos de lanzamiento se ingresan desde la línea de comando. Siga las instrucciones proporcionadas en la Sección 3 para completar el taller.

3. Instrucciones

1. Obtener las expresiones matemáticas para las ecuaciones diferenciales ordinarias. Vea la Sección 5.1
2. Investigar cómo funciona la librería `solve_ivp` del paquete `scipy.integrate`
3. Investigar cómo funciona la librearía `argparse` para controlar el ingreso de valores y opciones desde la línea de comandos.
4. Cree una carpeta con los archivos del proyecto denominada `cohete`.
5. Dentro de la carpeta cree un archivo `main.py` con el código proporcionado en la Sección 5.2
6. Reorganizar el código proporcionado de ejemplo para conformar un proyecto de código estructurado. Las funciones estarán dentro de una carpeta `utils` en el archivo `projectile_functions.py`. El código principal se coloca en el archivo `main.py` conforme al siguiente esquema de archivos.
   a) Con YASnippet habilitado escriba `imp` seguido de `C-TAB` para escribir `import`
   b) Escriba `main` seguido de la secuencia de comandos `C-TAB`. Esto genera una función llamada `main`
   c) escriba `ifm` seguido de la secuencia `C-TAB` para generar la condición de ejecución del programa Python como programa principal
   d) Use `fd` seguido de `C-TAB` para escribir una función con *docstrings* y use `r` seguido de `C-TAB` para escribir `return`
   e) Utilice `pars C-TAB` para crear un parser y `args C-TAB` para crear un argumento de entrada del programa. Importe la librería `argparse`

```
/
├── main.py
└── utils
    ├── __init__.py
    └── projectile_functions.py
```

4. Entregables del taller

1. Archivo .zip con el archivo `main.py` y la librería desarrollada `projectile_functions.py` y el gráfico obtenido de una carrera.
2. Completar en la sección [[]] de este documento una breve descripción de lo que hacen las librerías revisadas en este taller.
3. Una vez redactado el punto anterior suba el archivo .ORG y el .PDF

5. Recursos

Puede consultar más información en el siguiente vídeo y en GitHub

5.1. Desarrollo Matemático

La fuerza de resistencia del viento se puede considerar que es proporcional al cuadrado de la velocidad y en sentido opuesto a ésta. De ahí que las fuerzas actuando en el sistema quedarían determinadas por la ecuación

$$\vec{F}_{\text{net}} = \vec{F}_g + \vec{F}_f = -mg\hat{y} - b|\vec{v}|\vec{v} \tag{1}$$

La velocidad está dada por $\vec{v} = \dot{x}\hat{i} + \dot{y}\hat{j}$
En consecuencia la Fuerza neta del sistema queda:

$$\vec{F}_{\text{net}} = -mg\hat{y} - b\sqrt{\dot{x}^2 + \dot{y}^2}(\dot{x}\hat{i} + \dot{y}\hat{j}) \tag{2}$$

Que en forma vectorial se puede escribir como

$$\vec{F}_{\text{net}} = \begin{bmatrix} -b\sqrt{\dot{x}^2 + \dot{y}^2}\dot{x} \\ -mg - b\sqrt{\dot{x}^2 + \dot{y}^2}\dot{y} \end{bmatrix} \tag{3}$$

Dado que $\vec{F}_{\text{net}} = m\vec{a} = m\langle \ddot{x}, \ddot{y} \rangle$, se tiene

$$m\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} -b\sqrt{\dot{x}^2 + \dot{y}^2}\dot{x} \\ -mg - b\sqrt{\dot{x}^2 + \dot{y}^2}\dot{y} \end{bmatrix} \tag{4}$$

De donde se obtienen el par de ecuaciones diferenciales

$$\ddot{x} = -\frac{b}{m}\sqrt{\dot{x}^2 + \dot{y}^2}\dot{x} \tag{5}$$

$$\ddot{y} = -g - \frac{b}{m}\sqrt{\dot{x}^2 + \dot{y}^2}\dot{y} \tag{6}$$

Definiendo $x' = x/g$ and $y' = y/g$, se tiene entonces

$$\ddot{x'} = -\frac{bg}{m}\sqrt{\dot{x'}^2 + \dot{y'}^2}\dot{x'} \tag{7}$$

$$\ddot{y'} = -1 - \frac{bg}{m}\sqrt{\dot{x'}^2 + \dot{y'}^2}\dot{y'} \tag{8}$$

Sea $B \equiv bg/m$, entonces considerando el cambio de notación y eliminado la ' para simplificar, se propone resolver un sistema de ecuaciones de primer orden ODE siguiente:

$$\dot{x} = v_x \tag{9}$$
$$\dot{v_x} = -B\sqrt{\dot{x}^2 + \dot{y}^2}\dot{x} \tag{10}$$
$$\dot{y} = v_y \tag{11}$$
$$\dot{v_y} = -B\sqrt{\dot{x}^2 + \dot{y}^2}\dot{y} \tag{12}$$

5.2. Código Inicial Python

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import argparse
from utils.projectile_functions import dSdt

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-V', type=float, default=1.0)
    parser.add_argument('-B', type=float, default=0.0)
    parser.add_argument('-tmax', type=float, default=2.0)
    args = parser.parse_args()

    V = args.V
    B = args.B
    t_max = args.tmax

    t_eval = np.linspace(0, t_max, 1000)
    angulos = [40, 45, 50]

    for ang in angulos:
        rad = ang * np.pi / 180
        y0 = [0, V*np.cos(rad), 0, V*np.sin(rad)]
        sol = solve_ivp(dSdt, [0, t_max], y0=y0, t_eval=t_eval, args=(B,))
        plt.plot(sol.y[0], sol.y[2], label=f'$\\theta_0={ang}^\\circ$')

    plt.ylim(bottom=0)
    plt.legend()
    plt.xlabel('$x/g$', fontsize=20)
    plt.ylabel('$y/g$', fontsize=20)
    plt.show()

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

6. Descripción librerías usadas

6.1. $\text{SOLVE}_{\text{IVP}}$

Es una función perteneciente a la librería scipy.integrate. Se utiliza para resolver numéricamente sistemas de ecuaciones diferenciales ordinarias (EDO) con condiciones iniciales. En este proyecto, se encarga de calcular la posición (x, y) y la velocidad (v_x, v_y) del cohete en cada instante de tiempo, considerando la gravedad y la resistencia del aire.

6.2. ARGPARSE

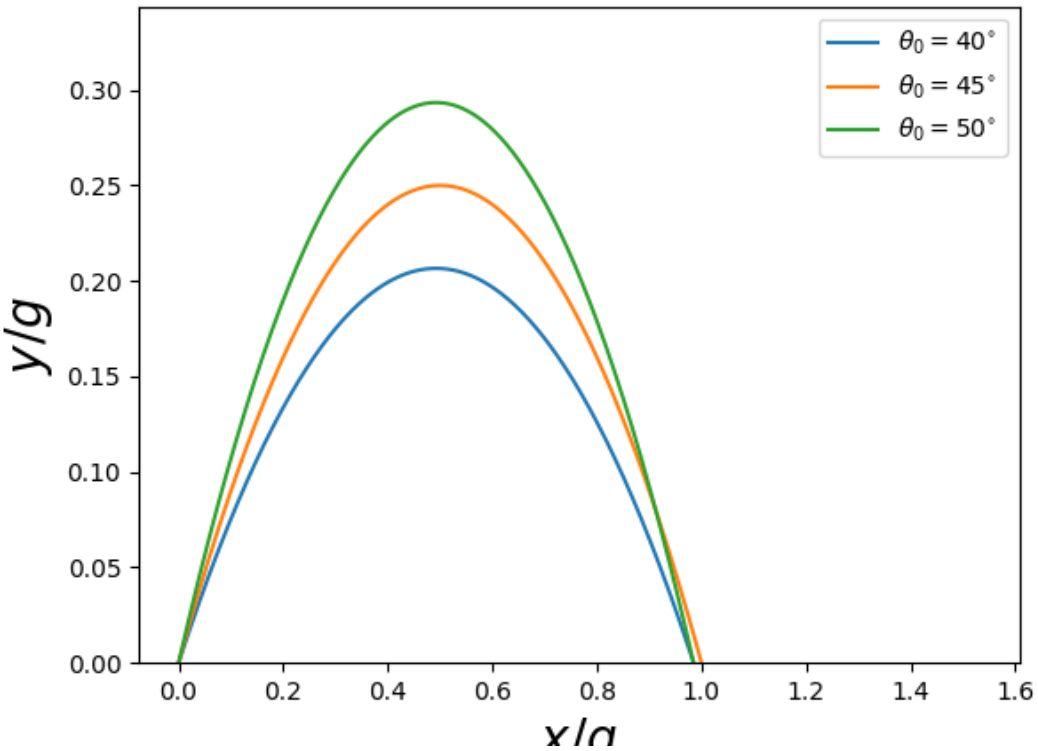
Es una librería estándar de Python diseñada para facilitar la creación de interfaces de línea de comandos. Permite que el usuario defina parámetros externos (como la velocidad inicial -V o el coeficiente de resistencia -B) al momento de ejecutar el script, evitando la necesidad de modificar el código fuente manualmente para realizar diferentes simulaciones.

6.3. NUMPY

Es la biblioteca fundamental para la computación científica en Python. Proporciona soporte para vectores y matrices de gran tamaño, además de funciones matemáticas de alta precisión como np.sin(), np.cos(), np.sqrt() y la constante np.pi, esenciales para descomponer las velocidades iniciales y calcular la magnitud de la velocidad del proyectil.

7. 7. Resultados Gráficos

7.1. Trayectoria sin resistencia (B=0)



7.2. Trayectoria con resistencia (B=0.1)