Laboratorio Teoría de Modelos y simulación de sistemas

Sesión 2. Introducción Simulación con Python

Hay dos formas diferentes de modelar sistemas: (1) mediante las ecuaciones diferenciales y (2) por medio de la función de transferencia.

Método 1

La biblioteca `scipy.integrate` de SciPy ofrece herramientas para la integración numérica y la solución de ecuaciones diferenciales que representan modelos. La función más usada de la librería es `solve_ivp` que permite resolver ecuaciones diferenciales de la forma

$$dy/dt = f(t, y)$$

Se recomienda ver la documentación antes de continuar con el ejemplo: Clic aquí

Ejemplo práctico

Se desea modelar el movimiento de una masa sobre una superficie rugusa, sobre la cual se aplica una fuerza. El sistema a modelar posee una entrada `u` (Fuerza aplicada) y una salida `x` que representa la posición de la masa en un tiempo `t`. El modelo del sistema dinámico se puede expresar mediante las ecuaciones de Newton

$$m\ddot{x} + c\dot{x} = F$$

Donde:

- m: masa del cuerpo (1kg)
- c: coeficiente de fricción del cuerpo sobre la superficie (0.8)
- F: fuerza aplicada (0.1N aplicados a los 5 segundos)

Se desea hacer la simulación por un tiempo total de 15 segundos, considerando que la velocidad inicial es -1 m/s

Solución paso a paso

1. Reescribir la ecuación como sistema de primer orden

Escribir las ecuaciones en forma de primer orden. Si el modelo a evaluar ya lo es, se omite este paso.

Se define

- $x_1 = x$ (posición)
- $x_2 = \dot{x}$ (velocidad)

Entonces, derivando cada una de ellas, se obtiene el sistema de ecuaciones equivalente

- $\dot{x}_1 = x_2$ -> por definición
- $\dot{x}_2 = \frac{1}{m}F - \frac{c}{m}x_2$ -> despejando de la ecuación

2. Crear la función que define las ecuaciones a resolver

De acuerdo con la documentación de `solve_ivp`, es necesario crear una función que calcule las derivadas de todas las variables de estado, en este caso x_1 y x_2. Siempre debe tener como primer parámetro la variable independiente, `t` en este caso, y como segundo argumento un vector con la misma cantidad de elementos como ecuaciones equivalentes. Los demás argumentos son opcionales y se pueden manejar según las necesidades. En este caso, se agregarán las constantes (`m`, `c`).

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
```

In [17]:
```
# Función que define las ecuaciones diferenciales a resolver
def sys (t, y, m, c):

    # Creación de la fuerza de entrada
    F = np.where(t >= 5, 0.1, 0)  # Consultar documentación

    # Descripción de las ecuaciones optenidas en el numeral 1
    x1, x2 = y      # y[0] = x (posición), y[1] = dx/dt (velocidad)
    d_x1 = x2
    d_x2 = (1/m) * F - (c/m) * x2
    return [d_x1, d_x2]
```

3. Realizar la simulación

Acá es donde toma papel la función `solve_ivp`.

`solve_ivp(fun, t_span, y0, method='RK45', t_eval=None,  args=None)`

Parámetros de entrada principales

- `fun` : Función creada en el numeral anterior
- `t_span` : Intervalo de integración para resolver las ecuaciones diferenciales (t0, tf). En este caso t0 = 0 y tf = 15 s.
- `y0` : Vector con condiciones iniciales. En este caso la posición inicial es 0 m y la velocidad inicial es -1m/s.
- `method` : Parámetro opcional que indica el método de integración utilizado. Por defecto es el método de Runge-Kutta (RK45)
- `t_eval` : Parmámetro opcional. Tiempo en el que se almacenará la solución, por lo que debe estar dentro del rango `t_span`. Si no se agrega, se utilizará una cantidad de puntos seleccionados por el solucionador.

- `args` : Parámetro opcional. Representa los argumentos adicionales para pasarle a la función definida en `fun` .

In [18]:
```
# Definición de constantes
m = 1
c = 0.8

y0 = [0 , -1]  # Condiciones iniciales

Tm = 0.1 # Asumido para este problema
total_time = 15

t_span = (0,total_time)

time = np.arange(0, total_time + Tm, Tm)


sol = solve_ivp(sys, t_span, y0, t_eval=time, args=(m,c))
```

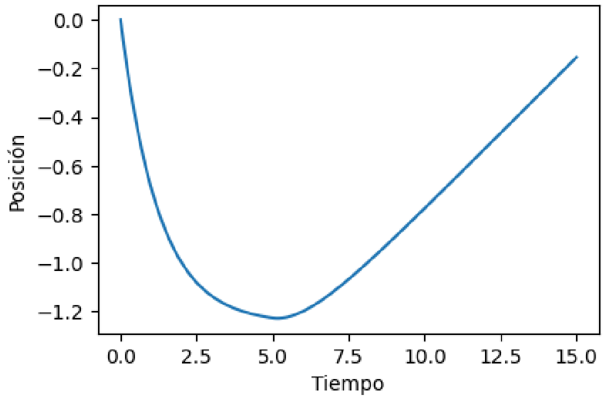
Parámetros de salida: En el objeto `sol` se encuentra `t` que son los puntos temporales (n_puntos,) y `y` que representa los valor de la solución en un tiempo t y tiene la forma (n, n_puntos).

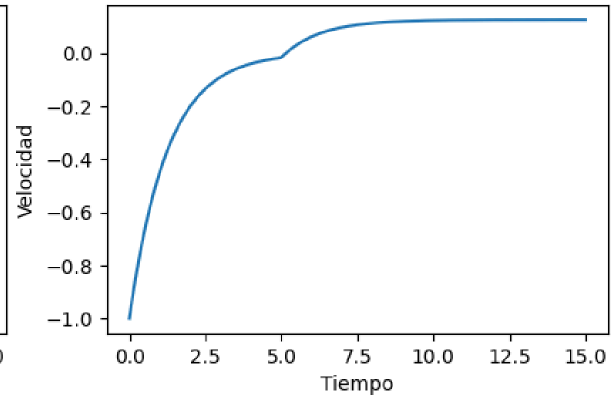
Así, en el ejemplo:

- El tiempo está en `sol.t`
- La posición está en `sol.y[0]`
- La velocidad está en `sol.y[1]`

In [19]:
```
# Gráfica de la solución
plt.figure(figsize=(10, 3))
plt.subplot(1, 2, 1)
plt.plot(sol.t, sol.y[0])

plt.xlabel('Tiempo')
plt.ylabel('Posición')

plt.subplot(1, 2, 2)
plt.plot(sol.t, sol.y[1])
plt.xlabel('Tiempo')
plt.ylabel('Velocidad')
```

Out[19]:
```
Text(0, 0.5, 'Velocidad')
```





Método 2

Para definir el sistema mediante su función de transferencia se puede utilizar la biblioteca `scipy.signal` con las funciones `TransferFunction` y `lsim`.

Se recomienda ver la documentación antes de continuar con el ejemplo: Clic aquí para TransferFunction y Clic aquí para lsim

Ejemplo práctico

Resolver el ejemplo de la sección anterior mediante el uso de la función de transferencia del sistema (salida/entrada) asumiendo que no hay condiciones iniciales.

Solución paso a paso

1. Obtener la función de transferencia

En el dominio temporal la expresión es: $m\ddot{x} + c\dot{x} = F$

Equivalente en el dominio de Laplace: $ms^2X(s) + csX(s) = F(s)$

Función de transferencia (Salida/Entrada):

$\frac{X(S)}{F(s)} = \frac{1}{ms^2+cs}$

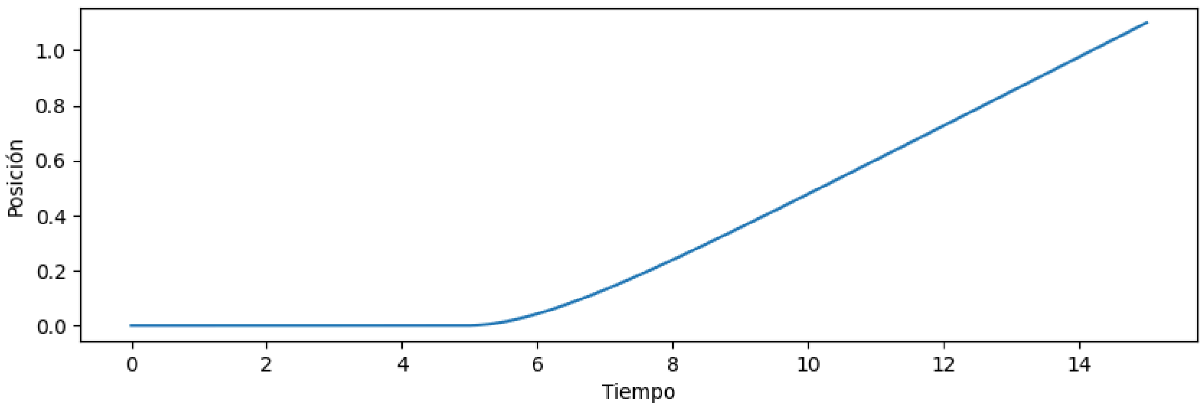
2. Realizar la simulación

In [20]:
```
import scipy.signal as signal

# Coeficientes del numerador y denominador
# de la fn de transferencia
num = [1]
den = [m, c, 0]

# Entrada
F = np.where(time >= 5, 0.1, 0)

# Creación del sistema
sys = signal.TransferFunction(num, den)
```

```
t_out, y_out, _ = signal.lsim(sys, F, T=time)
```

In [21]:
```
# Gráfica de la solución
plt.figure(figsize=(10, 3))
plt.plot(t_out, y_out)
plt.xlabel('Tiempo')
plt.ylabel('Posición')
```

Out[21]:
```
Text(0, 0.5, 'Posición')
```



Actividad

1. Resolver el ejercicio usando el primer método considerando todas las condiciones inciales iguales a cero. Compare en una sola gráfica ambas soluciones y concluya.