Control de Presión en un sistema tanque pulmón

CURVA DE REACCIÓN TANQUE PULMÓN

```
# 1. Importar todas las librerias y funciones que se necesitan
# ------------------------------------------------------------------------------
import numpy as np               # Manejo de vectores
import matplotlib.pyplot as plt  # Permite graficar
from google.colab import drive   # Permite montar mi Google Drive y leer datos
desde ahí
from matplotlib.ticker import MultipleLocator
#drive.mount("/content/drive")

# ------------------------------------------------------------------------------
# 2. Parámetros
# ------------------------------------------------------------------------------

# Leer el texto con las columnas sepradas por espacios con los datos
experimentales
data = np.genfromtxt('/content/drive/MyDrive/Semestre 2025-1S/Introduccio´n al
control de procesos biolo´gicos/Co´digos/Datos manuales_tanque_pulmon.txt')
# Vectores columna extraídos de data
t_e = data[:,0]  # Vector de tiempo experimental
u_e = data[:,1]  # Vector de u experimental, apertura de la válvula de entrada
y_e = data[:,2]  # Vector de y experimental, presión

# ------------------------------------------------------------------------------
# 3. Graficar datos experimentales
# ------------------------------------------------------------------------------
plt.figure(1,figsize=(10,7), facecolor='lightgray')

# Crear una figura con dos subgráficos
plt.subplot(211)  # Primer subgráfico, temperatura vs tiempo
plt.plot(t_e, y_e, 'm-',label= 'Presión tanque pulmón')
plt.gca().yaxis.set_major_locator(MultipleLocator(1))   # Define cada cuanto se
marca el eje y
plt.gca().xaxis.set_major_locator(MultipleLocator(50)) # Define cada cuanto se
marca el eje x

plt.title('Curva de rxn experimental tanque pulmón')
plt.xlabel('Tiempo [s]')
plt.ylabel(' Presión [Psig]')
plt.grid(True)
plt.legend(loc= 'best')

plt.subplot(212)  # Segundo subgráfico, % apertura válvula vs tiempo
plt.plot(t_e, u_e, 'y-',label='Válvula_ent')
plt.gca().xaxis.set_major_locator(MultipleLocator(50))

plt.xlabel('Tiempo [s]')
plt.ylabel('Presión en la válvula_ent [Psig]')
plt.grid(True)
plt.legend(loc= 'best')
```

CONTROL P PRESIÓN - Modelo fenomenológico

```
# 1. Importar todas las librerias y funciones que se necesitan
# ------------------------------------------------------------------------------
import numpy as np               # Manejo de vectores
import matplotlib.pyplot as plt  # Permite graficar
import math                      # Funciones matemáticas
# ------------------------------------------------------------------------------
# 2. Parámetros
# ------------------------------------------------------------------------------
Pe = 40            # Presión de entrada de la válvula de entrada [psi]
D_tanque = 0.3     # diámetro del tanque pulmón [m]
H_tanque = 0.805   # altura del tanque pulmón [m]
Cv = 0.0016        # coeficiente de la válvula [m^2]
Patm = 12.375536   # presión atmosférica [psig]
V = (np.pi*((D_tanque/2)**2))*H_tanque  # volumen tanque [m^3]

# ------------------------------------------------------------------------------
# 3. Condiciones iniciales para el modelo
# ------------------------------------------------------------------------------
Pv_ent = 5.2        # Presión sobre el vástago de la válvula de entrada [psig]
Pv_sal = 8          # Presión sobre el vástago de la válvula de salida [psig]
P_sp = 9            # Set point [psig]
P_tanque = 0        # Presión inicial en el tanque pulmón (variable controlada)

# ------------------------------------------------------------------------------
# 4. Vector tiempo
# ------------------------------------------------------------------------------
t_0 = 0          # Tiempo inicial [s]
t_f = 1800        # Tiempo final [s]
dt = 0.01        # Tamaño de paso
iteraciones = round(t_f/dt) # Cantidad de iteraciones
t = np.linspace(t_0, t_f, (iteraciones+1))

# ------------------------------------------------------------------------------
# 5. Entradas y constantes
# ------------------------------------------------------------------------------
y_0 = P_tanque        # Presión inicial de la variable controlada [psig]
d = Pv_sal            # Perturbación (presión sobre el el vástago de la válvula
de salida) [psig]
u_0 = Pv_ent          # Valor inicial de la variable manipulada [psig]
a = 1.1               # Constante empírica
rho = 0.97            # Densidad del aire [Kg/m^3]

# ------------------------------------------------------------------------------
# 6. Creación de Vectores
# ------------------------------------------------------------------------------
y_dif = np.zeros(iteraciones+1)  # Guarda los valores de presión en cada
iteración
uc_dif = np.zeros(iteraciones+1)  # vector de la variable manipulada
d_dif = np.zeros(iteraciones+1)  # vector de perturbación
SP_dif = np.zeros(iteraciones+1)  # vector de set point
E_dif = np.zeros(iteraciones+1)  # vector de error
# ------------------------------------------------------------------------------
# 7. Parámetros de control
# ------------------------------------------------------------------------------
```

```
Ku = -3.92    # [psig/%apertura]
Tao = 90    # Tiempo de respuesta [s]

# Ajuste por el Método Ziegler-Nichols
Kp = 0.5 * Ku    # Constante proporcional de control
print("ku=", Ku)
print("Tao=", Tao)
print("Kp=", Kp)

# ------------------------------------------------------------------------------
# 8. Ecuación diferencial (método de Euler)
# ------------------------------------------------------------------------------
# valores iniciales a las variables
y = y_0
u = u_0

Suma_int = 0
Delta = 0
for i in range(0,iteraciones+1):
  # Perturbaciones en la válvula de entrada
  if t[i]<660:
    d=4

  if t[i]>660 and t[i]<1380:
    d=11

  if t[i]>1380 and t[i]<1740:
    d=16

  d_dif[i] = d

  # Perturbaciones con cambios en el Y_sp
  if t[i]<660:
    P_sp=10

  if t[i]>660:
    P_sp=15

  SP_dif[i] = P_sp
  y_dif[i] = y

  Pe_abs = Pe+Patm
  fx_in = abs(a-math.exp(a-((u-3)/12)))
  fx_out = abs(a-math.exp(a-((d-3)/12)))
  V_in = Cv * fx_in * math.sqrt((Pe - y)/rho)
  V_out = Cv * fx_out * math.sqrt((y)/rho)
  dydt = (1/V)*((Pe_abs)*V_in - (Patm)*V_out) # Calcula la derivada de P según
el MSBF

  y = y + dydt * dt   # Actualiza el valor de la variable

# ------------------------------------------------------------------------------
# 7. Cambio a variables de control
# ------------------------------------------------------------------------------
  spam = 15-3  # Límites de la variable de interés
  error = (P_sp - y)*100/2 #[error en %]
```

```
  E_dif[i] = error

  uc = Kp*error   # Control P

  if uc > 100:
    uc = 100
  if uc < 0:
    uc = 0

  uc_dif[i] = uc

  # Convertir la variable de control a una de proceso
  u = (uc*spam/100)+ 3
# ------------------------------------------------------------------------------
# 7. Graficar resultados
# ------------------------------------------------------------------------------
plt.figure(1,figsize=(10,8), facecolor='lightgray')

# Parte 1: Presión
plt.subplot(311) # grafica la primera posición
plt.title('Modelo semi-físico de base fenomenológica')
plt.plot(t,y_dif,"c-", label = "Presión en el tanque pulmón")
plt.plot(t, SP_dif, 'r--',label='Ysp', linewidth=1) # graficar el set point
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión tanque pulmón [psi]') #Eje y
plt.legend(loc = 'lower right')
plt.grid(True)

# Parte 2: variable manipulada [%]
plt.subplot(312) # grafica la segunda posición
plt.plot(t, uc_dif, 'm', label = '% apertura válvula_ent', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('válvula_ent [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)

# Parte 3: Perturbación
plt.subplot(313) # grafica la tercera posición
plt.plot(t, d_dif, 'g', label = 'Perturbación (Pv_sal)', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión V_sal [psig]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)

# Gráfico del error
plt.figure(2)
plt.plot(t, E_dif, 'orange', label = 'Error', linewidth=2) # grafica el error
plt.title('Error en el tanque pulmón')
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Error presión [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)
```

CONTROL PID PRESIÓN - Modelo fenomenológico

```
# 1. Importar todas las librerias y funciones que se necesitan
```

```
# ------------------------------------------------------------------------------
import numpy as np                # Manejo de vectores
import matplotlib.pyplot as plt  # Permite graficar
import math                       # Funciones matemáticas
# ------------------------------------------------------------------------------
# 2. Parámetros
# ------------------------------------------------------------------------------
Pe = 40            # Presión de entrada de la válvula de entrada [psi]
D_tanque = 0.3     # diámetro del tanque pulmón [m]
H_tanque = 0.805   # altura del tanque pulmón [m]
Cv = 0.0016        # coeficiente de la válvula [m^2]
Patm = 12.375536   # presión atmosférica [psig]
V = (np.pi*((D_tanque/2)**2))*H_tanque  # volumen tanque [m^3]

# ------------------------------------------------------------------------------
# 3. Condiciones iniciales para el modelo
# ------------------------------------------------------------------------------
Pv_ent = 5.2        # Presión sobre el vástago de la válvula de entrada [psig]
Pv_sal = 8          # Presión sobre el vástago de la válvula de salida [psig]
P_sp = 9            # Set point [psig]
P_tanque = 0        # Presión inicial en el tanque pulmón (variable controlada)

# ------------------------------------------------------------------------------
# 4. Vector tiempo
# ------------------------------------------------------------------------------
t_0 = 0          # Tiempo inicial [s]
t_f = 1800        # Tiempo final [s]
dt = 0.01        # Tamaño de paso
iteraciones = round(t_f/dt) # Cantidad de iteraciones
t = np.linspace(t_0, t_f, (iteraciones+1))

# ------------------------------------------------------------------------------
# 5. Entradas y constantes
# ------------------------------------------------------------------------------
y_0 = P_tanque        # Presión inicial de la variable controlada [psig]
d = Pv_sal            # Perturbación (presión sobre el el vástago de la válvula
de salida) [psig]
u_0 = Pv_ent          # Valor inicial de la variable manipulada [psig]
a = 1.1               # Constante empírica
rho = 0.97            # Densidad del aire [Kg/m^3]

# ------------------------------------------------------------------------------
# 6. Creación de Vectores
# ------------------------------------------------------------------------------
y_dif = np.zeros(iteraciones+1)  # Guarda los valores de presión en cada
iteración
uc_dif = np.zeros(iteraciones+1)  # vector de la variable manipulada
d_dif = np.zeros(iteraciones+1)  # vector de perturbación
SP_dif = np.zeros(iteraciones+1)  # vector de set point
E_dif = np.zeros(iteraciones+1)  # vector de error
# ------------------------------------------------------------------------------
# 7. Parámetros de control
# ------------------------------------------------------------------------------
Ku = -3.92    # [psig/%apertura]
Tao = 90    # Tiempo de respuesta [s]
```

```
# Ajuste por el Método Ziegler-Nichols
Kp = 0.6 * Ku             # Constante proporcional de control
Ki = (1.2 * (Ku / Tao))   # Constante integral de control
Kd = ((3/40) * Ku * Tao)  # Constante derivativa de control
Tao_i = Kp/Ki             # Tiempo de integral de control
Tao_d = Kd/Kp             # Tiempo de derivativa de control
print("ku=", Ku)
print("Tao=", Tao)
print("Kp=", Kp)
print("Ki=", Ki)
print("Kd=", Kd)
print("Tao_i=", Tao_i)
print("Tao_d=", Tao_d)
# ------------------------------------------------------------------------------
# 8. Ecuación diferencial (método de Euler)
# ------------------------------------------------------------------------------
# valores iniciales a las variables
y = y_0
u = u_0

Suma_int = 0
Delta = 0
uc = 0
for i in range(0,iteraciones+1):
  # Perturbaciones en la válvula de entrada
  if t[i]<660:
    d=4

  if t[i]>660 and t[i]<1380:
    d=11

  if t[i]>1380 and t[i]<1740:
    d=16

  d_dif[i] = d

  # Perturbaciones con cambios en el Y_sp
  if t[i]<660:
    P_sp=10

  if t[i]>660:
    P_sp=15

  SP_dif[i] = P_sp
  y_dif[i] = y

  Pe_abs = Pe+Patm
  fx_in = abs(a-math.exp(a-((u-3)/12)))
  fx_out = abs(a-math.exp(a-((d-3)/12)))
  V_in = Cv * fx_in * math.sqrt((Pe - y)/rho)
  V_out = Cv * fx_out * math.sqrt((y)/rho)
  dydt = (1/V)*((Pe_abs)*V_in - (Patm)*V_out) # Calcula la derivada de P según
el MSBF

  y = y + dydt * dt   # Actualiza el valor de la variable
# ------------------------------------------------------------------------------
```

```
# 7. Cambio a variables de control
# ------------------------------------------------------------------------------
  spam = 15-3  # Límites de la variable de interés
  error = (P_sp - y)*100/2 #[error en %]
  E_dif[i] = error
  Suma_int = Suma_int + error*dt
  if i > 0:
    Delta = E_dif[i] - E_dif[i-1]

  uc = Kp*error + Ki*Suma_int + Kd*Delta/dt  # Control PID
  #uc += Kp*error + Ki*Suma_int + Kd*Delta/dt


  if uc > 100:
    uc = 100
  if uc < 0:
    uc = 0

  uc_dif[i] = uc

  # Convertir la variable de control a una de proceso
  u = (uc*spam/100)+ 3
# ------------------------------------------------------------------------------
# 7. Graficar resultados
# ------------------------------------------------------------------------------
plt.figure(1,figsize=(10,8), facecolor='lightgray')

# Parte 1: Presión
plt.subplot(311) # grafica la primera posición
plt.title('Modelo semi-físico de base fenomenológica')
plt.plot(t,y_dif,"c-", label = "Presión en el tanque pulmón")
plt.plot(t, SP_dif, 'r--',label='Ysp', linewidth=1) # graficar el set point
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión tanque pulmón [psi]') #Eje y
plt.legend(loc = 'lower right')
plt.grid(True)

# Parte 2: variable manipulada [%]
plt.subplot(312) # grafica la segunda posición
plt.plot(t, uc_dif, 'm', label = '% apertura válvula_ent', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('válvula_ent [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)

# Parte 3: Perturbación
plt.subplot(313) # grafica la tercera posición
plt.plot(t, d_dif, 'g', label = 'Perturbación (Pv_sal)', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión V_sal [psig]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)

# Gráfico del error
plt.figure(2)
plt.plot(t, E_dif, 'orange', label = 'Error', linewidth=2) # grafica el error
```

```
plt.title('Error en el tanque pulmón')
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Error presión [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)
```

SINTONIZACIÓN - CONTROL PID PRESIÓN - Modelo fenomenológico

```
# 1. Importar todas las librerias y funciones que se necesitan
# ------------------------------------------------------------------------------
import numpy as np               # Manejo de vectores
import matplotlib.pyplot as plt  # Permite graficar
import math                      # Funciones matemáticas
# ------------------------------------------------------------------------------
# 2. Parámetros
# ------------------------------------------------------------------------------
Pe = 40            # Presión de entrada de la válvula de entrada [psi]
D_tanque = 0.3     # diámetro del tanque pulmón [m]
H_tanque = 0.805   # altura del tanque pulmón [m]
Cv = 0.0016        # coeficiente de la válvula [m^2]
Patm = 12.375536   # presión atmosférica [psig]
V = (np.pi*((D_tanque/2)**2))*H_tanque  # volumen tanque [m^3]

# ------------------------------------------------------------------------------
# 3. Condiciones iniciales para el modelo
# ------------------------------------------------------------------------------
Pv_ent = 5.2        # Presión sobre el vástago de la válvula de entrada [psig]
Pv_sal = 8          # Presión sobre el vástago de la válvula de salida [psig]
P_sp = 9            # Set point [psig]
P_tanque = 0        # Presión inicial en el tanque pulmón (variable controlada)

# ------------------------------------------------------------------------------
# 4. Vector tiempo
# ------------------------------------------------------------------------------
t_0 = 0          # Tiempo inicial [s]
t_f = 1800        # Tiempo final [s]
dt = 0.01        # Tamaño de paso
iteraciones = round(t_f/dt) # Cantidad de iteraciones
t = np.linspace(t_0, t_f, (iteraciones+1))

# ------------------------------------------------------------------------------
# 5. Entradas y constantes
# ------------------------------------------------------------------------------
y_0 = P_tanque        # Presión inicial de la variable controlada [psig]
d = Pv_sal            # Perturbación (presión sobre el el vástago de la válvula
de salida) [psig]
u_0 = Pv_ent          # Valor inicial de la variable manipulada [psig]
a = 1.1               # Constante empírica
rho = 0.97            # Densidad del aire [Kg/m^3]

# ------------------------------------------------------------------------------
# 6. Creación de Vectores
# ------------------------------------------------------------------------------
y_dif = np.zeros(iteraciones+1)  # Guarda los valores de presión en cada
iteración
```

```
uc_dif = np.zeros(iteraciones+1)  # vector de la variable manipulada
d_dif = np.zeros(iteraciones+1)  # vector de perturbación
SP_dif = np.zeros(iteraciones+1)  # vector de set point
E_dif = np.zeros(iteraciones+1)  # vector de error
# ------------------------------------------------------------------------------
# 7. Parámetros de control
# ------------------------------------------------------------------------------
Ku = -3.92    # [psig/%apertura]
Tao = 90    # Tiempo de respuesta [s]

# Ajuste por el Método Ziegler-Nichols
Kp = 0.6 * Ku *2                # Constante proporcional de control
Ki = (1.2 * (Ku / Tao))*45      # Constante integral de control
Kd = ((3/40) * Ku * Tao)*0.001  # Constante derivativa de control
Tao_i = Kp/Ki             # Tiempo de integral de control
Tao_d = Kd/Kp             # Tiempo de derivativa de control
print("ku=", Ku)
print("Tao=", Tao)
print("Kp=", Kp)
print("Ki=", Ki)
print("Kd=", Kd)
print("Tao_i=", Tao_i)
print("Tao_d=", Tao_d)
# ------------------------------------------------------------------------------
# 8. Ecuación diferencial (método de Euler)
# ------------------------------------------------------------------------------
# valores iniciales a las variables
y = y_0
u = u_0

Suma_int = 0
Delta = 0
for i in range(0,iteraciones+1):
  # Perturbaciones en la válvula de entrada
  if t[i]<660:
    d=4

  if t[i]>660 and t[i]<1380:
    d=11

  if t[i]>1380 and t[i]<1740:
    d=16
  d_dif[i] = d

  # Perturbaciones con cambios en el Y_sp
  if t[i]<660:
    P_sp=10

  if t[i]>660:
    P_sp=15

  SP_dif[i] = P_sp
  y_dif[i] = y

  Pe_abs = Pe+Patm
  fx_in = abs(a-math.exp(a-((u-3)/12)))
```

```
  fx_out = abs(a-math.exp(a-((d-3)/12)))
  V_in = Cv * fx_in * math.sqrt((Pe - y)/rho)
  V_out = Cv * fx_out * math.sqrt((y)/rho)
  dydt = (1/V)*((Pe_abs)*V_in - (Patm)*V_out) # Calcula la derivada de P según
el MSBF

  y = y + dydt * dt   # Actualiza el valor de la variable
# ------------------------------------------------------------------------------
# 7. Cambio a variables de control
# ------------------------------------------------------------------------------
  spam = 15-3  # Límites de la variable de interés
  error = (P_sp - y)*100/2 #[error en %]
  E_dif[i] = error
  Suma_int = Suma_int + error*dt
  if i > 0:
    Delta = E_dif[i] - E_dif[i-1]

  uc = Kp*error + Ki*Suma_int + Kd*Delta/dt  # Control PID

  if uc > 100:
    uc = 100
  if uc < 0:
    uc = 0

  uc_dif[i] = uc

  # Convertir la variable de control a una de proceso
  u = (uc*spam/100)+ 3
# ------------------------------------------------------------------------------
# 7. Graficar resultados
# ------------------------------------------------------------------------------
plt.figure(1,figsize=(10,8), facecolor='lightgray')

# Parte 1: Presión
plt.subplot(311) # grafica la primera posición
plt.title('Modelo semi-físico de base fenomenológica')
plt.plot(t,y_dif,"c-", label = "Presión en el tanque pulmón")
plt.plot(t, SP_dif, 'r--',label='Ysp', linewidth=1) # graficar el set point
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión tanque pulmón [psi]') #Eje y
plt.legend(loc = 'lower right')
plt.grid(True)

# Parte 2: variable manipulada [%]
plt.subplot(312) # grafica la segunda posición
plt.plot(t, uc_dif, 'm', label = '% apertura válvula_ent', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('válvula_ent [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)

# Parte 3: Perturbación
plt.subplot(313) # grafica la tercera posición
plt.plot(t, d_dif, 'g', label = 'Perturbación (Pv_sal)', linewidth=2)
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Presión V_sal [psig]') #Eje y
```

```
plt.legend(loc='lower right')
plt.grid(True)

# Gráfico del error
plt.figure(2)
plt.plot(t, E_dif, 'orange', label = 'Error', linewidth=2) # grafica el error
plt.title('Error en el tanque pulmón')
plt.xlabel('Tiempo [s]') #Eje x
plt.ylabel('Error presión [%]') #Eje y
plt.legend(loc='lower right')
plt.grid(True)
```

Control de Temperatura en un sistema de calentamiento de agua con vapor

Curvas de reacción del sistema de calentamiento de agua con datos experimentales del Lab #2

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from google.colab import drive

# Montar Google Drive
drive.mount("/content/drive", force_remount=True)

# Leer archivo de datos
data = np.genfromtxt('/content/drive/MyDrive/Colab
Notebooks/curva_reaccion_manual_1_caldera.txt',
                     usecols=(0, 1, 2), skip_header=1, invalid_raise=False)

# Extraer columnas
t_exp = data[:, 0]
u_exp = data[:, 1]
y_exp = data[:, 2]

# Condiciones iniciales
d = 36.75
y_o_mod = y_exp[0]
u_o_mod = u_exp[0]
t_o_mod = int(t_exp[0])
t_f_mod = int(t_exp[-1])
dt = 0.5
iteraciones = round(t_f_mod / dt)  #cuántas veces se calcula
t_mod = np.linspace(t_o_mod, t_f_mod, iteraciones + 1)

# Vectores para almacenar resultados
y_all = np.zeros(iteraciones + 1)  #guarda los valores de temperatura simulada
que genera el modelo, paso a paso
u_all = np.zeros(iteraciones + 1)  #guarda los valores de apertura de válvula
interpolada en cada t_mod
y_all_interp = np.zeros(len(t_exp))  #se llena con los valores simulados de y,
pero alineados a los tiempos experimentales t_exp (usando interpolación)
```

```
# Valores iniciales Tao y Ku, respectivamente
x_0 = [12, 0.4]

# Función objetivo con modelo desplazado (evita caída inicial)
def objetivo(x):
    tao = x[0]
    Ku = x[1]
    y_mod = y_o_mod # Inicializa y_mod with the initial experimental value

    for i in range(iteraciones + 1):
        u_mod = np.interp(t_mod[i], t_exp, u_exp)
        u_all[i] = u_mod # Store interpolated u_mod
        dydt = (-y_mod+Ku*u_mod)/tao +(d/tao)   # calcula la derivada de T (o
de y) ELIMINÉ EL +D
        y_mod = y_mod + dydt*dt   # Actualiza el valor de la variable según el
método de Euler

        y_all[i] = y_mod  # guarda el valor de la temperatura y


    for j in range(len(t_exp)):
        y_interp = np.interp(t_exp[j], t_mod, y_all)
        y_all_interp[j] = y_interp

    error_obj = sum((y_exp - y_all_interp) ** 2)
    return error_obj



print("Valor inicial para tao =", round(x_0[0],3))
print("Valor inicial para Ku =", round(x_0[1],3))
print("Valor inicial del error =", round(objetivo(x_0),3))

# Optimización con límites
bounds = [(1, 100), (0.01, 2)] #Impone un rango permitido para cada valor
solucion = minimize(objetivo, x_0, bounds=bounds)  #Busca los mejores valores
de tao y Ku que minimizan el error.
x = solucion.x

print("Valor optimizado para tao =", round(x[0],2))
print("Valor optimizado para Ku =", round(x[1],3))
print("Valor optimizado del error =", round(objetivo(x),3))

# Ejecutar simulación final
objetivo(x)

# Gráficas

plt.figure(1, figsize=(10, 8), facecolor='lightgray')


# Panel 1: Temperatura experimental + simulada
plt.subplot(2, 1, 1)

#plt.figure(figsize=(10, 5))
plt.plot(t_exp, y_exp, 'r', label='Temperatura experimental')
```

```
plt.plot(t_mod, y_all, 'b-', label='Temperatura simulada')
plt.xlabel('Tiempo [min]')
plt.ylabel('Temperatura [°C]')
plt.title("Curvas de reacción")
plt.xlabel("Tiempo [min]")
plt.ylabel("Temperatura [°C]")
plt.legend()
plt.grid(True)


# Panel 1: Apertura (u)
plt.subplot(2, 1, 2)
plt.plot(t_mod, u_all, 'k-', label='% Apertura válvula')
plt.title("Variable manipulada (u)")
plt.xlabel("Tiempo [min]")
plt.ylabel("% Apertura válvula")

plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Control de temperatura Lab #3

```
# -*- coding: utf-8 -*-
"""
Editor de Spyder

Este es un archivo temporal.
"""
#Sistema de control de temperatura. Laboratorio 3. Curva de Temperatura

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from matplotlib.ticker import PercentFormatter


# === 1. Leer datos ===
data = np.genfromtxt(r"C:\Users\usuario\Downloads\Datos_Temp_LAB_3.txt",
skip_header=1)
t_exp = data[:,0]          # tiempo [min]
y_exp = data[:,1]          # Temperatura [°C]
u_exp = data[:,2]          # % Apertura
y_setp = data[:, 3]         # Setpoint [°C]
d = data[:, 4]              # Perturbaciones

#Condiciones iniciales
y_o_mod = y_exp[0]
t_o_mod = t_exp[0]
t_f_mod = t_exp[-1]
dt = 0.1
t_mod = np.arange(t_o_mod, t_f_mod + dt, dt)
```

```
iteraciones = len(t_mod)

#Interpola u_exp to match t_mod
u_exp_interp = np.interp(t_mod, t_exp, u_exp)


#Función objetivo
def objetivo(x):
    tau, Ku = x
    y_mod = y_o_mod
    y_all = np.zeros(iteraciones)

    for i in range(iteraciones):
        # Use the interpolated u_exp_interp
        u_model = u_exp_interp[i]
        dydt = (1 / tau) * (-y_mod + Ku * u_model)
        y_mod += dydt * dt
        y_all[i] = y_mod

    y_interp = np.interp(t_exp, t_mod, y_all)
    error = np.sum((y_exp - y_interp) ** 2)
    return error

#Optimización
x_0 = [10, 0.4]
bounds = [(1e-3, None), (1e-3, None)]

print("Valores iniciales:")
print(f"τ = {x_0[0]}  Ku = {x_0[1]}  Error = {objetivo(x_0):.4f}")

solucion = minimize(objetivo, x_0, bounds=bounds)
tau_opt, Ku_opt = solucion.x
error_opt = objetivo(solucion.x)

print("\nValores optimizados:")
print(f"τ = {tau_opt:.3f}")
print(f"Ku = {Ku_opt:.3f}")
print(f"Error = {error_opt:.4f}")

#Simulación con parámetros óptimos
y_mod = y_o_mod
y_all = np.zeros(iteraciones)

for i in range(iteraciones):
    # Use the interpolated u_exp_interp
    u_model = u_exp_interp[i]
    dydt = (1 / tau_opt) * (-y_mod + Ku_opt * u_model)
    y_mod += dydt * dt
    y_all[i] = y_mod

y_all_interp = np.interp(t_exp, t_mod, y_all)

#Interpolar el setpoint a t_mod si es necessario
y_setp_interp = np.interp(t_mod, t_exp, y_setp)

#Gráficas
```

```
# plt.figure(1)
#plt.figure(figsize=(8, 10))
plt.figure(1,figsize=(10,8), facecolor='lightgray')

# Panel 1: Temperatura experimental y Set Point
plt.subplot(3, 1, 1)
plt.plot(t_exp, y_exp, 'r', label='y [°C]')
plt.plot(t_mod, y_setp_interp, 'g--', label='ysp [°C]') #Graficar el Set Point
plt.title('Temperatura (y) y Set Point')   # Título propio
plt.ylabel('Temp. experimental [°C]', fontsize=11)
plt.legend()
plt.grid(True)
plt.xlim(0, 55)

# Panel 2: % Apertura de válvula
plt.subplot(3, 1, 2)

plt.plot(t_mod, u_exp_interp, 'b-', label='u [%]')
#plt.plot(t_mod, y_setp_interp, 'g--', label='Set Point')
  # rango de 0 a 100%
#plt.yticks(np.linspace(0, 100, 6))  # opcional: 0,20,40,60,80,100
#plt.gca().yaxis.set_major_formatter(PercentFormatter(xmax=100))
plt.title('Temperatura (y) y Apertura de la válvula (u)')
plt.ylabel('% Apertura', fontsize=11)
plt.legend()
plt.grid(True)
plt.xlim(0, 55)


# Panel 3: Perturbación
plt.subplot(3, 1, 3)
d_interp = np.interp(t_mod, t_exp, d)
plt.plot(t_mod, d_interp, 'm-', label='d [L/min]')
plt.title('Temperatura (y) y Perturbación (d)')
plt.ylabel('Flujo W ent (L/min)', fontsize=11)
plt.legend()
plt.grid(True)
plt.xlim(0, 55)



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