PROJETO - SISTEMAS EMBARCADOS

Gustavo Oliveira

Tecnologias

Carregar dataset

```python
def carregar_dataset():
    # Detecta o diretório base, considerando execução com PyInstaller
    if getattr(sys, "frozen", False):
        # Executável gerado com PyInstaller
        base_path = sys._MEIPASS
        base_path = os.path.join(base_path, "app")
    else:
        # Execução normal (modo desenvolvimento)
        base_path = os.path.dirname(__file__)

    file_path = os.path.join(base_path, "Dataset_Grupo1.mat")

    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Arquivo {file_path} não encontrado.")

    data = loadmat(file_path)
    print("Chaves do arquivo .mat:", data.keys())

    # Obtem o objeto principal
    reaction_data = data["reactionExperiment"]

    # A estrutura está encapsulada — precisamos acessar o primeiro item [0][0]
    reaction_data = reaction_data[0, 0]

    # Separar os campos
    sample_time = reaction_data["sampleTime"]
    data_input = reaction_data["dataInput"]
    data_output = reaction_data["dataOutput"]
    physical_quantity = reaction_data["physicalQuantity"]
    units = reaction_data["units"]

    sample_time = sample_time.astype(np.float64)
    return sample_time, np.mean(data_input), data_output
```

Identificação

```
# Calcular o ganho estático K
  valor_inicial = output[0]
  output_padronizado = output - valor_inicial
  k = output_padronizado[-1] / step

  if method == "Smith":
      val1 = 0.283 * output_padronizado[-1]
      val2 = 0.632 * output_padronizado[-1]
  else:  # Sundaresan
      val1 = 0.353 * output_padronizado[-1]
      val2 = 0.853 * output_padronizado[-1]

  index1 = np.where(output_padronizado >= val1)[0][0]
  index2 = np.where(output_padronizado >= val2)[0][0]
  t1 = time[index1]
  t2 = time[index2]

  if method == "Smith":
      tau = (t2 - t1) * 1.5
      theta = t2 - tau
  else:  # Sundaresan
      tau = (t2 - t1) * (2 / 3)
      theta = (1.3 * t1) - (0.29 * t2)
```

EQM

```
# Parte sem atraso
G = tf([k], [tau, 1])

# Aproximação de Padé para o atraso
num_delay, den_delay = pade(theta, 6)
delay = tf(num_delay, den_delay)

# Sistema com atraso aproximado
f_identification = series(delay, G)

# Simular a resposta ao degrau do sistema identificado
t_sim, y_sim = step_response(f_identification, T=time)
y_sim = y_sim * step

# Calcular EQM
output_ref = output - output[0]
eqm = np.sqrt(np.mean((output_ref - y_sim) ** 2))
```

EQM Smith = 1,278

EQM Sundaresan = 1,94

Cálculo do melhor método

```
# Calcular os parâmetros para os diferentes métodos de identificação
k_sun, tau_sun, theta_sun, eqm_sun = identification_process(
    step, time_dataset, output_dataset, "Sundaresan"
)
k_smi, tau_smi, theta_smi, eqm_smi = identification_process(
    step, time_dataset, output_dataset, "Smith"
)

# Obter o modelo FOPDT com atraso via Padé (modelo exato)
fopdt_model_with_delay = identificar_fopdt(step, time_dataset, output_dataset)

# Comparar qual método gerou o menor EQM
eqms = {"Sundaresan": eqm_sun, "Smith": eqm_smi}
best_method = min(eqms, key=eqms.get)
```

Melhor método

```
# Definir o sistema conforme o melhor método
if best_method == "Sundaresan":
    f_identification = tf([k_sun], [tau_sun, 1])
    num_delay, den_delay = pade(theta_sun, 2)
    k = round(k_sun, 3)
    tau = round(tau_sun, 3)
    theta = round(theta_sun, 3)
else:  # Smith
    f_identification = tf([k_smi], [tau_smi, 1])
    num_delay, den_delay = pade(theta_smi, 2)
    k = round(k_smi, 3)
    tau = round(tau_smi, 3)
    theta = round(theta_smi, 3)

delay = tf(num_delay, den_delay)
f_identification = series(delay, f_identification)

# Simular a resposta ao degrau do modelo FOPDT (modelo exato)
t_sim_fopdt, y_sim_fopdt = step_response(fopdt_model_with_delay, T=time_dataset)
y_sim_fopdt = y_sim_fopdt * step

# Simular a resposta ao degrau do sistema identificado
t_sim_ident, y_sim_ident = step_response(f_identification, T=time_dataset)
y_sim_ident = y_sim_ident * step

# Criar o gráfico
plt.figure(figsize=(6, 4))
plt.plot(time_dataset, y_sim_fopdt, "b", label="Referencia")
plt.plot(time_dataset, y_sim_ident, "m--", label=f"Modelo ({best_method})")
```

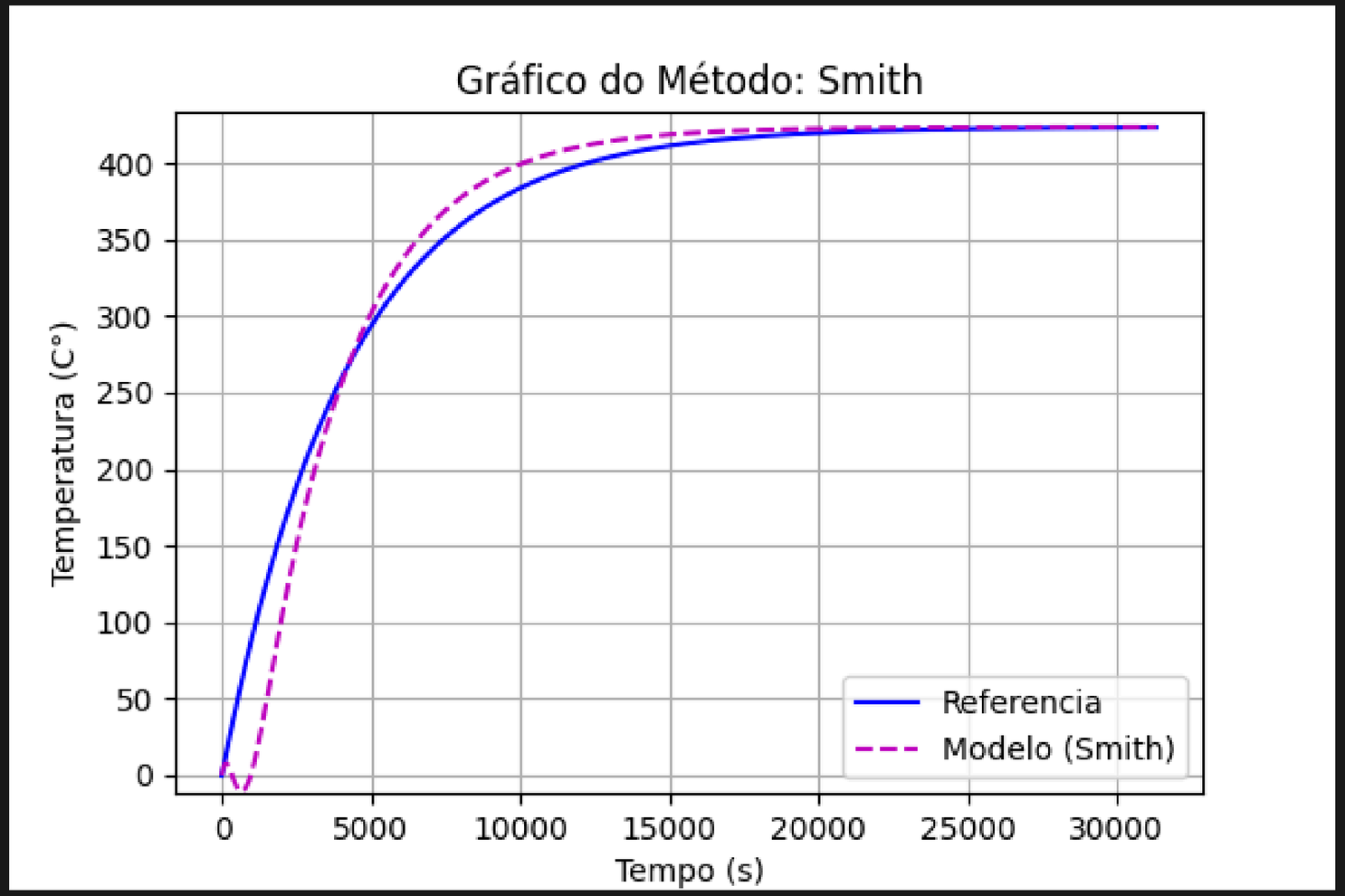
Smith

Ganho (K): 4.663

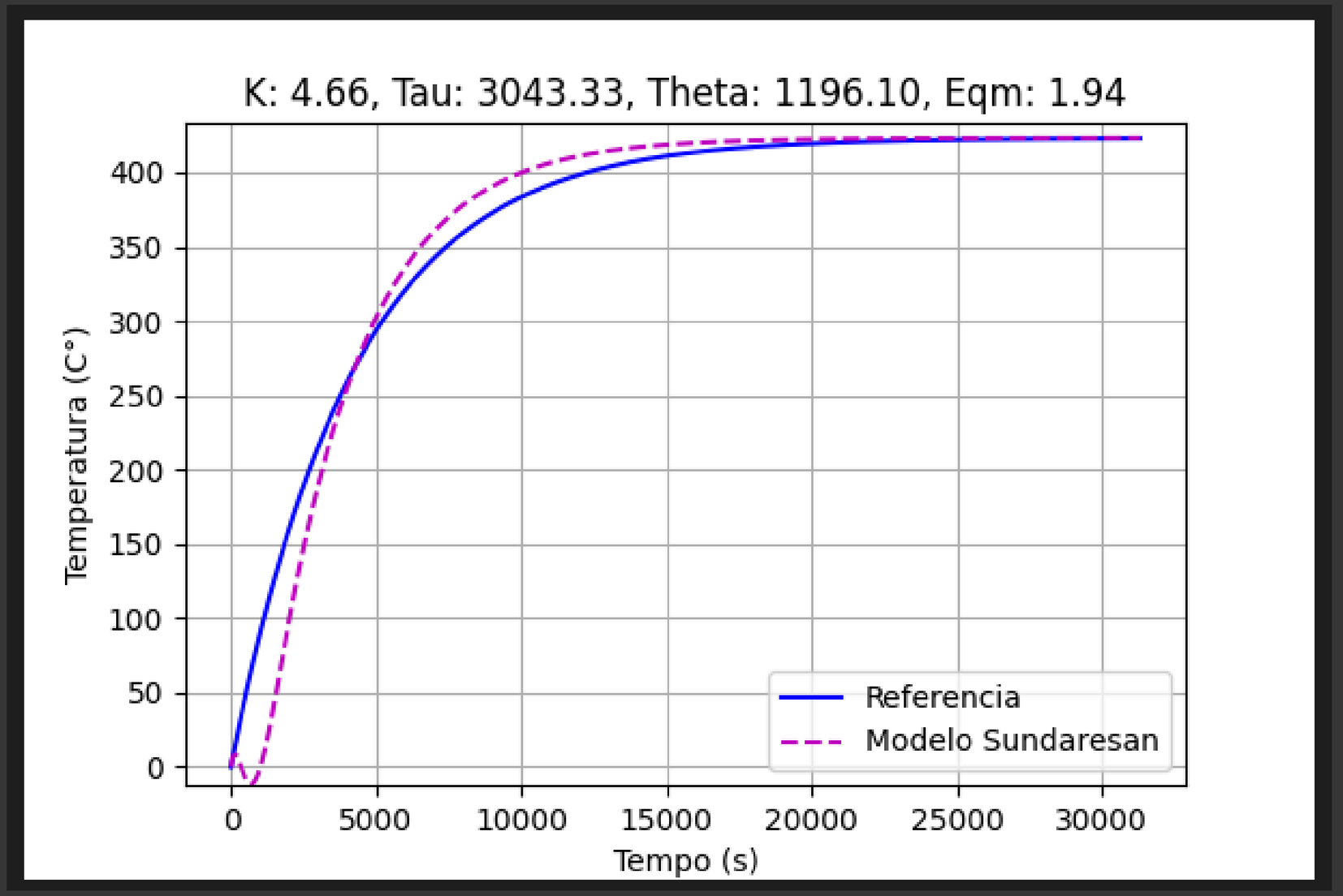
Tau (τ): 3097.5

Theta (θ): 1132.5

EQM: 1.278



Sundaresan

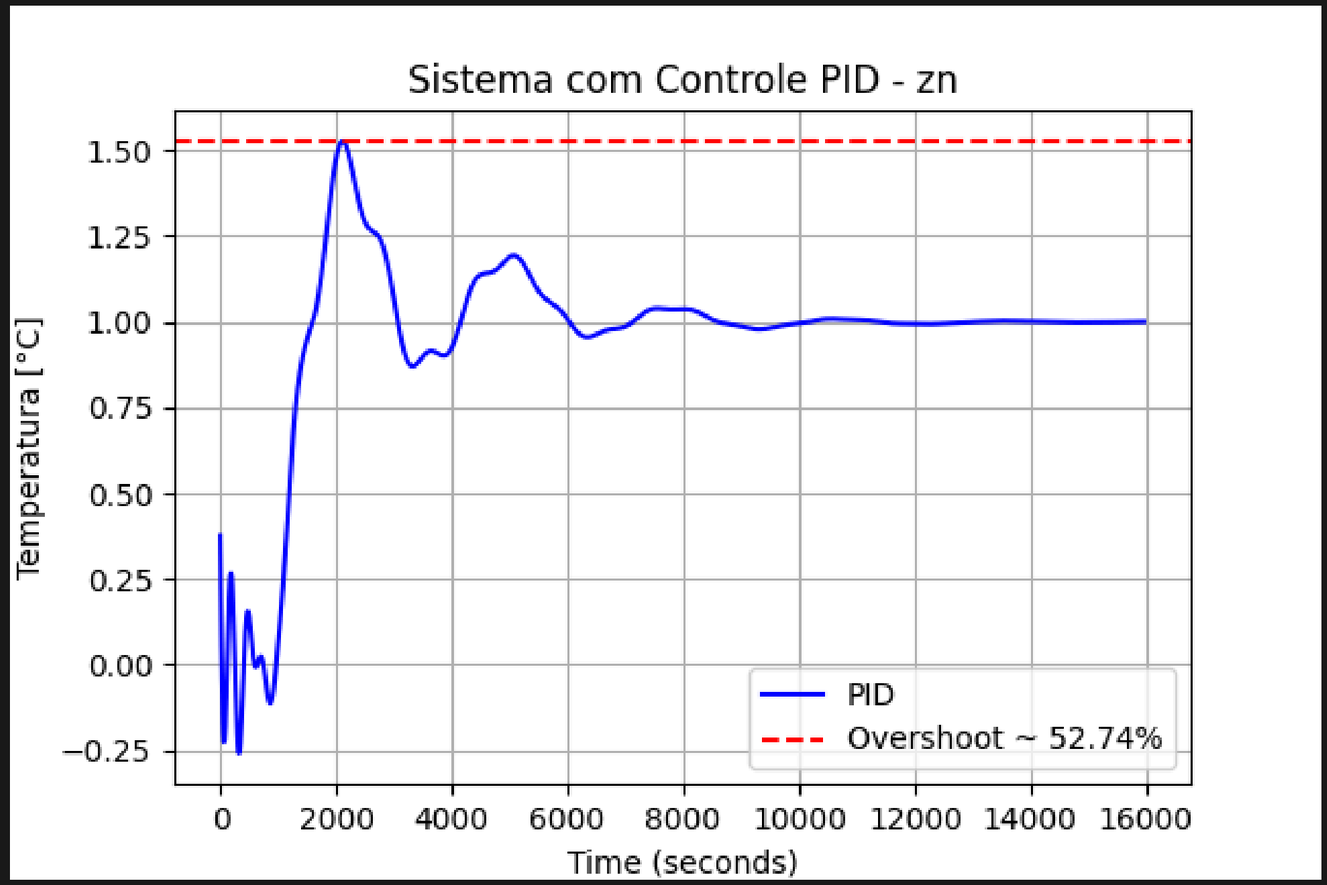

K: 4.66, Tau: 3043.33, Theta: 1196.10, Eqm: 1.94
Temperatura (C°)
Tempo (s)
0
50
100
150
200
250
300
350
400
0
5000
10000
15000
20000
25000
30000
Referencia
Modelo Sundaresan

Controladores PID

```
if method == "zn":
    kp, ti, td = ziegler_nichols_malha_aberta(k, tau, theta)
elif method == "chr":
    kp, ti, td = chr_com_sobre_valor(k, tau, theta)

# Criar função de transferência do controlador PID
PID = tf([kp * td, kp, kp / ti], [1, 0])

# Sistema de processo (sem controle)
G = tf([k], [tau, 1])

num_delay, den_delay = pade(theta, 6)
delay = tf(num_delay, den_delay)

# Sistema com atraso
sistema = series(delay, G)

# Sistema em malha fechada com PID
malha_fechada = feedback(series(PID, sistema), 1)

# Resposta ao degrau
t, y = step_response(malha_fechada)
y_max = np.max(y)
y_min = np.min(y)
```

Ziegler Nichols

Kp: 0.704 **Ti:** 2265.000 **Td:** 566.250 **Overshoot:** 52.74%



CHR (com sobrevalor)

Kp: 0.669 **Ti:** 4203.307 **Td:** 535.673 **Overshoot:** 33.99%