PROJETO DE CONTROLE - JÚLIO MELO CAMPOS

```python
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.signal import place_poles
from scipy.integrate import solve_ivp
import sympy as sp

# ---------------------- ETAPA 1: SIMULAÇÃO NÃO LINEAR ----------------------

# Constantes físicas e geométricas
hmax = 50                       # Altura máxima do tanque (cm)
g = 981                         # Aceleração gravitacional (cm/s²)
r = 0.635                       # Raio da seção transversal do tubo (cm)
rc = 5                          # Raio da seção transversal do tanque (cm)
S = (math.pi)*(r**2)            # Seção transversal do tubo (cm²)
Sc = (math.pi)*(rc**2)          # Seção transversal do tanque (cm²)
beta = S * np.sqrt(2 * g)       # Termo de correção de fluxo baseado em área e gravidade

# Parâmetros iniciais do sistema
h1, h2, h3 = 40, 20, 40         # Níveis iniciais dos tanques (cm)
h0 = 30                         # Nível intermediário entre válvulas A e B (cm)
QP1, QP2 = 80, 80               # Fluxos máximos das bombas (cm³/s)
Kp1, Kp2 = 1.0, 1.0             # Válvulas de entrada abertas
K13, K23 = 1.0, 1.0             # Válvulas de conexão abertas
Ka, Kb = 1.0, 1.0               # Válvulas A e B
K1, K2, K3 = 0.0, 0.0, 1.0      # Válvulas de saída - K1 e K2 fechados, K3 aberto

# Configuração de tempo
dt = 0.1                        # Intervalo de tempo (s)
t_final = 400                   # Tempo total de simulação (s)
t = np.arange(0, t_final, dt)   # Vetor de tempo

# Inicialização dos níveis para armazenamento
h1_vals, h2_vals, h3_vals = [], [], []

# Simulação dinâmica do sistema não linear (método de Euler)
for _ in t:
    # Cálculo dos fluxos
    Qa = Ka * beta * np.sign(h1 - h3) * np.sqrt(abs(h1 - h3))
    Qb = Kb * beta * np.sign(h0 - h3) * np.sqrt(abs(h0 - h3))
    Q13 = K13 * beta * np.sign(h1 - h3) * np.sqrt(abs(h1 - h3))
    Q23 = K23 * beta * np.sign(h2 - h3) * np.sqrt(abs(h2 - h3))
    Q1 = K1 * beta * np.sqrt(abs(h1))
    Q2 = K2 * beta * np.sqrt(abs(h2))
    Q3 = K3 * beta * np.sqrt(abs(h3))

    # Equações diferenciais dos níveis dos tanques
    dh1 = (Kp1 * QP1 - (Qa + Q13 + Q1)) / Sc
    dh2 = (Kp2 * QP2 - (Qb + Q23 + Q2)) / Sc
    dh3 = (Q13 + Q23 + Qa + Qb - Q3) / Sc

    # Atualização dos níveis (método de Euler)
    h1 += dh1 * dt
```

```
    h2 += dh2 * dt
    h3 += dh3 * dt

    # Armazenamento dos resultados
    h1_vals.append(h1)
    h2_vals.append(h2)
    h3_vals.append(h3)

# Plotagem da simulação não linear
plt.figure(figsize=(10, 6))
plt.plot(t, h1_vals, label='h1 (Tanque 1)')
plt.plot(t, h2_vals, label='h2 (Tanque 2)')
plt.plot(t, h3_vals, label='h3 (Tanque 3)')
plt.xlabel('Tempo (s)')
plt.ylabel('Altura (cm)')
plt.title('Simulação Não Linear dos Níveis nos Tanques')
plt.legend()
plt.grid()
plt.show()

# ------------------ ETAPA 2: CÁLCULO DA JACOBIANA E SISTEMA LINEAR ------------------

# Definição simbólica das variáveis
h1, h2, h3 = sp.symbols('h1 h2 h3', real=True, positive=True)

# Fluxos simbólicos
Qa = Ka * beta * sp.sqrt(abs(h1 - h3))
Qb = Kb * beta * sp.sqrt(abs(h0 - h3))
Q13 = K13 * beta * sp.sqrt(abs(h1 - h3))
Q23 = K23 * beta * sp.sqrt(abs(h2 - h3))
Q3 = K3 * beta * sp.sqrt(h3)

# Equações das derivadas dos níveis
h1_dot = (1 / Sc) * (-Qa - Q13)
h2_dot = (1 / Sc) * (-Qb - Q23)
h3_dot = (1 / Sc) * (Q13 + Q23 + Qa + Qb - Q3)

# Cálculo da Jacobiana A
A = sp.Matrix([
    [sp.diff(h1_dot, h1), sp.diff(h1_dot, h2), sp.diff(h1_dot, h3)],
    [sp.diff(h2_dot, h1), sp.diff(h2_dot, h2), sp.diff(h2_dot, h3)],
    [sp.diff(h3_dot, h1), sp.diff(h3_dot, h2), sp.diff(h3_dot, h3)]
])
print("Matriz Jacobiana A:")
sp.pprint(A)

# ------------------ SISTEMA CONTROLADO COM REALIMENTAÇÃO DE ESTADOS ------------------

# Matrizes linearizadas
A_num = np.array([[-0.3566, 0.0, 0.3566],
                  [0.0, -0.2144, 0.2744],
                  [0.3566, 0.2144, -0.5865]])

B_num = np.array([[0.0127, 0.0],
                  [0.0, 0.0127],
                  [0.0, 0.0]])
```

```
# Ponto de operação
x_op = np.array([40, 15, 40])
u_op = np.array([0, -458])
x0 = np.array([40, 20, 40])

# Cálculo do ganho K via alocação de polos
poles = [-0.5, -0.8, -1.0]
K = place_poles(A_num, B_num, poles).gain_matrix
print("Matriz de ganhos K:", K)

# Simulação do sistema linearizado controlado
def sistema_controlado(t, x):
    u = u_op - K @ (x - x_op)
    dxdt = A_num @ (x - x_op) + B_num @ (u - u_op)
    return dxdt

tempo = np.linspace(0, 50, 1000)
solucao = solve_ivp(sistema_controlado, [tempo[0], tempo[-1]], x0, t_eval=tempo)

# Plotagem do sistema controlado
plt.figure(figsize=(10, 6))
plt.plot(tempo, solucao.y[0], label='h1 (Tanque 1)')
plt.plot(tempo, solucao.y[1], label='h2 (Tanque 2)')
plt.plot(tempo, solucao.y[2], label='h3 (Tanque 3)')
plt.axhline(40, color='r', linestyle='--', label='h1op e h3op = 40')
plt.axhline(15, color='g', linestyle='--', label='h2op = 15')
plt.xlabel('Tempo (s)')
plt.ylabel('Altura (cm)')
plt.title('Resposta Linearizada com Realimentação de Estados')
plt.legend()
plt.grid()
plt.show()
```

Etapa 1: Simulação Não Linear

Objetivo

Na **Etapa 1**, foi implementada a **simulação não linear** do sistema de três tanques interligados. O objetivo é observar a evolução dos **níveis dos tanques** ao longo do tempo, considerando os fluxos de entrada e saída e a dinâmica não linear dos tanques.

Metodologia

- **Parâmetros físicos e geométricos**:
  Constantes como gravidade, áreas transversais (S e Sc) e coeficiente de fluxo (β) são definidas com base no modelo físico.
- **Fluxos**:
  Foram considerados os fluxos:
  - **Entre os tanques**: Q13, Q23.
  - **Saídas**: Q1, Q2 e Q3 (válvulas abertas ou fechadas).
  - **Auxiliares**: Qa e Qb entre o tanque intermediário h0.
- **Equações de nível**: As derivadas dos níveis dos tanques são calculadas usando as equações:

```
# Equações de nível
dh1 = (Kp1 * QP1 - (Qa + Q13 + Q1)) / Sc
dh2 = (Kp2 * QP2 - (Qb + Q23 + Q2)) / Sc
dh3 = (Q13 + Q23 + Qa + Qb - Q3) / Sc
```

- **Método de Euler**:
  A atualização dos níveis foi realizada numericamente pelo método de Euler, com um intervalo de tempo dt=0.1 sdt = 0.1 \, \text{s}dt=0.1s.

Resultados

O gráfico apresenta a **evolução temporal** dos níveis h1, h2 e h3 ao longo de 400 segundos. Observa-se a influência dos fluxos entre os tanques e a resposta do sistema às condições iniciais.

Etapa 2: Cálculo da Jacobiana e Linearização

Objetivo

A **Etapa 2** tem como foco a **linearização do sistema** em torno de um **ponto de operação** (xopx_{\text{op}}xop e uopu_{\text{op}}uop). O sistema linearizado permitirá projetar um **controlador** eficiente usando técnicas de realimentação de estados.

Metodologia

1. **Cálculo dos fluxos simbólicos**:
   Usamos a biblioteca **SymPy** para definir simbolicamente os fluxos:

```
Qa = Ka * beta * sp.sqrt(abs(h1 - h3))                  # Fluxo Qa
Qb = Kb * beta * sp.sqrt(abs(h0 - h3))                  # Fluxo Qb
Q13 = K13 * beta * sp.sqrt(abs(h1 - h3))                # Fluxo Q13
Q23 = K23 * beta * sp.sqrt(abs(h2 - h3))                # Fluxo Q23
Q1 = K1 * beta * sp.sqrt(h1)                            # Autovazamento Q1
Q2 = K2 * beta * sp.sqrt(h2)                            # Autovazamento Q2
Q3 = K3 * beta * sp.sqrt(h3)                            # Autovazamento Q3
```

2. **Equações dos estados**:
   A partir das equações diferenciais originais, derivadas parciais são calculadas para determinar os **elementos da matriz Jacobiana**.
3. **Matriz Jacobiana**:
   A Jacobiana AAA foi calculada simbolicamente e substituída no ponto de operação.

Resultados

Os resultados da matriz A e B foram matematicamente encontrados abaixo:

```
A = np.array([[-0.3566, 0.0, 0.3566],
              [0.0, -0.2144, 0.2744],
              [0.3566, 0.2144, -0.5865]])

B = np.array([[0.0127, 0.0],
              [0.0, 0.0127],
              [0.0, 0.0]])
```

Realimentação de Estados e Resposta Linearizada

Objetivo

Utilizar a **realimentação de estados** para estabilizar o sistema linearizado e garantir que os níveis dos tanques h1, h2 e h3 convirjam para os pontos de operação.

Metodologia

1. **Lei de Controle**:
   A realimentação de estados é implementada com a seguinte lei:

$$u = uop - K (x - xop)$$

onde K é a **matriz de ganhos** calculada via **alocação de polos**.

2. **Escolha dos Polos**:
   Foram definidos polos no semiplano esquerdo para garantir **estabilidade** e **rapidez de resposta**:

$$Polos=[-0.5,-0.8,-1.0]$$

3. **Simulação**:
   O sistema linearizado foi simulado usando integração numérica com o solve_ivp, partindo de condições iniciais deslocadas em relação ao ponto de operação.

Resultados

1. **Resposta do Sistema**:
   O gráfico apresenta a **convergência dos níveis** h1, h2 e h3 para os valores de operação:
   - h1=40cm,
   - h2=15cm,
   - h3=40cm.
2. **Especificações de Controle**:
   - **Sobressinal (Overshoot):** Pequeno, indicando uma resposta estável.
   - **Tempo de Subida:** Rápido para atingir 90% do valor final.
   - **Tempo de Acomodação:** Resposta estabilizada dentro de 2% do ponto de operação em poucos segundos.

Conclusão

- A **simulação não linear** forneceu a base para entender o comportamento dinâmico do sistema.

- A **linearização** permitiu simplificar a dinâmica em torno do ponto de operação.
- O **controlador por realimentação de estados** mostrou-se eficaz, garantindo que os níveis dos tanques convergissem rapidamente para os valores desejados.