Documentação do Código para Estimação de Parâmetros no Pêndulo Duplo (Precisão Aprimorada)

Autor: Seu Nome

30 de março de 2025

1 Introdução

Este código tem como objetivo estimar os parâmetros de um pêndulo duplo utilizando uma abordagem de otimização baseada em Algoritmos Genéticos (AG). O modelo físico descreve o comportamento de um pêndulo duplo com alta precisão, e os parâmetros a serem estimados são os comprimentos dos braços ($L1$ e $L2$) e as massas dos pêndulos ($m1$ e $m2$). A simulação é realizada com tolerâncias muito rigorosas, utilizando o método `DOP853` para garantir alta precisão.

Além disso, o código incorpora a aplicação de penalidades quadráticas para garantir que os parâmetros estimados permaneçam dentro de intervalos plausíveis. Os resultados são comparados com os valores reais, e uma análise gráfica permite verificar o desempenho do algoritmo.

2 Estrutura do Código

O código está organizado em quatro partes principais:

2.1 1. Modelo Físico do Pêndulo Duplo (Precisão Aprimorada)

A função `pendulo_duplo` implementa as equações diferenciais do pêndulo duplo. Os principais passos são:

- Cálculo do ângulo de diferença δ entre os dois pêndulos.
- Cálculo dos termos auxiliares, como o cosseno e seno de δ, e dos senos dos ângulos individuais.
- Cálculo dos denominadores `den1` e `den2`, que aparecem na formulação das equações.
- Cálculo das derivadas da velocidade angular (dw1 e dw2) e retorno do vetor das derivadas.

```
def pendulo_duplo(t, y, L1, L2, m1, m2, g):
    theta1, omega1, theta2, omega2 = y

    delta = theta2 - theta1
    cdelta = np.cos(delta)
```

```
    den1 = (m1 + m2)*L1 - m2*L1*cdelta**2
    den2 = (L2/L1)*den1

    # C lculos otimizados
    sd, cd = np.sin(delta), cdelta
    st1, st2 = np.sin(theta1), np.sin(theta2)

    # Equa  es diferenciais vetorizadas
    dw1 = (m2*L1*omega1**2*sd*cd + m2*g*st2*cd +
           m2*L2*omega2**2*sd - (m1 + m2)*g*st1) / den1
    dw2 = (-m2*L2*omega2**2*sd*cd + (m1 + m2)*g*st1*cd -
           (m1 + m2)*L1*omega1**2*sd - (m1 + m2)*g*st2) / den2

    return [omega1, dw1, omega2, dw2]
```

2.2 2. Configuração de Simulação de Alta Precisão

Nesta seção, são definidos os parâmetros reais do sistema, o intervalo de tempo e as condições iniciais para a simulação:

- Parâmetros reais: $L1_{\text{real}} = 1.0$, $L2_{\text{real}} = 0.8$, $m1_{\text{real}} = 1.0$, $m2_{\text{real}} = 0.5$ e $g_{\text{real}} = 9.81$.
- Intervalo de tempo: 15 segundos, discretizado em 2000 pontos.
- Condição inicial: ângulos iniciais maiores para melhor observação.

A simulação de referência é realizada com o método `DOP853` e tolerâncias muito rigorosas:

```
L1_real, L2_real = 1.0, 0.8
m1_real, m2_real = 1.0, 0.5
g_real = 9.81

t_data = np.linspace(0, 15, 2000)  # 15 segundos, 2000 pontos
y0 = [np.pi/3, 0, np.pi/4, 0]  #  ngulos  iniciais

sol_real = solve_ivp(pendulo_duplo, [0, 15], y0,
                     args=(L1_real, L2_real, m1_real, m2_real, g_real),
                     t_eval=t_data, method='DOP853',
                     rtol=1e-10, atol=1e-10)
```

Para simular dados reais, é adicionado ruído controlado (1%) aos ângulos:

```
theta1_real = sol_real.y[0] + np.random.normal(0, 0.01, len(t_data))
theta2_real = sol_real.y[2] + np.random.normal(0, 0.01, len(t_data))
```

2.3 3. Algoritmo Genético Aprimorado

Utilizando a biblioteca DEAP, o código configura um algoritmo genético para estimar os parâmetros $L1$, $L2$, $m1$ e $m2$. Principais etapas:

- Criação das classes `FitnessMin` e `Individual`.
- Registro dos atributos com intervalos restritos baseados em conhecimento prévio.
- Definição dos operadores genéticos (cruzamento, mutação e seleção) com parâmetros refinados.

A função de fitness calcula o erro entre a simulação (focando nos primeiros 4 segundos) e os dados reais, aplicando penalidades quadráticas para forçar os parâmetros a permanecerem dentro dos limites esperados:

```python
def fitness(individual):
    L1, L2, m1, m2 = individual
    g = 9.81

    penalty = (max(0, abs(L1-1.0)-0.1)*100)**2 + \
              (max(0, abs(L2-0.8)-0.08)*100)**2 + \
              (max(0, abs(m1-1.0)-0.15)*100)**2 + \
              (max(0, abs(m2-0.5)-0.05)*100)**2

    try:
        t_eval_fit = np.linspace(0, 4, 400)
        sol = solve_ivp(pendulo_duplo, [0, 4], y0, args=(L1, L2, m1, m2, g),
                        t_eval=t_eval_fit, method='DOP853',
                        rtol=1e-8, atol=1e-8)

        error = 0.7*np.mean((sol.y[0] - theta1_real[:400])**2) + \
                0.3*np.mean((sol.y[2] - theta2_real[:400])**2)

        return error + penalty,
    except:
        return 1e10,
```

O algoritmo é executado com uma população de 300 indivíduos, 500 gerações, e utiliza estatísticas e um Hall of Fame para registrar os melhores resultados.

2.4 4. Visualização e Análise dos Resultados

Após a execução do algoritmo genético, a solução obtida é utilizada para simular o sistema e comparar com os dados reais. Principais pontos:

- São exibidas as curvas dos ângulos θ_1 e θ_2 reais e os estimados pelo AG, focando nos primeiros 8 segundos da simulação.
- A plotagem inclui legendas que informam os valores reais e os valores estimados dos parâmetros.
- Os resultados finais são também impressos no terminal em formato tabular, detalhando o erro percentual para cada parâmetro.

Trechos de código para a visualização:

```python
sol_ag = solve_ivp(pendulo_duplo, [0, 15], y0,
                   args=(*best_ag, 9.81), t_eval=t_data, method='DOP853',
                   rtol=1e-10, atol=1e-10)

plt.figure(figsize=(16, 8))
plt.plot(t_data[:800], theta1_real[:800], 'b-',
         label=f"Real 1 (L1={L1_real:.4f}, m1={m1_real:.4f})", alpha=0.7)
plt.plot(t_data[:800], theta2_real[:800], 'g-',
         label=f"Real 2 (L2={L2_real:.4f}, m2={m2_real:.4f})", alpha=0.7)
```

```
plt.plot(t_data[:800], sol_ag.y[0][:800], 'r--',
         label=f"AG  1 (L1={best_ag[0]:.4f}, m1={best_ag[2]:.4f})", linewidth=2)
plt.plot(t_data[:800], sol_ag.y[2][:800], 'm--',
         label=f"AG  2 (L2={best_ag[1]:.4f}, m2={best_ag[3]:.4f})", linewidth=2)

plt.xlabel("Tempo (s)", fontsize=12)
plt.ylabel(" ngulo  (rad)", fontsize=12)
plt.title("Ajuste de Par metros do P ndulo Duplo (Primeiros 8s)\nErro  m dio:  1  = {:.2e},  2  = {:.2e}".format(
          np.mean((sol_ag.y[0][:400]-theta1_real[:400])**2),
          np.mean((sol_ag.y[2][:400]-theta2_real[:400])**2)), fontsize=14)
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.show()

print("\n=== Resultados de Alta Precis o ===")
print(f"| {'Par metro':<8} | {'Valor Real':<12} | {'AG Estimado':<12} | {'Erro %':<10} | {'Limites':<12} |")
print("|----------|--------------|--------------|------------|------------|")
print(f"| L1       | {L1_real:<12.4f} | {best_ag[0]:<12.4f} | {abs(best_ag[0]-L1_real)/L1_real*100:<9.4f}% | [0.9, 1.1]   |")
print(f"| L2       | {L2_real:<12.4f} | {best_ag[1]:<12.4f} | {abs(best_ag[1]-L2_real)/L2_real*100:<9.4f}% | [0.72, 0.88] |")
print(f"| m1       | {m1_real:<12.4f} | {best_ag[2]:<12.4f} | {abs(best_ag[2]-m1_real)/m1_real*100:<9.4f}% | [0.85, 1.15] |")
print(f"| m2       | {m2_real:<12.4f} | {best_ag[3]:<12.4f} | {abs(best_ag[3]-m2_real)/m2_real*100:<9.4f}% | [0.45, 0.55] |")
```

3 Conclusão

Este código integra técnicas de simulação de alta precisão e otimização via algoritmos genéticos para a estimação dos parâmetros do pêndulo duplo. A abordagem empregada demonstra que, mesmo em sistemas complexos e com comportamento potencialmente caótico, é possível recuperar parâmetros com elevada exatidão, desde que se delimite adequadamente o espaço de busca e se apliquem penalizações consistentes. A documentação apresentada visa fornecer uma compreensão detalhada da estrutura e do funcionamento do código, facilitando a manutenção e a replicação dos experimentos, bem como a extensão da metodologia para outros sistemas dinâmicos.