Trabalho Computacional 2 | ONL

👥 Alunos:

- Felippe Veloso Marinho, 2021072260
- Otávio Augusto Bernardes Rodrigues, 2024421606

Introdução

Este relatório apresenta uma análise comparativa entre três métodos clássicos de otimização com restrições aplicados ao problema da treliça de três barras: Penalidade Exterior, Penalidade Interior (Barreira) e Lagrangeano Aumentado. O objetivo do problema é minimizar o peso da treliça, respeitando restrições de tensão e de limites das variáveis.

Questão 1

Seja a treliça de três barras mostrada na Figura 1. O comprimento de cada barra é denotada por A1, A2 e A3 e, neste problema, iremos assumir que o comprimento das duas barras exteriores são iguais (isto é A1 = A3). O espaçamento horizontal entre as barras e a distância até a ponta serão denotadas por uma mesma constante H. O parâmetro P denota a força-peso da carga que é colocada na treliça e é constante no nosso problema.

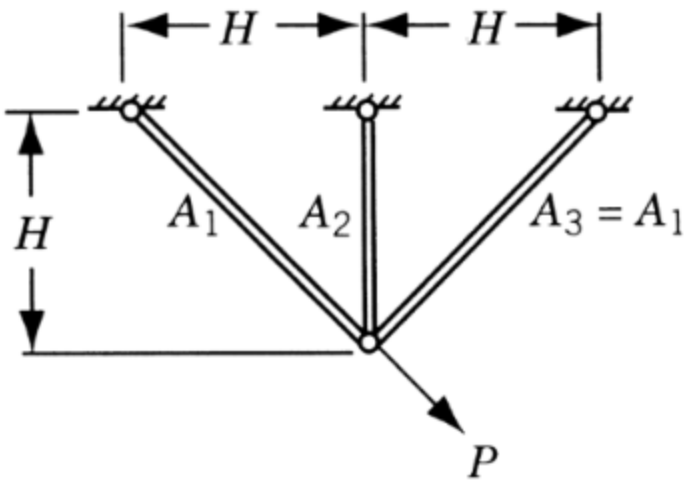


Figura 1: Treliça de três barras

$$\min \quad f(x_1, x_2) = (2\sqrt{2})x_1 + x_2 \tag{1a}$$

$$\text{sujeito a : } P\frac{x_2 + x_1\sqrt{2}}{x_1^2\sqrt{2} + 2x_1x_2} \leq 20 \tag{1b}$$

$$P\frac{1}{x_1 + x_2\sqrt{2}} \leq 20 \tag{1c}$$

$$-P\frac{x_2}{x_1^2\sqrt{2} + 2x_1x_2} \leq -5 \tag{1d}$$

$$0.1 \leq x_1, x_2 \leq 5 \tag{1e}$$

Figura 2: Problema de minimização

Sendo P =20. A partir das informações será aplicado os três métodos de otimização restrita.

Lagrangeano Aumentado

Assim como os métodos de penalidade, o ALM transforma o problema restrito em um problema irrestrito. Adicionando as restrições de igualdade e desigualdade à função objetivo.

$$p(\boldsymbol{x}) = r^h \sum_{k=1}^{m} \lambda_k [h_k(\boldsymbol{x})]^2 + r^g \sum_{j=1}^{l} [\max\{g_j(\boldsymbol{x}), -\frac{\beta_j}{2r^g}\}]^2 + \dots + \sum_{k=1}^{m} \lambda_k h_k(\boldsymbol{x}) + \sum_{j=1}^{l} max\{g_j(\boldsymbol{x}), -\frac{\beta_j}{2r^g}\} \quad (4.10)$$

Método da Barreira ou da Penalidade interna

O método de barreira possui uma estrutura que pode ser expressa da seguinte forma:

- minimize f(x) + b(x)

onde b(x) =

$$b(\boldsymbol{x}) = -r^g \sum_{i=1}^{l} \frac{1}{g_i(\boldsymbol{x})}$$

Se alguma g(x)≥0, a função −log(−g(x)) **explode** (vai para ∞) ou recebe um valor enorme artificialmente (ex: 1e10).

Método da Penalidade Externa

$$p(\boldsymbol{x}) = r^h \sum_{j=1}^{m} h_j(\boldsymbol{x})^2 + r^g \sum_{i=1}^{l} (\max\{0, g_i(\boldsymbol{x})\})^2$$

Onde:

São as restrições de igualdade.

$$p(\boldsymbol{x}) = r^h [h_j(\boldsymbol{x})]^2$$

São as restrições de desigualdade.

$$p(\boldsymbol{x}) = r^g[\max\{0, g_i(\boldsymbol{x})\}]^2$$

Ou seja, o método penaliza apenas **violações** das restrições com termos do tipo (max(0,g(x)))^2

Utilizando o método do Langrangeano Aumentado, seguimos com a seguinte implementação:

```
def lagrangeano_aumentado(x, mu, u):
    x1, x2, = x
    f = (2*raiz2) * x1 + x2

    g = np.zeros(7)
    # g1: P*(x2 + x1*sqrt2)/(x1^2*sqrt2 + 2*x1*x2) - 20 <= 0
    g[0] = P*(x2 + x1*raiz2)/(x1**2*raiz2 + 2*x1*x2) - 20
    # g2: P/(x1 + x2*sqrt2) - 20 <= 0
    g[1] = P/(x1 + x2*raiz2) - 20
    # g3: 5 - P*x2/(x1^2*sqrt2 + 2*x1*x2) <= 0  ⇒  P*x2/(...) - 5 >= 0
    g[2] = 5 - (P*x2/(x1**2*raiz2 + 2*x1*x2))
    # Limites x1 >= 0.1, x1 <= 5, x2 >= 0.1, x2 <= 5
    g[3] = 0.1 - x1
    g[4] = x1 - 5
    g[5] = 0.1 - x2
    g[6] = x2 - 5

    return f + np.sum(mu*g) + 0.5*u * np.sum(np.maximum(g, 0 )**2)
```

Com base nos resultados obtidos utilizando o ponto inicial recomendado pelo enunciado (1.0,3.0)(1.0, 3.0)(1.0,3.0), a solução encontrada foi:

- x*=[0.7888, 0.4083] | f(x*)≈2.639

- Todas as restrições g1g_1g1 a g7g_7g7 foram respeitadas:
  - g1=−0.0023

- g2=−5.36
- g3=−0.36
- g4=−0.68
- .....

A solução é **viável ou quase viável**, com todas as restrições satisfeitas. Observa-se que a **restrição g1g_1g1** está muito próxima de ser **ativa**, ou seja, está na **fronteira da região factível**.

Entre os multiplicadores de Lagrange atualizados (μi\mu_iμi), apenas:

- **μ1 = 0,0729** foi positivo.

As demais permaneceram nulas, o que é esperado no método, pois:

> No método dos multiplicadores de Lagrange aumentado, os multiplicadores associados a restrições não ativas tendem a permanecer nulos. Isso ocorre porque a atualização de μi só acontece de fato quando gi(x)≥0, ou seja, a restrição está ativa ou quase ativa. Como as demais restrições apresentaram folgas negativas significativas, seus multiplicadores não foram atualizados.

Foram utilizados outros pontos para exemplificar melhor a atuação do método em relação aos outros.

(5, 5)

Iterações realizadas: 4

```
x* = [ 0.01581799 -0.02273542]
f(x*) = 0.02200460402161534
g(x*): [ 6.56984089e-05 -1.24438313e+03 -1.23938320e+03  8.41820114e-02
 -4.98418201e+00  1.22735424e-01 -5.02273542e+00]
mu: [1.47821420e-04 0.00000000e+00 0.00000000e+00 3.11221547e-01
```

```
 0.00000000e+00 6.53883595e-01 0.00000000e+00]
Penalidade final u: 3.375
```

Apesar de f(x) ser menor, duas restrições foram violadas. g6 = +0.1227 e g4 = +0.0841 → duas violações positivas ⇒ solução inviável.

(0.2, 0.2)

Iterações realizadas: 30

```
x* = [0.78879641 0.40790537]
f(x*) = 2.638958536455129
g(x*): [ 4.05413871e-08 -5.35508483e+00 -3.55084874e-01 -6.88796412e-01
 -4.21120359e+00 -3.07905370e-01 -4.59209463e+00]
mu: [0.12913977 0.         0.         0.         0.         0.
 0.        ]
Penalidade final u: 127834.03948858939
```

O resultado foi similar ao ponto inicial porém com uma penalidade final muito maior (u = 127.834). O número de iterações foi de 30, ou seja, houve uma demora muito maior para convergir. Isso mostra que esse ponto, mesmo que na região viável, está mais distante da solução ideal.

Para os outros dois métodos de penalidades, foram realizados testes com os 3 mesmos pontos iniciais utilizados no método anterior.

A implementação de ambos são semelhantes em exceção ao loop onde no método da barreira utilizamos a função de barreira e a função de penalização.

```
from scipy.optimize import minimize
import numpy as np

# Função objetivo
def objetivo(x):
    x1, x2 = x
    return 2*np.sqrt(2)*x1 + x2

# Restrições
```

```
def restricoes(x):
    x1, x2 = x
    raiz2 = np.sqrt(2)
    P = 20
    g = np.zeros(7)
    g[0] = P*(x2 + x1*raiz2)/(x1**2*raiz2 + 2*x1*x2) - 20
    g[1] = P/(x1 + x2*raiz2) - 20
    g[2] = 5 - (P*x2/(x1**2*raiz2 + 2*x1*x2))
    g[3] = 0.1 - x1
    g[4] = x1 - 5
    g[5] = 0.1 - x2
    g[6] = x2 - 5
    return g
```

```
# Loop com função penalizadora
for k in range(max_iter):
    def func_penalizada(x):
        g_val = restricoes(x)
        penal = np.sum(np.maximum(g_v
        return objetivo(x) + u * penal

    res = minimize(func_penalizada, x_
    x_novo = res.x

    if np.linalg.norm(x_novo - x_ant)/(1
        x_atual = x_novo
        break

    x_ant = x_novo
    x_atual = x_novo
    u *= alpha
```

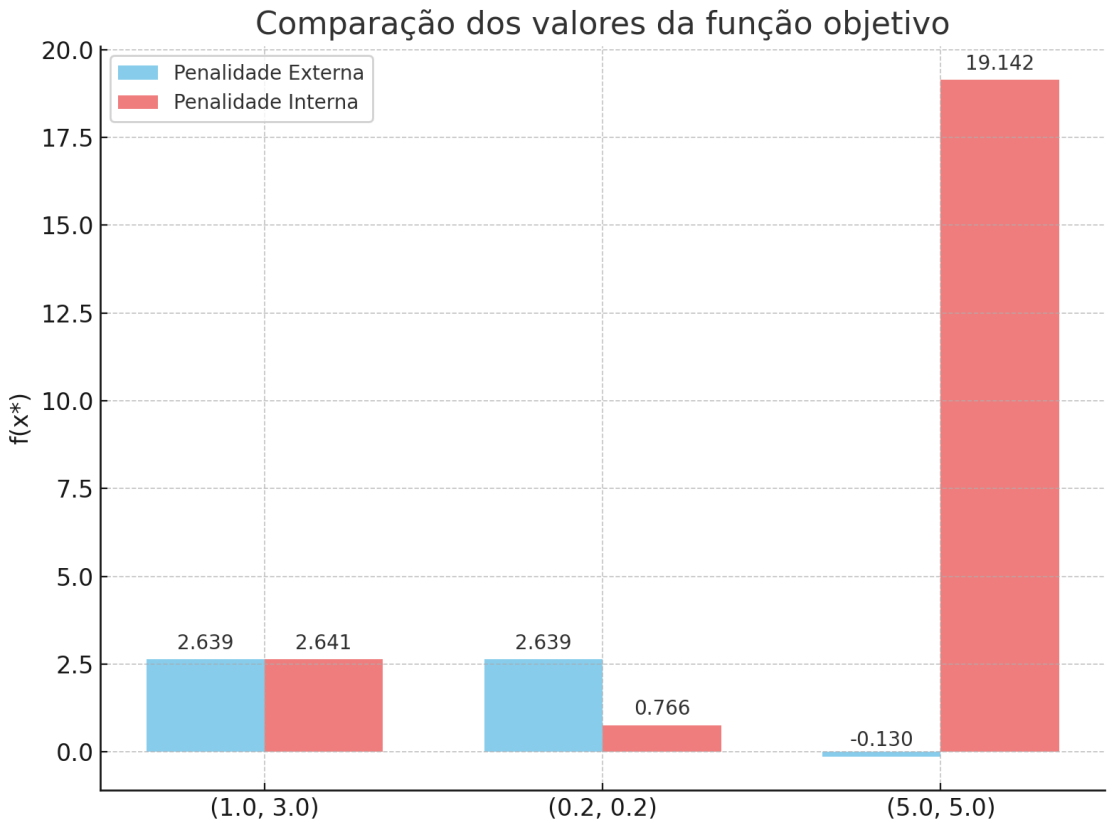
```
# Loop com função de barreira
for k in range(max_iter):
    def func_barreira(x):
        g_val = restricoes(x)
        if np.any(g_val >= 0):
            return 1e10
        return objetivo(x) - u * np.sum(n

    res = minimize(func_barreira, x_atu
    x_novo = res.x

    if np.linalg.norm(x_novo - x_ant)/(1
        x_atual = x_novo
        break

    x_ant = x_novo
    x_atual = x_novo
    u *= alpha_interna
```

Os resultados obtidos nos dois métodos apresentaram convergência em valores próximos nos 3 pontos iniciais. O ponto (5, 5) foi o que apresentou maior diferença entre os valores de penalidades externas e internas. Provavelmente isso se deve indicar que a função de barreira sofre muito nesse ponto inicial, especialmente devido à estrutura logarítmica.



Questão 2

Introdução

Neste trabalho, resolvemos o problema para **três unidades geradoras** com funções de custo quadráticas e considerando **perdas de potência** modeladas por uma matriz de coeficientes BBB. Foram utilizados dois métodos de otimização restrita estudados na disciplina: **Penalidade Exterior** e **Lagrangeano Aumentado**.

A implementação foi feita na linguagem **Python**, utilizando a **biblioteca otimo.py** fornecida pelo professor, que contém implementações dos métodos de otimização vistos em sala de aula.

Queremos resolver o seguinte problema de otimização:

Função Objetivo (custo total):

$minf(P1, P2, P3) = C1(P1) + C2(P2) + C3(P3)$

Com:

$C1(P1) = 0,15P12 + 38P1 + 736$

$C2(P2) = 0,10P22 + 46P2 + 451$

$C3(P3) = 0,25P32 + 40P1 + 1049$

Restrição de Igualdade (demanda + perdas):

$P1 + P2 + P3 = 850 + PL$

Perdas de Potência (PL):

$$PL = \sum_{i=1}^{3} \sum_{j=1}^{3} P_i B_{ij} P_j$$

com matriz B dada por:

B = 0,000049 0,000014 0,000015 0,000014 0,000045 0,000016 0,000015 0,000016 0,000039

Limites de Geração:

$150 \leq P1 \leq 600$

$100 \leq P2 \leq 400$

$50 \leq P3 \leq 200$

Implementação

Nesta tarefa foram usadas 2 Bibliotecas a Numpy e ótimo.py

```
# Função objetivo:
def custo_total(P):
    P1, P2, P3 = P
```

```
    return (
        0.15 * P1**2 + 38 * P1 + 756 +
        0.1 * P2**2 + 46 * P2 + 451 +
        0.25 * P3**2 + 40 * P3 + 1049
    )
# Matriz de Perdas e Demandas
B = np.array([
    [0.000049, 0.000014, 0.000015],
    [0.000014, 0.000045, 0.000016],
    [0.000015, 0.000016, 0.000039]
])
D = 850
# Perda de potencia:
def perda_potencia(P):
    return P @ B @ P

# Restrições:
def restricao_demanda(P): return np.sum(P) - D - perda_potencia(P)
def r1(P): return P[0] - 150
def r2(P): return P[1] - 100
def r3(P): return P[2] - 50
def r4(P): return 600 - P[0]
def r5(P): return 400 - P[1]
def r6(P): return 200 - P[2]

# Agrupamento das restrições e especificações de cada método, está etapa é ne
restricoes = [restricao_demanda, r1, r2, r3, r4, r5, r6]
tipos_penalidade = np.array(['>', '>', '>', '>', '>', '>', '>'])
tipos_lagrangeano = np.array(['=', '>', '>', '>', '>', '>', '>'])

# Parâmetros de inicialização:
x0 = np.array([300.0, 300.0, 150.0])
busca_1d = SecaoAurea(precisao=1e-6)
irrestrito = GradienteConjugado(busca_1d, precisao=1e-6)
# Metodos de otimização
# Penalidade exterior
```

```
def resolver_penalidade_exterior():
    metodo = PenalidadeExterior(precisao=1e-6)
    sol = metodo.resolva(
        custo_total, x0, restricoes, tipos_penalidade,
        irrestrito, penalidade=1.0, aceleracao=2.0
    )
    return sol
# Lagrangeano aumentado:
def resolver_lagrangeano_aumentado():
    metodo = LagrangeanoAumentado(precisao=1e-6)
    sol = metodo.resolva(
        custo_total, x0, restricoes, tipos_lagrangeano,
        irrestrito, penalidade=1.0, aceleracao=2.0
    )
    return sol
# Execução e resultados
print("=== Método 1: Penalidade Exterior ===")
res1 = resolver_penalidade_exterior()
P1 = res1.x
print("P =", P1)
print("Custo total =", custo_total(P1))
print("Perda =", perda_potencia(P1))
print("Soma das potências =", np.sum(P1))
print("Demanda + perda =", D + perda_potencia(P1))

print("\n=== Método 2: Lagrangeano Aumentado ===")
res2 = resolver_lagrangeano_aumentado()
P2 = res2.x
print("P =", P2)
print("Custo total =", custo_total(P2))
print("Perda =", perda_potencia(P2))
print("Soma das potências =", np.sum(P2))
print("Demanda + perda =", D + perda_potencia(P2))
```

Resultados

```
# === Método 1: Penalidade Exterior ===
P = [294.89018536 399.26406173 175.57573583]
Custo total = 75542.92538094848
Perda = 19.730023098416886
Soma das potências = 869.7299829141984
Demanda + perda = 869.7300230984168

# === Método 2: Lagrangeano Aumentado ===
P = [294.87820089 399.29121844 175.56088855]
Custo total = 75542.93033791147
Perda = 19.730307958824337
Soma das potências = 869.7303078830519
```

Conclusão

O problema de despacho econômico foi resolvido por meio dos métodos da **Penalidade Exterior** e do **Lagrangeano Aumentado**, ambos implementados com a biblioteca otimo.py. As soluções obtidas por ambos os métodos apresentaram resultados semelhantes em termos de custo total, alocação de potência entre as unidades geradoras e atendimento à demanda acrescida das perdas de transmissão.