Trabalho Computacional 1 | ONL

Disciplina:

ELE077 – Otimização Não Linear

Universidade Federal de Minas Gerais – Escola de Engenharia

Aluno:

- Felippe Veloso Marinho, 2021072260

Introdução

Este relatório tem como objetivo colocar em prática os métodos de resolução de problemas irrestritos dados em sala de aula. Cada problema envolve a aplicação de métodos de otimização não linear a diferentes cenários práticos, incluindo controle de sistemas, regressão logística, maximização de lucros em campanhas promocionais e modelagem térmica. Neste momento, não serão apresentadas implementações ou resultados numéricos. Em vez disso, será estabelecida uma base teórica e estrutural para guiar a implementação futura dos algoritmos de otimização.

Questão 1

Neste problema, busca-se ajustar os parâmetros Kp, Ki e Kd de um controlador PID aplicado ao controle da temperatura de um forno industrial. O sistema é modelado por uma equação diferencial de primeira ordem, e o objetivo é minimizar o **erro quadrático médio (MSE)** da diferença entre a temperatura real T(t) e a temperatura de referência Tref ao longo de um intervalo de simulação.

A função-objetivo foi previamente implementada nos arquivos disponíveis e pode ser definida como:

$$\min_{K_p,K_i,K_d} \int_0^{T_{\text{sim}}} (T_{\text{ref}} - T(t, K_p, K_i, K_d))^2 \, dt$$

A dinâmica do sistema é dada pela EDO:

$$\frac{dT}{dt} = -\frac{T}{\tau} + \frac{K}{\tau}u(t)$$

Para obter o ajuste de ganhos foi feita a escolha da comparação dos métodos de **Gradiente Conjulgado** e o BFGS.

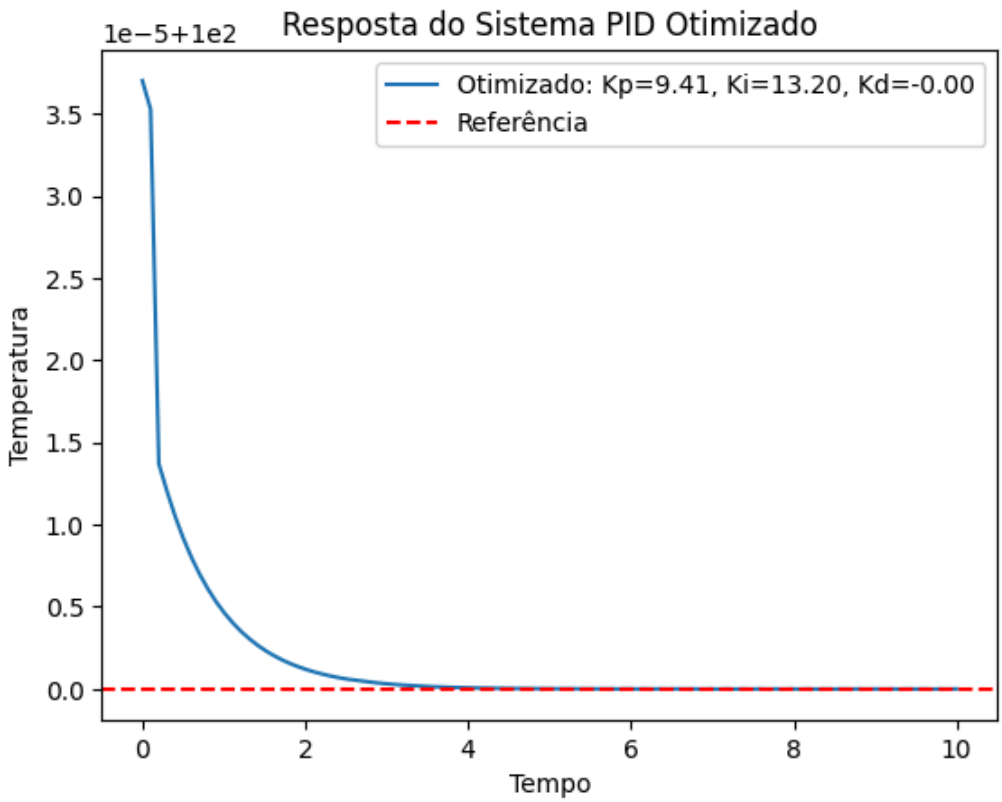
Para completar o código, foi utilizado a biblioteca `scipy.optimize.minimize` . A partir dele, foram extraídos os resultados passando a função objetivo pré implementada, o ponto_inicial e o método escolhido.

Foram escolhidos os métodos de BFSG e **Gradiente Conjulgado**.

> Parâmetros PID otimizados com CG e BFSG:
> Kp = 9.4084, Ki = 13.2000, Kd = -0.0000
> Erro quadrático médio (MSE): 0.000000

Ambos os métodos convergiram para a mesma solução ótima com precisão numérica (MSE ≈ 0), o que pode indicar que a função objetivo é suave e bem comportado e o problema tem um mínimo global bem definido e acessível a partir do ponto inicial.

O fato de Kd ser aproximadamente igual a 0 sugere que o **termo derivativo não foi necessário** para alcançar um bom desempenho neste sistema específico. Ou seja, a escolha do método do gradiente acaba sendo mais adequada nesse problema por ser uma alternativa viável e mais leve pensando na não necessidade do uso da Hessiana.



Questão 2

Este problema aborda a minimização da função de custo da regressão logística regularizada, aplicada a um conjunto de músicas rotuladas como sucesso (1) ou não (0). O vetor de características de cada música possui 500 atributos.

A função a ser minimizada é a entropia cruzada regularizada:

$$J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h(\mathbf{x}_i;\mathbf{w})) + (1-y_i)\log(1-h(\mathbf{x}_i;\mathbf{w}))] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

com:

$$h(\mathbf{x}_i;\mathbf{w}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}_i}}$$

Saída do modelo logístico (função sigmoide)

O objetivo é encontrar o vetor de pesos w que minimiza a função de custo J(w) utilizando o conjunto de dados fornecidos com m = 1000 amostras e n = 500 características.

Por ser um problema de grande dimensão, um método que apresenta uma boa escalabilidade seria o ideal. Para isso, foi feita a escolha do **Gradiente Conjulgado** novamente.

O tempo de execução foi de 28m e 14s com o custo final (função J(w)) de 0.0924, 5 iterações e 8517 avaliações da função-objetivo.

O valor da função de custo indica que o modelo está com bom desempenho visto que como a função inclui **entropia cruzada + regularização L2**, valores baixos (próximos de 0) indicam alta probabilidade correta para os rótulos e sem overfitting.

Entretanto, nessa primeira análise o número de avaliações foi altíssimo para tão poucas interações. Uma sugestão é que o método está usando muitos passos pequenos com alta precisão.

Pesquisando sobre o uso da função minimize da biblioteca do scipy, foi visto que é possível a adição de um argumento com o gradiente da função. Isso é feito para reduzir o número de chamadas.

Isto feito, o número de avaliações da função-objetivo caiu de 8517 para 17. O que consequentemente diminuiu o tem de execução de aproximadamente 28m e 14s para 7.8s.

Porque isso aconteceu?

O uso do `jac=gradiente` reduz o custo da otimização **de forma exponencial** em alta dimensão, pois evita milhares de avaliações repetitivas.

Sem o uso do gradiente, para cada iteração é estimado

$\partial J/\partial wi \approx J(wi + \epsilon) - J(wi)/\epsilon$ para cada um dos 500 pesos. O que significa que no mínimo 500 chamadas à função objetivo são necessárias por interação. Ao fornecer o gradiente, o algoritmo usa diretamente o gradiente vetorizado sem estimar nada numericamente. Cada iteração passa a exigir uma chamada da função objetivo + uma chamada do gradiente, sendo muito mais eficiente.

Avaliando o modelo de regressão

O custo final apresentado, como dito anteriormente, indica alta probabilidade de acerto do modelo. O cálculo da acurácia demonstra que o modelo classifica corretamente quase todas as amostras (98,27%). Isso sugere que o modelo aprendeu bem o padrão do conjunto de dados.

A matriz de confusão demonstrou os seguintes resultados:

0	173
0	9827

- **TN (verdadeiros negativos)** = 0
- **FP (falsos positivos)** = 173
- **FN (falsos negativos)** = 0
- **TP (verdadeiros positivos)** = 9827

O que foi um resultado um pouco preocupante. Há um desequilíbrio de classes, todos os exemplos de rótulo 0 forma classificados incorretamente como positivos. Indicando que o modelo prevê 1 mesmo quando deveria ser 0.

Depois do susto inicial, o conjunto de dados foi verificado e realmente a maior parte dos rótulos está para a classe 1. O que novamente mostra que o problema não está no ajuste dos pesos.

```
Classe 0: 173 amostras
Classe 1: 9827 amostras
Classe 0 representa 1.73% dos dados
Classe 1 representa 98.27% dos dados
```

Estratégias como oversampling ou undersamplig poderiam ser aplicadas mas fogem dos requisitos da questão.

Questão 3

Aqui, o objetivo é maximizar o lucro de uma campanha promocional ajustando os parâmetros: desconto d, tempo de duração t, e orçamento de marketing m. O lucro é modelado como:

$$L(d,t,m) = R(d,t,m) - C(d,t,m)$$

onde

$$R(d,t,b) = Vb * (1 + f1(d) + f2(t)) * log(1 + b)$$

- VB é o número de vendas sem promoção.
- f1(d) = −0.005d^2 + 0.2d é o incremento percentual nas vendas devido ao desconto.
- f2(t) = 0.05t é o incremento nas vendas proporcional ao tempo da promoção

$$C(d, t, m) = Cb + Cm(m) + P(d, t, m)$$

- CB é o custo fixo inicial.
- CM(m) = m é o custo de marketing.
- P(d, t) é uma penalização não contínua:
  - Penalização de R$5000 se d > 30% (desconto muito agressivo).
  - Penalização de R$2000 se t > 15 dias.

E importante frisar que:

- O desconto não pode ser menor que 0% nem maior que 50% (0 ≤ d ≤ 50).
- O tempo de duração da promoção não pode ser menor que 1 dia nem maior que 30 dias (1 ≤ t ≤ 30).
- O orçamento de marketing não pode ser menor que R$1000 nem maior que R$50000 (1000 ≤ m ≤ 50000).

A receita depende de funções não lineares dos parâmetros e o custo inclui penalizações descontínuas. Além disso, há restrições nas variáveis de decisão que devem ser tratadas como penalizações.

Sendo assim, a utilização de um método de resolução de problemas restritos acaba sendo utilizado, o método da penalização externa.

A implementação das funções e das penalidades assim como dito no enunciado foram feitas da seguinte forma:

```
# Função de custo negativa (para maximização do lucro)
def funcaoobjetivo(x):
    d, t, m = x  # Desconto (%), tempo (dias), orçamento (R$)
    VB = 100000  # Vendas básicas
    CB = 10000   # Custo fixo inicial
```

```
    # Receita
    f1 = -0.005 * d**2 + 0.2 * d
    f2 = 0.05 * t
    receita = VB * (1 + f1 + f2) * np.log(1 + m)

    # Custo
    custo_marketing = m
    penalidades = 0

    # Penalizações específicas do problema
    if d > 30:
        penalidades += 5000
    if t > 15:
        penalidades += 2000

    # Restrições explícitas transformadas em penalizações grandes
    if not (0 <= d <= 50):
        penalidades += 1e6
    if not (1 <= t <= 30):
        penalidades += 1e6
    if not (1000 <= m <= 50000):
        penalidades += 1e6

    custo_total = CB + custo_marketing + penalidades

    # Lucro
    lucro = receita - custo_total
```

A escolha de um algoritmo foi feita com base no pequeno número de variáveis, a não existência de derivadas e as penalidades serem contínuas.

Para isso, a escolha do “Nelder-Mead” é adequada, pensando que é um método sem derivadas, robusto para problemas não suaves.

O algoritmo convergiu para os seguintes parâmetros ótimos:

- Desconto = 20.00%,

- Tempo = 30.00 dias,
- Orçamento = R$49949.02
- Lucro máximo estimado: R$4806501.13

Os valores encontrados respeitam todas as restrições impostas e evitam penalizações, mostrando a eficácia da abordagem de penalização externa aliada ao método de busca heurística sem gradientes.

Questão 4

O último problema propõe a minimização de uma função quadrática de duas variáveis x1 e x2, que representam temperaturas em uma aleta unidimensional. A função a ser minimizada é:

$$f(x1, x2) = 0.6382x1^2 + 0.3191x2^2 - 0.2809x1x2 - 67.906x1 - 14.29x2$$

A implementação da função-objetivo é a seguinte:

```
# Função-objetivo
def funcaoobjetivo(x):
    x1, x2 = x
    return 0.6382 * x1**2 + 0.3191 * x2**2 - 0.2809 * x1 * x2 - 67.906 * x1 - 14.29
```

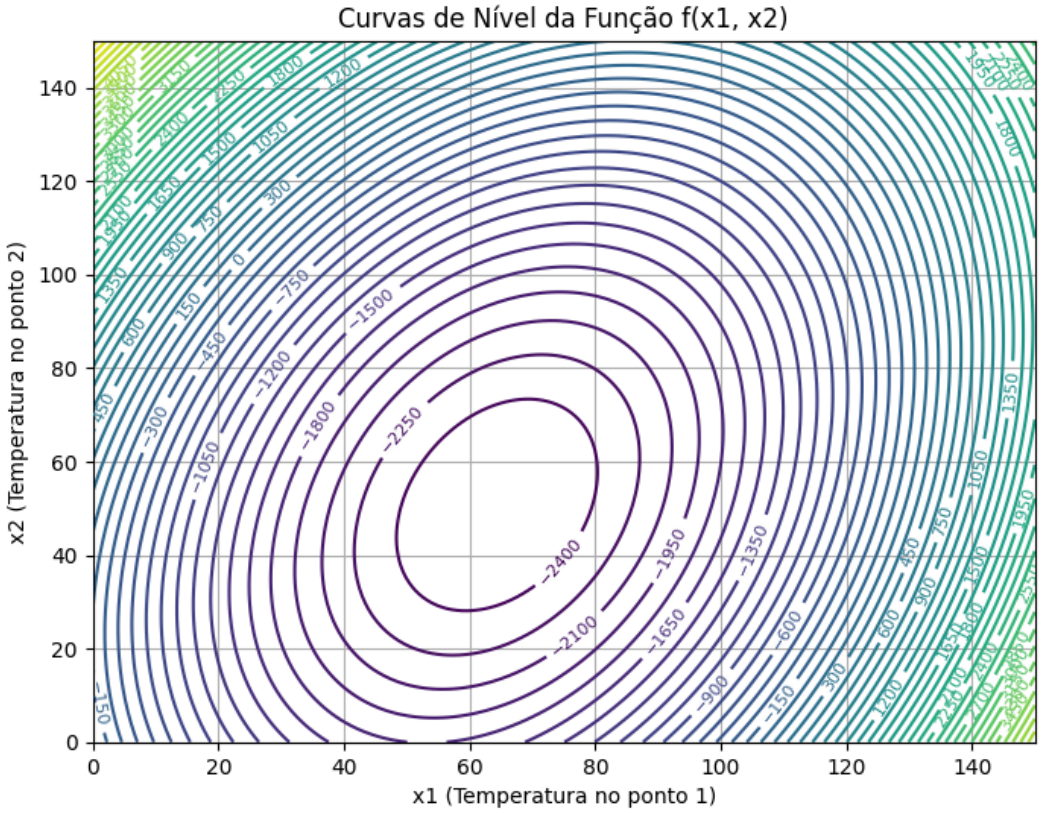
```
# Grade para x1 e x2
x1 = np.linspace(0, 150, 200)
x2 = np.linspace(0, 150, 200)
X1, X2 = np.meshgrid(x1, x2)

# Calcular os valores de f(x1, x2)
Z = 0.6382 * X1**2 + 0.3191 * X2**2 - 0.2809 * X1 * X2 - 67.906 * X1 - 14.29 * X2

# Plot das curvas de nível
plt.figure(figsize=(8, 6))
contours = plt.contour(X1, X2, Z, levels=50, cmap='viridis')
plt.clabel(contours, inline=True, fontsize=8)
plt.title("Curvas de Nível da Função f(x1, x2)")
```

```
plt.xlabel("x1 (Temperatura no ponto 1)")
plt.ylabel("x2 (Temperatura no ponto 2)")
plt.grid(True)
plt.show()
```

A função á ser otimizada é uma função convexa unimodal que pode ser representada pelas curvas de nível abaixo:



Tendo isso em vista, os algoritmos á serem escolhidos devem levar em consideração que é uma função convexa quadrática, as derivadas são bem definidas e suaves e são poucas variáveis.

Desta forma, a escolha de um método sem a necessidade de Hessiana explícita como o BFGS ou o método de Newton são adequados para a otimização do problema.

```
from scipy.optimize import minimize
```

```
# Ponto inicial arbitrário
ponto_inicial = [50, 50]

# Otimização
res = minimize(funcaoobjetivo, ponto_inicial, method='BFGS')

# Resultados
x1_opt, x2_opt = res.x
f_min = res.fun
```

Temperaturas ótimas:

- x1 = 64.3633,
- x2 = 50.7202
- Valor mínimo da função: f(x1, x2) = -2547.7231

O valor mínimo da função representa a **energia potencial associada à distribuição de temperatura** no regime permanente. Esse resultado sugere que as temperaturas x1x_1×1 e x2x_2×2 devem se estabilizar nesses valores para que o sistema atinja sua configuração de **mínima energia térmica**, conforme descrito pelo modelo.

A solução está de acordo com a natureza parabólica da função, conforme verificado pelas curvas de nível plotadas anteriormente. O algoritmo convergiu rapidamente para a solução ótima, o que reforça a adequação da escolha do método BFGS neste contexto.