Universidade Federal de Minas Gerais
Escola de Engenharia
Departamento de Engenharia Elétrica
ELE077 Otimização Não-Linear

Trabalho Computacional II
Otimização Restrita

Daniel Augusto Castro de Paula - 2022060550

Introdução

Em muitos problemas práticos de engenharia e ciência da computação, frequentemente nos deparamos com problemas de otimização que envolvem restrições. Essas restrições podem ser igualdades ou desigualdades que limitam os valores que as variáveis de decisão podem assumir. Esses problemas restritos muitas vezes apresentam desafios para os métodos de otimização tradicionais. Para contornar essas dificuldades, uma abordagem comum é transformar um problema de otimização restrita em um problema irrestrito, onde métodos de otimização irrestrita podem ser aplicados efetivamente. Este trabalho foca em três métodos principais utilizados para essa transformação: o Método das Penalidades Exteriores, o Método Lagrangeano Aumentado e o Método das Penalidades Interiores, cada um com suas peculiaridades e aplicações ideais.

Na otimização restrita, a transformação de problemas restritos em irrestritos é realizada usando métodos como as Penalidades Exteriores, Penalidades Interiores e o Método Lagrangeano Aumentado. O Método das Penalidades Exteriores aborda a otimização aproximando soluções a partir de fora da região factível, permitindo partir de qualquer ponto inicial e utilizando um parâmetro de penalidade que aumenta até o infinito para garantir conformidade com as restrições. Em contraste, o Método de Barreiras, ou Penalidades Interiores, é aplicável apenas a problemas com restrições de desigualdade e trabalha aproximando soluções por dentro da região factível, exigindo que o ponto inicial esteja dentro desta região e que o parâmetro de penalidade diminua progressivamente até zero. Por fim, o Método Lagrangeano Aumentado oferece uma abordagem equilibrada onde a constante de penalidade não precisa escalar agressivamente ao infinito; em vez disso, ela aumenta de forma mais suave, facilitando a convergência e permitindo uma adaptação mais flexível às restrições do problema.

Questões

Questão 1) a) Para solucionar o problema do desenvolvimento de embalagens biodegradáveis iremos reescrever a função objetivo para aplicando o método das penalidades exteriores, o transformando em problema irrestrito

$$\begin{aligned}
\min \quad & f(a,b,c) = 1.5(2ab + 2ac + 2bc) && \text{(1a)}\\
\text{sujeito a :} \quad & abc = 0.032 && \text{(1b)}\\
& 2(a+b) \leq 1.5 && \text{(1c)}\\
& b \leq 3a && \text{(1d)}\\
& c \leq \frac{2}{3}b && \text{(1e)}\\
& 0 \leq a \leq 0.5 && \text{(1f)}\\
& 0 \leq b \leq 0.5 && \text{(1g)}\\
& c \geq 0 && \text{(1h)}
\end{aligned}$$

Para formular a função penalidade devemos somar f(x) que seria (1a), com as demais penalidades de cada igualdade e desigualdade das restrições, acompanhadas de uma constante de penalidade u. Para igualdade adicionamos $u*(h(x))^2$, e para desigualdade $u*\max(0,g(x))^2$.

```
func_penalidade = 1.5 * (2ab + 2ac + 2bc)  → pois temos que adicionar f(x)   (1a)
func_penalidade += u * (a*b*c - 0.032)**2                                    (1b)
func_penalidade += u * max(0, 2*(a + b) - 1.5)**2                            (1c)
func_penalidade += u * max(0, b - 3*a)**2                                    (1d)
func_penalidade += u * max(0, c - (2/3)*b)**2                                (1e)
func_penalidade += u * max(0, a - 0.5)**2                                    (1f)
func_penalidade += u * max(0, b - 0.5)**2                                    (1g)
func_penalidade += u * max(0, -a)**2                                         (1f)
func_penalidade += u * max(0, -b)**2                                         (1g)
func_penalidade += u * max(0, -c)**2                                         (1h)
```

b) Para esse problema poderíamos considerar escolher o BFGS como algoritmo para otimização após converter o problema restrito para irrestrito pois é um problema não quadrático com poucas variáveis. Porém podem surgir complicações devido à natureza da função de penalidade. Especificamente, as penalidades podem introduzir descontinuidades ou não-suavidades na função objetivo, especialmente quando envolvem termos como max(0, x), que são não-diferenciáveis em certos pontos. Essas características complicam o cálculo dos gradientes e das aproximações da Hessiana necessárias para o BFGS, podendo levar a convergências para mínimos locais subótimos ou falhas em encontrar o mínimo global. Por outro lado, métodos sem derivadas como o Nelder-Mead não dependem de gradientes e, portanto, são mais robustos frente a funções objetivo com essas características. Eles exploram o espaço de busca de maneira mais flexível, adaptando-se à função sem a necessidade de derivadas precisas, o que os torna particularmente úteis em cenários complexos com múltiplas restrições e penalidades significativas.

c) No arquivo questao1.ipynb temos 2 implementações, a primeira com o algoritmo Nelder-Mead. Nele foi utilizado critério de parada de precisão quando a mudança na solução for menor igual a 10^{-13}. A penalidade inicial foi de 1, e o fator de crescimento da penalidade foi de 1.5. Com o ponto inicial [1,1,1] obtivemos o seguinte resultado

Resultado Final
Dimensões ótimas (a, b, c): [0.25161768 0.46662247 0.27254916]
Custo mínimo: 0.9395
Número de iterações: 51
Valor final da penalidade (u): 637621500.2140496

Nesse mesmo arquivo implementamos o método trust-constr que é um método desenvolvido na biblioteca do scipy onde não precisamos passar as penalidades apenas passamos as equações das restrições. Nele com ponto inicial [0.1, 0.2, 0.3] foi obtido o seguinte resultado

Resultado Final
Dimensões ótimas (a, b, c): [0.31318896 0.39148721 0.26099126]
Custo mínimo: 0.9196
Número de iterações: 423

Podemos concluir que o resultado foi satisfatório para ambos já que eles apresentam um valor próximo.

Questão 2) a) Para esse problema iremos determinar o peso de cada barra que minimiza o peso da treliça respeitando uma série de restrições que estão definidas nas seguintes equações

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

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

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

$$-P\frac{x_2}{x_1^2\sqrt{2} + 2x_1x_2} \leq -15 \tag{2d}$$

$$0.1 \leq x_1, x_2 \leq 5 \tag{2e}$$

Iremos utilizar o método do Lagrangeano Aumentado e para isso devemos escrever o problema com a seguinte formulação, no qual g(x) são restrições de desigualdade e h(x) são restrições de igualdade.

$$L_A(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}, u) = f(\mathbf{x}) + \sum_i \mu_i g_i(\mathbf{x}) + \sum_j \lambda_j h_j(\mathbf{x}) + \\ + \frac{u}{2} \sum_i \max\left[0, g_i(\mathbf{x})\right]^2 + \frac{u}{2} \sum_j \left[h_j(\mathbf{x})\right]^2$$

Em python essa conversão ficou no seguinte formato.

```python
def lagrangeano_aumentado(x, mu, u):
    x1, x2 = x
    # f(x) = (2*sqrt(2))*x1 + x2
    f = (2*sqrt2)*x1 + x2

    g = np.zeros(7)
    # g1: P*(x2 + x1*sqrt2)/(x1^2*sqrt2 + 2*x1*x2) - 20 <= 0
    g[0] = P*(x2 + x1*sqrt2)/(x1**2*sqrt2 + 2*x1*x2) - 20
    # g2: P/(x1 + x2*sqrt2) - 20 <= 0
    g[1] = P/(x1 + x2*sqrt2) - 20
    # g3: 15 - P*x2/(x1^2*sqrt2 + 2*x1*x2) <= 0  =>  P*x2/(...) - 15 >= 0
    g[2] = 15 - (P*x2/(x1**2*sqrt2 + 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)
```

b) Para resolver o problema após converter para um problema irrestrito, escolhemos o método BFGS devido à sua eficácia em lidar com funções contínuas e diferenciáveis e por sua capacidade de utilizar informações de segunda ordem aproximadas. O BFGS adapta-se bem à presença de múltiplas restrições transformadas em penalidades, ajustando sua direção de busca de forma eficiente para explorar a superfície da função objetivo de maneira mais estratégica. Isso é particularmente vantajoso em cenários onde as restrições são integradas como termos de penalidade no Lagrangeano Aumentado, permitindo que o método mantenha a diferenciabilidade necessária para cálculos de gradientes precisos. Além disso, a rápida convergência do BFGS, comparada a métodos de primeira ordem e métodos sem derivadas, nos deu uma solução mais eficiente e robusta, minimizando o custo computacional e aumentando a precisão da solução final.

c) No arquivo questao2.ipynb foi utilizado critério de parada de precisão quando a mudança na solução for menor igual a 10^{-13}. A penalidade inicial foi de 1, o fator de crescimento da penalidade foi de 1.5 e o número máximo de iterações foi 100. Com o ponto inicial [1,1]

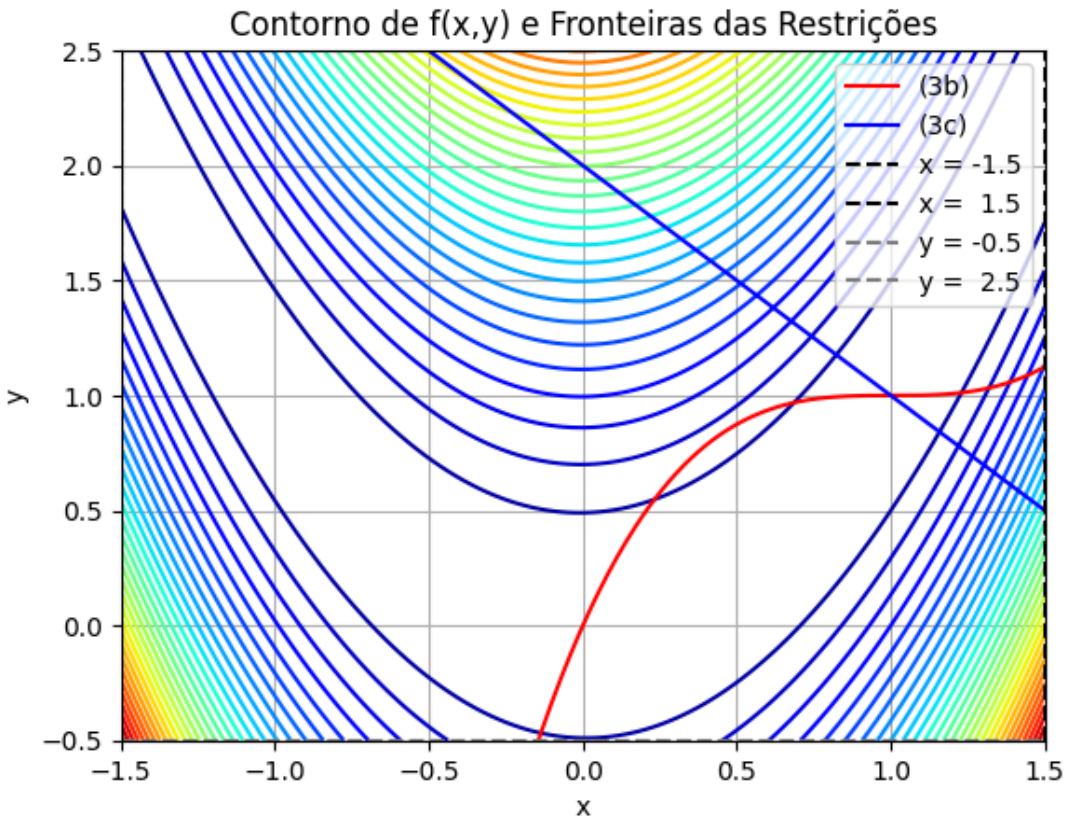
obtivemos o seguinte resultado

Resultado Final
Iterações realizadas: 9
x* = [0.57142857 2.42436592]
f(x*) = 4.040609988393852
g(x*): [2.04657400e-07 -1.49999997e+01 1.26080455e-07 -4.71428570e-01
-4.42857143e+00 -2.32436592e+00 -2.57563408e+00]
mu: [0.60032516 0. 0.531051 0. 0. 0. 0.]
Penalidade final u: 25.62890625

Esse resultado foi satisfatório pois chegamos exatamente no ponto ótimo.

Questão 3) a) Para o problema canônico de otimização restrita plotamos o gráfico da função objetivo e curvas de restrições e obtivemos o seguinte resultado

$$\min \quad f(x,y) = (1-x)^2 + 100(y-x^2)^2 \tag{3a}$$
$$\text{sujeito a}: (x-1)^3 - y + 1 \leq 0 \tag{3b}$$
$$x + y - 2 \leq 0 \tag{3c}$$
$$-1.5 \leq x \leq 1.5 \tag{3d}$$
$$-0.5 \leq y \leq 2.5 \tag{3e}$$



b) O método da penalidade interior (método das barreiras) só pode ser aplicado para problemas onde as restrições são de desigualdade. A função barreira deve apresentar a soma de f(x) com as restrições penalizadas pelo formato $\frac{-u}{g(x)}$ onde u é a constante de penalidade, ficando com o seguinte formato

$$b(\mathbf{x}, u) = -u \left\{ \sum_{i=1}^{l} \frac{1}{g_i(\mathbf{x})} \right\}$$

A conversão para código python ficou assim:

```python
# Definição da função objetivo
def f_obj(xy):
    x, y = xy
    return (1 - x)**2 + 100*(y - x**2)**2

# Definição das restrições g_i(x,y) <= 0
def g_funcs(xy):
    x, y = xy
    return np.array([
        (x - 1)**3 - y + 1,   # g1
        x + y - 2,            # g2
        x - 1.5,              # g3 (x <= 1.5)
        -x - 1.5,             # g4 (x >= -1.5)
        y - 2.5,              # g5 (y <= 2.5)
        -y - 0.5              # g6 (y >= -0.5)
    ])

# Função de penalidade interior (evita dividir por zero)
def interior_penalty(xy, u):
    val_f = f_obj(xy)
    g_vals = g_funcs(xy)

    penalty = 0.0
    eps = 1e-15  # tolerância para evitar divisão por zero exata

    for gi in g_vals:
        # Se g_i >= 0, significa que o ponto não está no interior
        if gi >= 0:
            return np.inf

        if gi > -eps:
            gi = -eps

        penalty += -u / gi

    return val_f + penalty
```

c) No presente estudo, escolheu-se o método Nelder-Mead para resolver o problema de otimização restrita devido à sua capacidade de lidar eficazmente com funções que apresentam complexidades como vales estreitos e curvaturas acentuadas, típicas no problema de Rosenbrock modificado. Este método de otimização sem derivadas é particularmente vantajoso em situações onde o cálculo de gradientes é impraticável ou sujeito a erros devido à presença de descontinuidades ou condições não suaves introduzidas pelas restrições. A abordagem adaptativa do Nelder-Mead, que ajusta o procedimento de busca às características da função objetivo e das restrições sem a

necessidade de derivadas, mostrou-se adequada para explorar o espaço de soluções complexas e restritas.

Outros métodos como o BFGS, apesar de ser robusto para muitas aplicações de otimização irrestrita, enfrentou dificuldades neste contexto específico. A dependência de boas estimativas iniciais e a sensibilidade às complexidades introduzidas pelas restrições do problema podem resultar em convergências para mínimos locais não ótimos. Além disso, o BFGS requer gradientes precisos, o que pode ser desafiador em cenários com restrições não lineares e penalidades severas.

d) No arquivo questao3.ipynb temos 2 implementações, a primeira com o algoritmo Nelder-Mead. Nele foi utilizado critério de parada de precisão quando a mudança na solução for menor igual a 10^{-13}. A penalidade inicial foi de 1, o fator de crescimento da penalidade foi de 1 e o máximo de iterações foi 100. Com o ponto inicial [0.5,1] obtivemos o seguinte resultado

Resultado Final
Número de iterações: 2
x* = [0.83423424 1.06942821]
f(x*) = 13.976317241407916
g(x*): [-0.07398317 -0.09633755 -0.66576576 -2.33423424 -1.43057179 -1.56942821]
Penalidade final u: 1

Na segunda implementação foi utilizado o método SLSQP nativo do scipy para esses tipos de problemas restritos, com ele o resultado obtido foi:

Resultado Final
x* = [0.99999967 1.00000033]
f(x*) = 9.914816552283568e-11
Sucesso: True
Motivo do término: Optimization terminated successfully
Número de iterações: 4

Mais uma vez, os dois métodos apresentaram resultados satisfatoriamente próximos.

Conclusões

Após a análise dos 3 problemas descritos previamente nesse trabalho, destaca-se a importância da seleção adequada do método de otimização ao tratar problemas de otimização restrita convertidos em problemas irrestritos. Cada um dos métodos de otimização restrita explorados, como penalidades exteriores, penalidades interiores e o método de Lagrangeano aumentado possui suas próprias vantagens e desvantagens, influenciando diretamente a eficácia e a eficiência da solução final. A escolha do algoritmo para resolver o problema irrestrito subsequente é importante, pois diferentes algoritmos podem lidar de maneira variável com as complexidades introduzidas pelas penalidades e restrições.

Além disso, os parâmetros utilizados na formulação das penalidades e na configuração dos algoritmos têm um impacto significativo na qualidade e na convergência das soluções. Ajustes inadequados podem levar a convergências indevidas para mínimos locais, soluções subótimas, ou falhas na observância das restrições originais do problema. Portanto, é importante ter um bom entendimento da estrutura do problema e até mesmo dos algoritmos irrestritos para escolher os melhores parâmetros e algoritmos.

Por fim, é relevante mencionar que os métodos nativos disponíveis em bibliotecas como o SciPy muitas vezes oferecem implementações mais eficientes e robustas do que abordagens manualmente implementadas. Estes métodos nativos são otimizados para aproveitar algoritmos avançados, proporcionando uma melhor performance e maior facilidade de uso. A utilização desses métodos integrados é uma boa opção, especialmente quando se busca eficiência e confiabilidade na solução de problemas complexos de otimização.