Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Alunos: Deandreson Alves - 10145310,
Hiago Vinícius Américo -11218469

SME0212 Otimização Não Linear

Professor: Prof. Dr. Elias Salomão Helou Neto

Método de Penalidades

Resumo

Este trabalho tem como objetivo apresentar o Método de Penalidades, um dos métodos de otimização com restrições que é frequentemente escolhido pela sua relativa simplicidade. Disso, abordaremos a sua definição e implementaremos para resolução de alguns problemas com restrições no qual buscamos penalizar qualquer violação das restrições do problema original.

1 Dissertação

O **Método de Penalidades** é uma técnica fundamental na otimização não linear, particularmente importante quando se trata de problemas no qual as variáveis de decisão estão sujeitas a restrições que não são facilmente incorporáveis em métodos de otimização diretos. Este método, é um dos vários métodos de otimização com restrições e é frequentemente escolhido pela sua relativa simplicidade e versatilidade.

Em muitos problemas reais, as restrições desempenham um papel crucial, limitando o conjunto de soluções factíveis. No entanto, lidar diretamente com essas restrições pode ser matematicamente complexo e computacionalmente oneroso. Aqui, o Método de Penalidades oferece uma abordagem pragmática, transformando um problema de otimização com restrições em um mais fácil de resolver, um problema de otimização sem restrições. Assim, suponha que queremos minimizar a função objetivo e suas restrições definidas por:

$$min \quad f(x) \tag{1}$$
$$s.a. \quad g_i \leq 0, i = 1, ..., m \tag{2}$$
$$h_j = 0, j = 1, ..., n \tag{3}$$

O conceito central do **Método de Penalidades** baseia-se na ideia de **"penalizar"** qualquer violação das restrições do problema original. Isso é feito adicionando termos de penalidade à função objetivo. Esses termos são projetados de tal forma que seu valor aumenta quando uma restrição é violada. Assim, a função de penalização da restrição e desigualdade ficam:

No caso de um problema com uma restrição de igualdade:

$$\begin{cases} min \quad f(x) \\ s.a. \quad h_j = 0, j = 1, ..., n \end{cases} \quad \text{sendo} \quad min \quad f(x) + u[h_j(x)]^2$$

No caso de um problema com uma restrição de desigualdade:

$$\begin{cases} min \quad f(x) \\ s.a. \quad g_i \leq 0, i = 1, ..., m \end{cases} \quad \text{sendo} \quad min \quad f(x) + u\mathbf{max}[0, g_i(x)]^2$$

Sendo a função geral de penalização:

$$p(x, u) = u\{ \ \textstyle\sum_{j=1}^{n}[h_j(x)]^2 + \sum_{i=1}^{m} \mathbf{max}[0, g_i(x)]^2 \ \}$$

E disso,

$$min \quad f(x) + p(x, u) \tag{4}$$

Na essência do **Método de Penalidades** para otimização não linear encontra-se o princípio de incorporar uma função de penalidade à função objetivo original conforme equação (4), uma abordagem que permite a transformação de um problema com restrições em um problema sem restrições.

Além disso, este método se destaca pela sua capacidade de penalizar as violações das restrições, onde a função de penalidade aumenta o valor da função objetivo sempre que uma solução proposta não adere às restrições impostas. O ajuste das penalidades é um elemento crucial neste processo; começando com penalidades mais leves e aumentando-as progressivamente, busca-se um equilíbrio que force o respeito às restrições sem comprometer a busca pela solução ótima da função objetivo. Comumente, utiliza-se a penalidade quadrática, no qual o termo de penalidade é proporcional ao quadrado da violação da restrição, permitindo uma graduação na severidade da penalidade de acordo com o grau da violação. Este método, portanto, oferece uma estratégia pragmática e eficaz para lidar com problemas de otimização que apresentam restrições complexas, facilitando a aplicação de métodos de otimização tradicionais em um contexto que, de outra forma, seria desafiador.

E pela teoria que prevê que a solução converge para a solução ótima quando u=$u_k \to \infty$. Além disso, a variação da penalidade u_k deve ser gradual. E o método pode apresentar mal-condicionamento numérico para u_k extremo.

Em comparação com outros métodos citamos os **Métodos de Lagrangiano Aumentado e o de Pontos Interiores (Barreira)**.

O Método de Pontos Interiores (Barreira) busca garantir que os pontos gerados sejam sempre viáveis. E quando há tentativa de sair da região factível, ocorre uma penalização por meio de uma função de penalidade que ao aproximar cada vez mais da barreira, faz a penalização tender ao infinito. Sendo a solução ótima aproximada internamente por uma sequência de soluções do problema irrestrito transformado.

Já o Método de Lagrangiano Aumentado é uma extensão dos métodos de penalidades. Sua implementação, possibilita que não tenha o problema da função de penalidade mal-condicionada com o aumento de u, sobressaindo-se nesse quesito em relação ao Método de Penalidades, pois não há a necessidade de fazer u=$u_k \to \infty$. Além disso, a solução inicial não precisa ser factível e os multiplicadores de Lagrange diferentes de zero identificam as restrições ativas no ponto solução.

2 Implementação

Criar um algoritmo em Python que implementa o **Método de Penalidades** para otimização não linear envolve várias etapas. Vamos começar com um exemplo simples: um problema de otimização com uma única restrição. Suponha que queremos minimizar a função objetivo $f(x)$ sujeita a uma restrição $g(x) \leq 0$. Usaremos uma penalidade quadrática para lidar com a restrição.

Vamos lidar com o algoritmo passo a passo para poder nos organizar e entender melhor. Nosso algoritmo segue os seguintes passos:

1. **Definir a Função Objetivo e a Restrição**
   Escrever funções para calcular $f(x)$ e $g(x)$
2. **Definir a Função Penalizada**
   Combina a função objetivo com a penalidade pela violação da restrição
3. **Escolher um Método de Otimização**
   Usar um otimizador disponível em uma biblioteca, como *'scipy.optimize'*
4. **Ajustar o peso da Penalidade**
   Iniciar com um peso de penalidade baixo e aumentá-lo iterativamente
5. **Iterar até a convergência**
   Resolver o problema penalizado em cada iteração e verificar a convergência.

Funções do código

Função Objetivo '$f\_complex(x)$'

Definimos aqui a função objetivo do problema de otimização. A implementação de $f\_complex(x)$ é uma combinação de funções trigonométricas e quadráticas. Especificamente, $f(x, y) = \sin(x) * \cos(y) + (x-1)^2 + (y-2)^2$. Aqui x e y são as variáveis do problema, representadas por x[0] e x[1] no presente algoritmo. Esta função objetivo é mais complexa do que uma simples função quadrática, tornando o problema de otimização mais desafiador e interessante para demonstrar o método estudado.

Função de Restrição '$g\_complex(x)$'

Está função define a restrição do problema de otimização. A implementação de '$g\_complex(x)$ é definida como $x^2 + y^2 - 4$, representados por *x[0]**2 + x[1]**2 - 4* no algoritmo. Esta restrição impõe que as soluções devem estar dentro de uma círculo de raio 2 centrado na origem, adicionando uma condição geométrica ao problema restrigindo o conjunto de soluções factíveis.

Função Penalizada '$penalized\_function\_complex(x, penalty\_weight)$'

Combina a função objetivo com a penalidade pela violação da restrição. A implementação calcular o valor objetivo de '$f\_complex(x)$' e adiciona um termo de penalidade baseado na violação de restrição '$g\_complex(x)$'. O termo de penalidade é '$max(0,'$g_complex(x)$'^2$', que é ponderado pelo '$penalty\_weight$. A função penalizada permite transformar um problema de otimização com restrições em um problema sem restrições, no qual a penalidade desencoraja a violação das restrições.

O algoritmo

Aqui nós implementamos o Método de Penalidades para encontrar uma solução ótima do problema de otimização. Iniciamos com um palpite inicial ('$initial\_guess$' e um peso de penalidade ('$initial\_penalty\_weight$'. Em cada iteração, o problema penalizado (definido por '$penalized\_function\_complex$') é resolvido usando o método 'minimize' do SciPy. O peso da penalidade é aumentado após cada iteração pelo fator '$penalty\_multiplier$' até que a solução satisfaça a restrição ou um número máximo de iterações seja alcançado. O objetivo é iterativamente aproximar-se da solução do problema original, ajustando as penalidades para forçar a solução a respeitar as restrições.

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# Nova função objetivo
def f_complex(x):
    return np.sin(x[0]) * np.cos(x[1]) + (x[0] - 1)**2 + (x[1] - 2)**2

# Nova restrição
def g_complex(x):
    return x[0]**2 + x[1]**2 - 4  # x^2 + y^2 <= 4 (círculo de raio 2)

# Função penalizada ajustada
def penalized_function_complex(x, penalty_weight):
    penalty = max(0, g_complex(x)) ** 2  # Penalidade quadrática para g_complex(x) > 0
    return f_complex(x) + penalty_weight * penalty

# Algoritmo do método de penalidades ajustado
def penalty_method_complex(initial_guess, initial_penalty_weight=1, max_iterations=100,
penalty_multiplier=10):
    x_best = initial_guess
    penalty_weight = initial_penalty_weight
    history = []  # Guardar histórico das soluções

    for i in range(max_iterations):
        result = minimize(lambda x: penalized_function_complex(x, penalty_weight), x_best)
        x_best = result.x
        history.append(x_best)

        # Checar se a penalidade ainda está ativa; se não, parar
        if max(0, g_complex(x_best)) == 0 and penalty_weight > 1e5:
            break

        penalty_weight *= penalty_multiplier  # Aumentar o peso da penalidade

    return x_best, history

# Executando o algoritmo ajustado
```

```
initial_guess_complex = [0, 0]
optimal_solution_complex, history_complex = penalty_method_complex(initial_guess_complex)

# Preparando os dados para plotar a taxa de convergência
objective_values_complex = [f_complex(point) for point in history_complex]
convergence_rate_complex = np.abs(np.diff(objective_values_complex))

# Plotando a taxa de convergência

plt.plot(convergence_rate_complex, marker='o')
plt.title("Taxa de Convergência no Método de Penalidades")
plt.xlabel("Iteração")
plt.ylabel("Mudança Absoluta no Valor da Função Objetivo")
plt.yscale("log")  # Usar escala logarítmica para melhor visualização
plt.grid(True)
plt.show()
```

Plotagem do gráfico

Fizemos uma plotagem para visualizar a eficácia do algoritmo ao longo das iterações, conforme Figura 1. Após executar o algoritmo *penalty_method_complex*, calcula-se o valor da função objetivo *f_complex* para cada ponto na história das soluções. A taxa de convergência é então determinada pela diferença absoluta entre os valores consecutivos da função objetivo. O gráfico da taxa de convergência fornece uma visualização intuitiva de como o valor da função objetivo está se aproximando do mínimo ao longo das iterações, evidenciando a eficácia do Método de Penalidades.

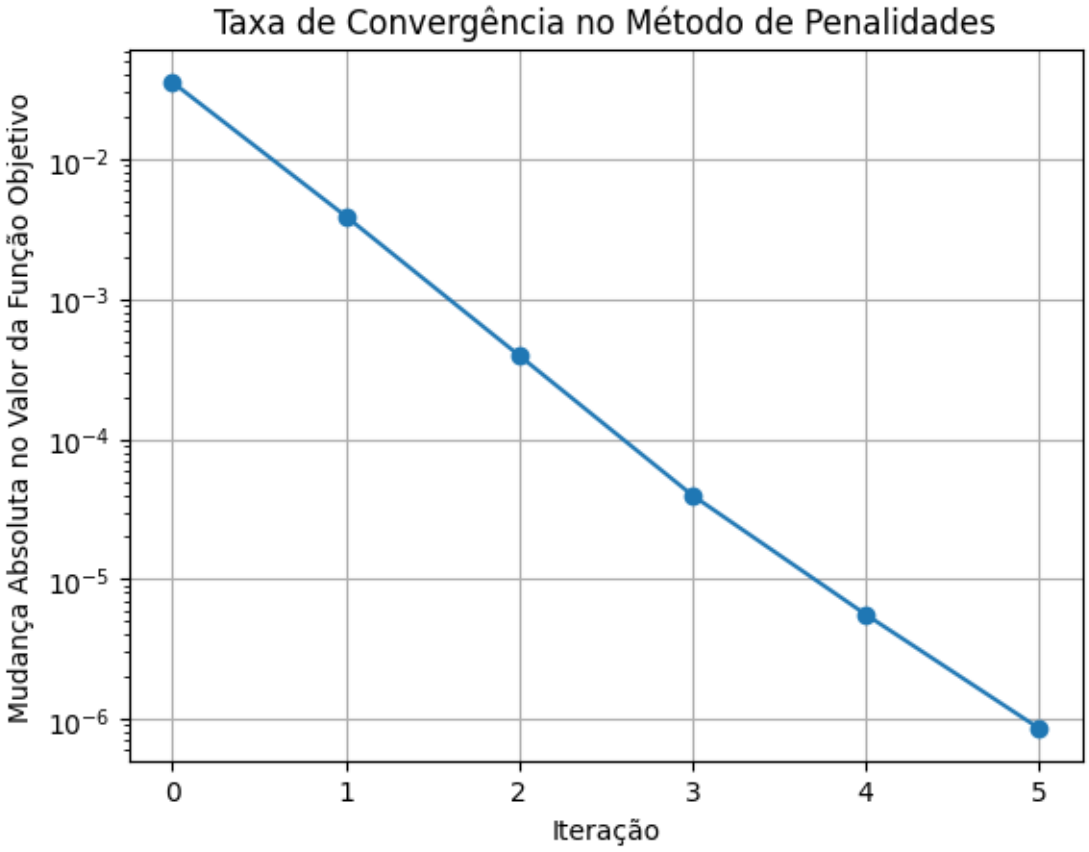

Figura 1: Taxa de Convergência no Método de Penalidades

3 Execução

Vamos gerar problemas de otimização com um número crescente de variáveis. Por exemplo, começando com problemas de 2 dimensões e aumentando gradualmente. Usaremos uma função de cronometragem,

como *time.time*() do Python, para medir quanto tempo o algoritmo leva para resolver cada problema. A precisão pode ser avaliada comparando a solução encontrada com uma solução conhecida ou esperada, ou avaliando a proximidade da solução às restrições do problema.

```
import numpy as np
import time
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# Exemplo de função objetivo (adaptável para diferentes dimensões)
def f_sintetica(x):
    return np.sum((x - np.arange(len(x)))**2)

# Exemplo de restrição (a soma dos elementos deve ser menor que um valor)
def g_sintetica(x, limit):
    return np.sum(x) - limit

# Função para o Método de Penalidades (adaptável para diferentes dimensões)
def penalized_function_sintetica(x, penalty_weight, limit):
    penalty = max(0, g_sintetica(x, limit)) ** 2
    return f_sintetica(x) + penalty_weight * penalty

# Algoritmo de otimização com penalidades
def penalty_method_sintetica(dim, limit, penalty_weight=100, max_iterations=100):
    initial_guess = np.zeros(dim)
    start_time = time.time()
    result = minimize(lambda x: penalized_function_sintetica(x, penalty_weight, limit), initial_guess
    end_time = time.time()
    execution_time = end_time - start_time
    return result, execution_time

# Análise para diferentes tamanhos de problemas
sizes = [2, 5, 10, 20, 50, 100]  # Exemplo de tamanhos de problema
limit = 10  # Exemplo de limite para a restrição
execution_times = []

for size in sizes:
    result, exec_time = penalty_method_sintetica(size, limit)
    execution_times.append(exec_time)

# Plotando o gráfico dos resultados
plt.plot(sizes, execution_times, marker='o')
plt.title("Tempo de Execução do Método de Penalidades vs Tamanho do Problema")
plt.xlabel("Tamanho do Problema (Número de Variáveis)")
plt.ylabel("Tempo de Execução (segundos)")
plt.grid(True)
plt.show()
```

O código acima é uma simplificação para fins de demonstração. Em problemas reais, as funções objetivo e de restrição podem ser mais complexas. Ao analisar os resultados, é importante considerar não apenas o tempo de execução, mas também a qualidade das soluções encontradas. Em alguns casos, pode ser necessário equilibrar tempo e precisão. Este experimento é limitado pela escolha das funções e pelos parâmetros do método. Resultados diferentes podem ser obtidos com diferentes configurações.

Conforme Figura 2, temos o gráfico que mostra o tempo de execução do Método de Penalidades em relação ao tamanho do problema, medido pelo número de variáveis. Conforme o tamanho do problema aumenta, observa-se uma tendência de aumento no tempo de execução. Este comportamento é esperado, pois problemas com mais variáveis geralmente são mais complexos e requerem mais cálculos durante a otimização.

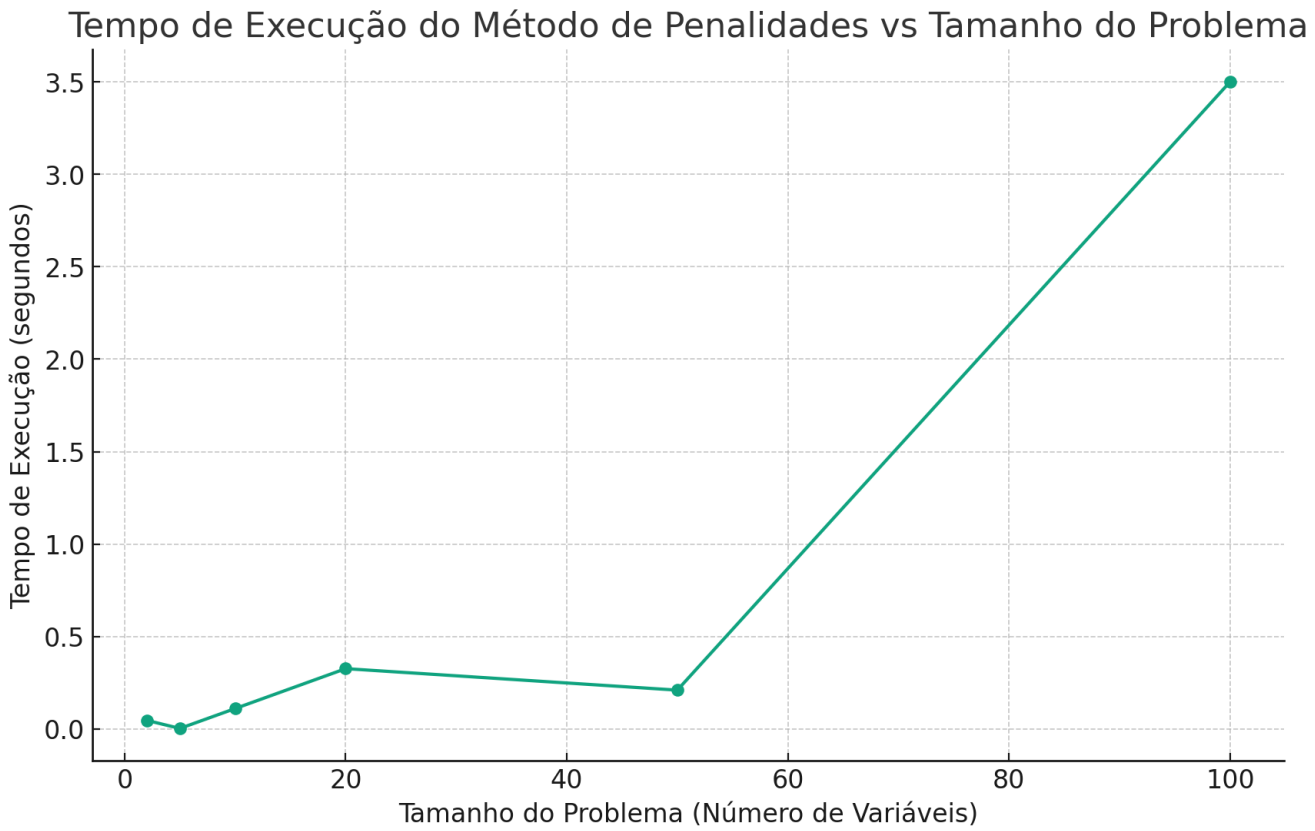

Figura 2: Desempenho Numérico

Esse gráfico é útil para entender como o desempenho do Método de Penalidades é afetado pelo tamanho do problema, fornecendo insights importantes sobre a escalabilidade do método em diferentes cenários.

4 Referências

Carrano, E. G.; Guimarães, F. G.; Batista, L. S. (2023). **Otimização Multiobjetivo e Restrita**. Slides de aula. Disponível em: http://www.cpdee.ufmg.br/ lusoba/disciplinas/eee910/slides/lusoba/03-otim-restrita.pdf. Acesso em: 27 de dezembro de 2023.

DE OLIVEIRA ROCHA, Ramon; FRANCO, Hernando José Rocha. **Uma implementação do método de Penalidades para problemas de otimização restrita**. Revista de Matemática, v. 2, p. 1-14, 2022. Disponível em: https://periodicos.ufop.br/rmat/article/view/5272. Acesso em: 27 de dezembro de 2023.

Friedlander, A. (2012). **Elementos de programaçao nao-linear**. Livro-texto. UNICAMP. Disponível em: http://www.ime.unicamp.br/ andreani/MS629/Textos/livroana.pdf. Acesso em: 27 de dezembro de 2023.