Lista 01

MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica

Pedro Henrique Cardoso Paulo
pedrorjpaulo.phcp@gmail.com

Professor: Ivan Menezes

Departamento de Engenharia Mecânica
PUC-RJ Pontifícia Universidade Católica do Rio de Janeiro
abril de 2023

Lista 01

MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica

Pedro Henrique Cardoso Paulo

abril de 2023

1 Introdução

1.1 Objetivos

Esse é o entregável da Lista 01 da disciplina MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica. Esse trabalho tem como objetivos:

1. Implementar os principais métodos para cálculo de ponto de mínimo em funções de uma variável
2. Aplicar esses métodos em funções 2D ao longo de uma dada direção
3. Exercitar a linguagem de programação e as ferramentas de visualização gráfica

1.2 Links úteis

Nesta seção são listados alguns links e referências úteis para se entender o trabalho desempenhado.

1. Apostila de programação matemática da disciplina
2. GitHub usado para essa disciplina
3. Notebook com o código para as figuras desse relatório
4. Pasta com os códigos a serem aproveitados em todas as listas

2 Questão 01

2.1 Enunciado

Implementar, usando o MATLAB ou Python, os seguintes métodos para cálculo do ponto de mínimo de funções de uma única variável:

- Passo constante (com $\Delta\alpha = 0.01$)
- Bisseção (usando o Passo Constante para obtenção do intervalo de busca)
- Seção Áurea (usando o Passo Constante para obtenção do intervalo de busca)

2.2 Solução

De modo a garantir o reaproveitamento do código para tarefas futuras, os três métodos foram implementados como classes Python, garantindo a possibilidade de herança entre classes. Um esquemático das 3 classes implementadas é mostrado na Figura 1, onde nota-se que foi convencionado ter como classe-pai das demais a classe do passo constante. Esse arranjo foi considerado adequado uma vez que o passo constante é o método básico do qual todos os demais métodos partem para refinar a estimativa inicial de passo.

O código completo das classes implementadas pode ser visto no arquivo adicionado ao GitHub `steps.py`, onde a implementação completa das 3 classes está armazenada. Abaixo, para fins de exemplo, é mostrada a implementação da classe de passo constante, da qual todas as demais herdam.

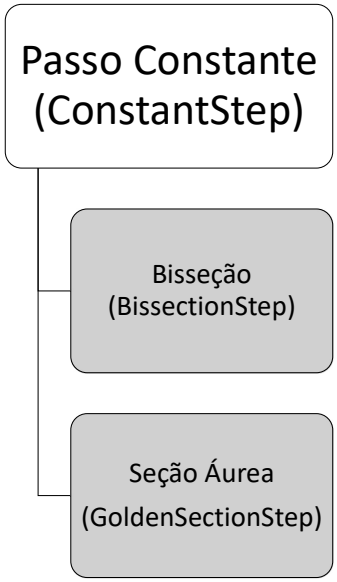


Figura 1: Estrutura de classes implementada e heranças

```
class ConstantStep:

    def __init__(self, da):

        self.da = da
        self.aL = None
        self.aU = None
        self.fL = None
        self.fU = None

        self.reset_step()

    def reset_step(self):

        self.aL = 0
        self.aU = self.da
        self.fL =  0.0
        self.fU = -1.0

    def calculate_bounds(self, p_initial, direction, function):

        self.fL = function(*(p_initial + self.aL*direction))
        self.fU = function(*(p_initial + self.aU*direction))

    def __call__(self, p_initial, direction, function):

        self.reset_step()
        self.calculate_bounds(p_initial, direction, function)
        while self.fL > self.fU:
            self.aL = self.aU
            self.aU += self.da
            self.calculate_bounds(p_initial, direction, function)
        pend = p_initial + self.aL*direction
        return self.aL, pend
```

A usabilidade das classes de passo implementadas é feita de forma similar à de uma função graças à implementação de um método `__call__` em seu corpo. Dessa forma, uma vez declarado o step com seus parâmetros, uma quantidade infinita de passos podem ser dados fornecendo-se como entrada a função, ponto inicial e direção. Um exemplo do uso dessa classe pode ser visto abaixo.

```
import numpy as np
import steps
```

```
# Funcao a ser otimizada
def f(x1, x2):
    return np.sin(x1 + x2) + (x1 - x2)**2 - 1.5*x1 + 2.5*x2

# Ponto inicial e direcao
p_inicial = np.array([-2, 3])
d = np.array([1.453, -4.547])

# Instanciando o passo
stp = steps.ConstantStep(da = 0.01)

# Dando o passo
ak, p_final = stp(p_inicial, d, f)
```

3 Questão 02

3.1 Enunciado

Utilizando os métodos implementados na questão anterior, testar a sua implementação encontrando o ponto de mínimo das seguintes funções:

(a) Primeiro Exemplo:
$f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2$
Ponto inicial: $\mathbf{x^0} = [1, 2]^T$, Direção: $\mathbf{d} = [-1, -2]^T$

(b) Função de McCormick:
$f(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2$
Ponto inicial: $\mathbf{x^0} = [-2, 3]^T$, Direção: $\mathbf{d} = [1.453, -4.547]^T$

(c) Função de Himmelblau:
$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$
Ponto inicial: $\mathbf{x^0} = [0, 5]^T$, Direção: $\mathbf{d} = [3, 1.5]^T$

Para cada função acima, utilize o MATLAB ou Python para desenhar (na mesma figura) as curvas de nível e o segmento de reta conectando o ponto inicial ao ponto de mínimo. Adotar tolerância de 10^{-5} para verificação da convergência numérica.

3.2 Solução

Para executar esse exercício, serão usadas as classes já descritas na Questão 01 (seção 2). A biblioteca `matplotlib` do Python será novamente usada para gerar os gráficos necessários. Para facilitar a vaisualização, será gerado um gráfico para cada método de cálculo do passo, sendo que cada gráfico deverá conter, pelo menos:

- As curvas de nível da função a ser estudada
- Os pontos iniciais e finais do passo
- A direção esperada do passo
- Uma seta ou linha ligando os pontos iniciais e finais
- O tamanho do passo α_k e coordenadas do ponto final (incluindo $z = f(x_1, x_2)$) no título do gráfico

Um exemplo simples do código que foi usado para gerar os gráficos apresentados neste relatório é apresentado abaixo. Em seguida, são apresentadas as seções que mostram os resultados obtidos e alguns comentários a respeito destes.

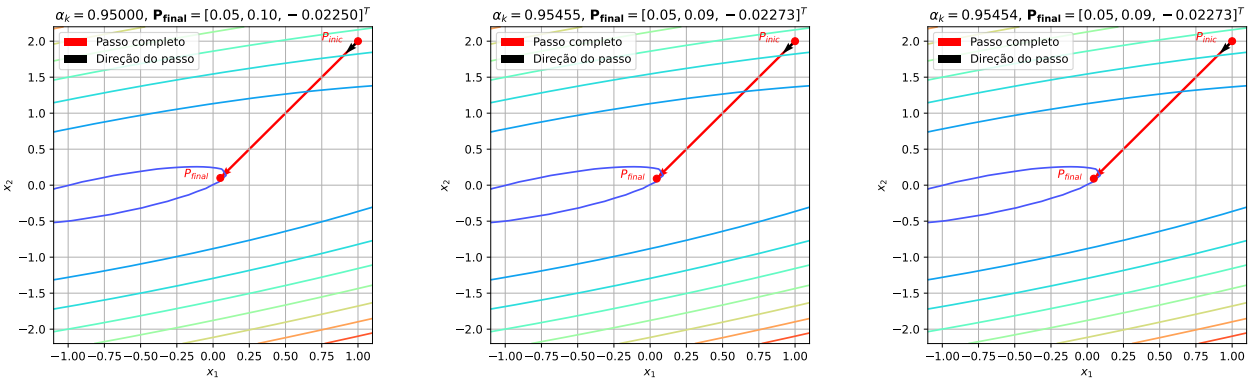
```
step_list   = [steps.ConstantStep(da = 0.01), steps.BissectionStep(da = 0.01, tol = 1e-5),
                                              steps.GoldenSectionStep(da = 0.01, tol = 1e-5)]

item = 'a'
x = np.linspace(-3.3, 0.1, 50)
y = np.linspace(3, 5.1, 50)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

i = 0
for step in step_list:
    i += 1
    fig, ax = plt.subplots(1,1, figsize=(5, 5))
    ax.contour(X, Y, Z, cmap='rainbow')
    ax.plot(*p_inicial, 'ro')
    ax.text(p_inicial[0]-0.25, p_inicial[1]+0.04, '$P_{inic}$', color='red')
    ax.plot(*p_final, 'ro')
    ax.text(p_final[0]-0.25, p_final[1]+0.04, '$P_{final}$', color='red')
    ax.quiver(p_inicial[0], p_inicial[1], p_final[0]-p_inicial[0], p_final[1]-p_inicial[1],
                                                color='red', angles='xy', scale_units='xy',
                                                 scale=1)
    ax.quiver(p_inicial[0], p_inicial[1], d[0], d[1], color='black', angles='xy', label='
                                                Direcao do passo')
    ax.grid()
    ax.legend()
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_title(f'$\\alpha_k = {ak:.5f}$, $\mathbf{{P_{{final}}}} = [{p_final[0]:.2f}, {
                                                p_final[1]:.2f}]^T$')
```

3.2.1 Resultados para a função (a)

Os resultados para a análise da função (a) são apresentados na Figura 2. Analisando os resultados é possível ver que os três méetodos retornaram resultados bem similares, com os étodos da Bisseção e Seção áurea retornando uma estimativa ligeiramente mais refinada para o mínimo, com mínimas diferenças no α_k estimado. É interessante notar que, para este exemplo, o ponto estimado parece ser aproximadamente o ponto em que a direção em que tentamos calcular o mínimo tangencia uma curva de nível, o que convergeria para um ponto em que esperaríamos ter o primeiro mínimo local da função nessa direção.

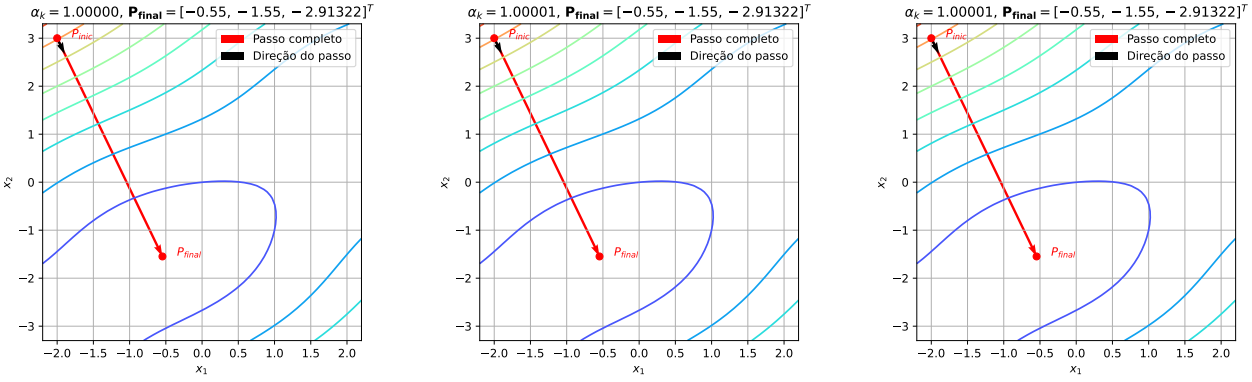


(a) Passo constante **(b)** Bisseção **(c)** Seção áurea

Figura 2: Resultados para a função (a) caso com diferentes métodos

3.2.2 Resultados para a função (b)

Os resultados para a análise da função (b) são apresentados na Figura 3. Novamente, não há diferenças aapreciáveis entre os resultados dos métodos. Isso provavelmente se deve ao fato de o $\Delta\alpha$ especificado para o passo constante (e usado de base para delinear o intervalo dos demais métodos) já ser refinado o suficiente para obtermos uma resposta adequada sem a necessidade de uma posterior Bisseção ou Seção Áurea.



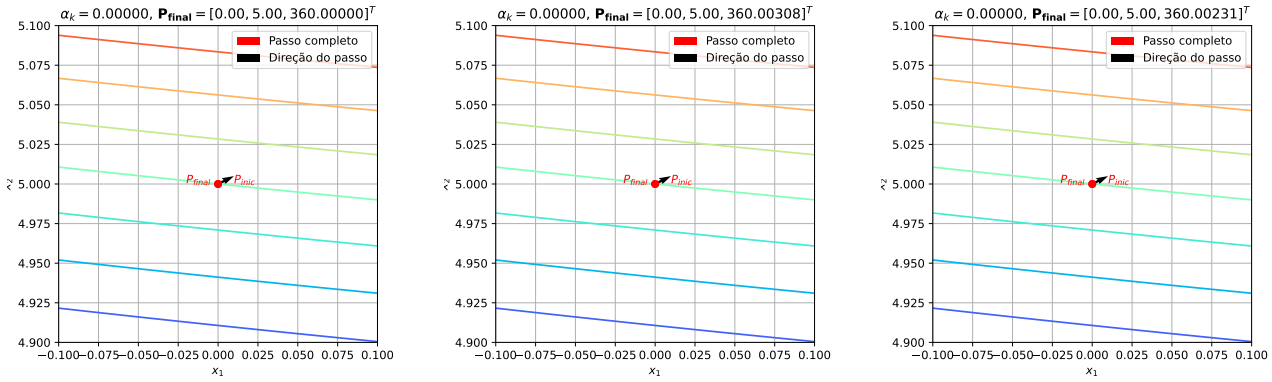
(a) Passo constante **(b)** Bisseção **(c)** Seção áurea

Figura 3: Resultados para a função (b) com diferentes métodos

3.2.3 Resultados para a função (c)

Os resultados para a análise da função (c) são apresentados na Figura 4. Diferentemente dos casos anteriores, aqui vemos que os três métodos convergiram pra pontos iguais ou muito próximos do ponto inicial, ou seja, nosso ponto inicial já é o mínimo esperado. Isso é algo que faz sentido ao notarmos que a direção em que buscamos o nosso mínimo tem sentido apontando para curvas de nível de maior valor, ou seja, a direçõ de busca necessariamente leva a valorres maiores da função na proximidade do ponto inicial.

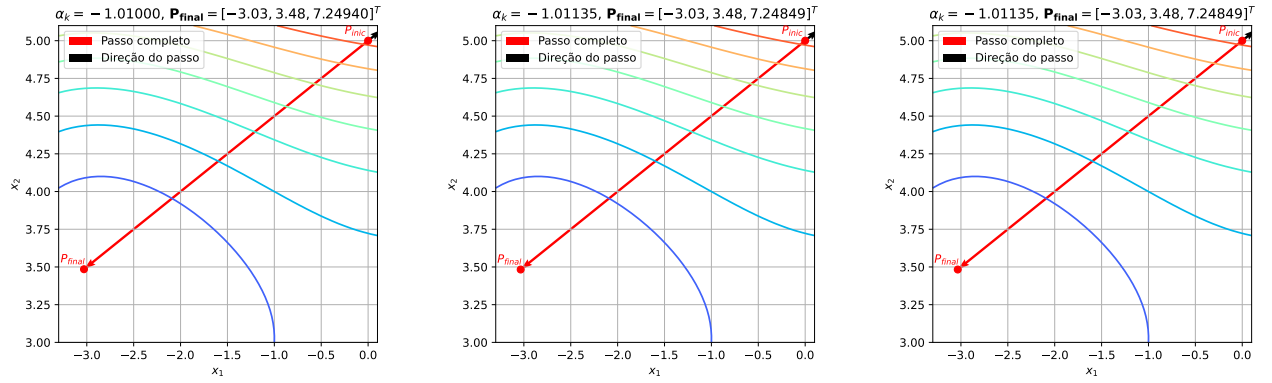
É importante ressaltar que isso não é um problema do método numérico por si, mas um problema dos inputs forencidos para a estimativa do mínimo. De forma a provar esse argumento, a Figura 5 mostra o resultado que seria obtido fazendo-se a bisca na mesma direção mas com vetor de sentido contrário ($\mathbf{d} = [-3, -1.5]^T$). Nota-se



(a) Passo constante **(b)** Bisseção **(c)** Seção áurea

Figura 4: Resultados para a função (c) com diferentes métodos

que nesse caso o método seguiu até um ponto de mínimo inferior ao ponto inicial, evidenciando o argumento de que, além de termos algoritmos capazes de estimar corretamente o α_k, é igualmente importante sermos capazes de estimar adequadamente a direção de busca.



(a) Passo constante **(b)** Bisseção **(c)** Seção áurea

Figura 5: Resultados para a função (c) com diferentes métodos (revertendo a direção)

De modo a robustecer a implementação para usos futuros, a Figura 5 foi gerada por meio da implementação de uma checagem de derivada numérica idêntica a feita no método da Bisseção para selecionar a metade a manter no início do método do passo constante. Essa checagem, por sua vez, definiu um multiplicador a ser aplicado em $\mathbf{d}$ durante o processo e reportado em α como seu sinal. Para fins de exemplo, um valor booleano adicional foi posto na declaração da função passo para desligar essa checagem, de modo a permitir testes em que seguiríamos cegamente a direção dada, como o que gerou a Figura 4.