Lista 02

MEC 2403 - Otimização e Algoritmos para Engenharia 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
maio de 2023

Lista 02

MEC 2403 - Otimização e Algoritmos para Engenharia Mecânica

Pedro Henrique Cardoso Paulo

maio de 2023

1 Introdução

1.1 Objetivos

Esse é o entregável da Lista 02 da disciplina MEC 2403 - Otimização e Algoritmos para Engenharia Mecânica. Esse trabalho tem como objetivos:

1. Implementar os principais métodos determinação de direções para otimização em múltiplas variáveis
2. Aplicar esses métodos em funções 2D com o uso de passos analíticos
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

Dada a função $f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2$, minimizá-la partindo do ponto $\mathbf{x^0} = [2, 2]^T$ utilizando os métodos de otimização: (a) *Univariante*; (b) *Powell*; (c) *Steepest Descent*; (d) *Fletcher–Reeves*; (e) *BFGS*; e (f) *Newton–Raphson*.

Preencher uma tabela com os resultados obtidos adotando uma tolerência de 10^-5 e um número máximo de 3 passos para cada método. Para cada passo (iteração) de cada método indicar o valor de α obtido na busca linear.

2.2 Solução

E execução deste exercício demandou uma pequena refatoração do código do pacote `steps.py`, desenvolvido para a solução da Lista 01. Essa refatoração implicou na criação de uma classe de passo genérico (`GenericStep`) da qual passos dissociados do passo constante pudessem herdar. Além disso, mais dois arquivos foram criados: `functions.py`, onde foi definido um objeto capaz de aramzenar as informações de uma função, seu gradiente e sua Hessiana para casos analíticos e numéricos e `optimizers.py`, onde foram definidos objetos responsáveis pela execução dos métodos de otimização (sua definição de direção de busca e iteração até a convergência). A Figura 1 exemplifica o resultado final das classes implementadas. Mais detalhes da implementação dos otimizadores será apresentado no Trabalho 01, em elaboração.

Como o principal objetivo do presente exercício é validar a implementação dos métodos de otimização e a função a ser estudada é sabidamente quadrática, dada uma direção de busca $\mathbf{d}_i$, o valor do passo linear ideal α_i pode ser calculado com base no gradiente da função ∇f e em sua Hessiana $\mathbf{H}(f)$ por meio da equação 1.

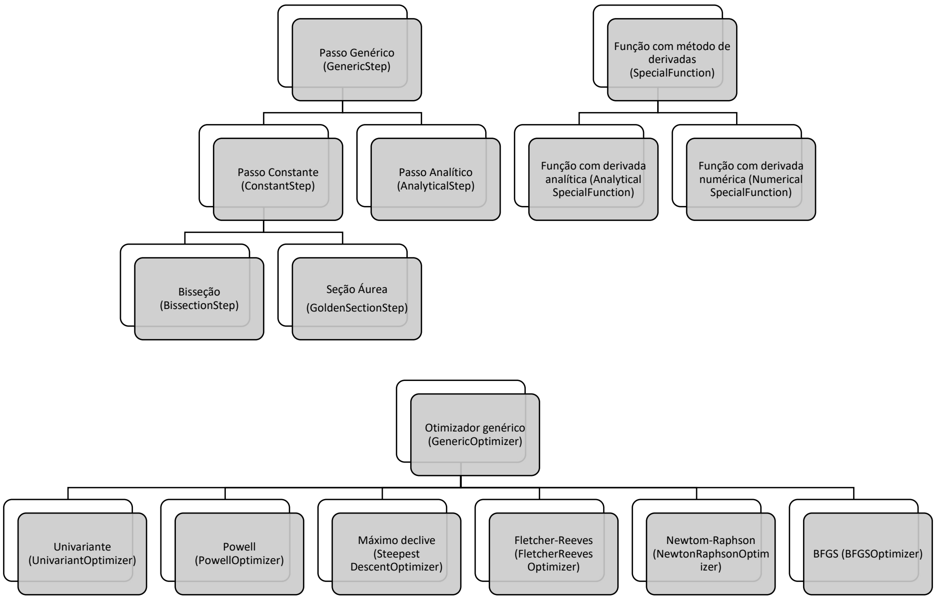


Figura 1: Estrutura de classes implementada e heranças

$$\alpha_i = \frac{\nabla f^T \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{H}(f) \mathbf{d}_i} \tag{1}$$

Para a função $f(x_1, x_2)$ descrita no enunciado, o gradiente e sua Hessiana são descritos nas equações 2 e 3.

$$\nabla f = \begin{bmatrix} 2x_1 - 3x_2 + 1 \\ -3x_1 + 8x_2 - 1 \end{bmatrix} \tag{2}$$

$$\mathbf{H}(f) = \begin{bmatrix} 2 & -3 \\ -3 & 8 \end{bmatrix} \tag{3}$$

Essa equação foi implementada como um objeto chamado `AnalyticalStep` para ser passada aos métodos de otimização. O código desse objeto é mostrado abaixo:

```
class AnalyticalStep(GenericStep):

  def __init__(self):

    super().__init__()

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


    grad = function.grad(*p_initial).reshape(-1,1)
    direction = direction.reshape(-1,1)
    Q = function.Hessian(*p_initial)

    ak = - np.dot(grad.T, direction) / (direction.T @ Q @ direction)
    ak =  ak.reshape(-1)[0]

    pend = pend = p_initial + ak*direction.reshape(-1)

    return ak, pend
```

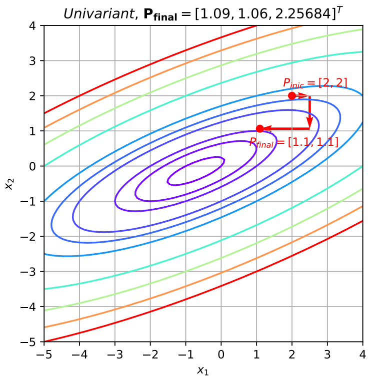
Ressalta-se que o critério de parada usado para os otimizadores foi o critério de parada aplicado para a otimização foi, no presente estudo, um limite de 3 iterações e uma condção de chegada ao ponto crítico definida pela equação 4, com a tolerência *tol* sendo igual a 10^{-5}.

$$|\nabla f| \leq tol \tag{4}$$

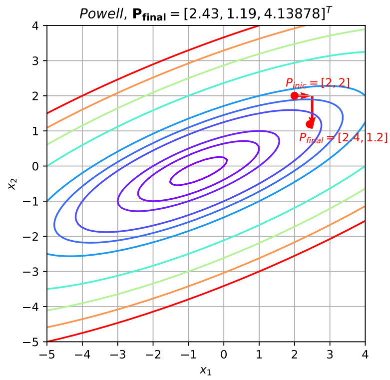
Os resultados finais das 3 iterações executadas são resumidos na Tabela 1, com os resultados por passo em forma gráfica sendo mostrados na Figura 2. Nota-se a partir dos resultados que os valores finais obtidos foram coerentes com o previsto pela teoria, com os métodos Fletcher-Reeves, Newton-Raphson e BFGS sendo os únicos a terem convergido no total de 3 passos ou menos, conforme esperado. Além disso, nota-se que o passo analítico para o método de Newton-Raphson retornou também um valor de 1 na única iteração que este precisou, o que novamente está de acordo com o esperado pelo desenvolvimento teórico do método. Também é interessante notar que os métodos de Powell e Univariante apresentaram passos iguais nas duas primeiras iterações, algo que novamente corrobora com a boa implementação dos métodos dado que eles são idênticos nessas duas iterações.

Método	Ponto de mínimo	Passos	α_1	α_2	α_3
Univariante	$[1.093750, 1.062500, 2.256836]^T$	3	0.50000	-0.93750	-1.40625
Powell	$[2.432024, 1.189955, 4.138784]^T$	3	0.50000	-0.93750	-0.13596
Steepest Descent	$[0.484934, 0.320841, 0.344249]^T$	3	0.11647	0.70690	0.11648
Fletcher-Reeves	$[-0.714286, -0.142857, -0.285714]^T$	2	0.11647	1.22648	–
Newton-Raphson	$[-0.714286, -0.142857, -0.285714]^T$	1	1.00000	–	–
BFGS	$[-0.714286, -0.142857, -0.285714]^T$	2	0.11647	1.22648	–

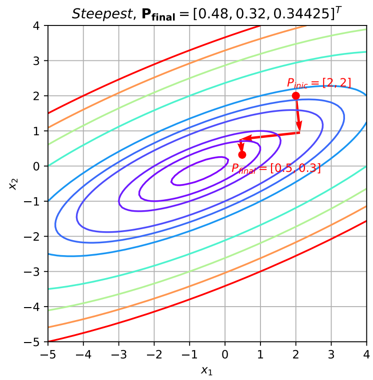
Tabela 1: Resumo dos resultados obtidos



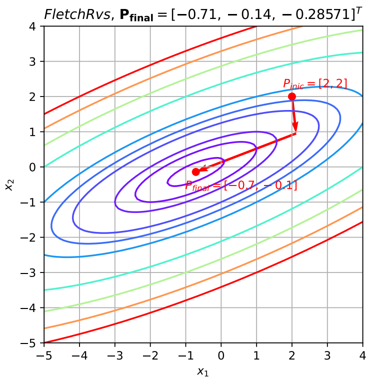
(a) Univariante



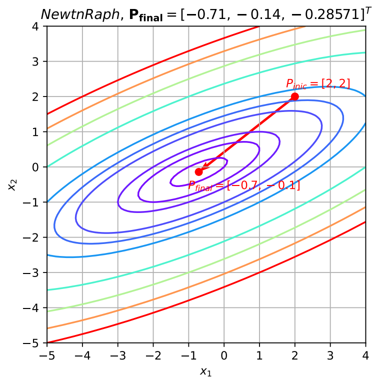
(b) Powell



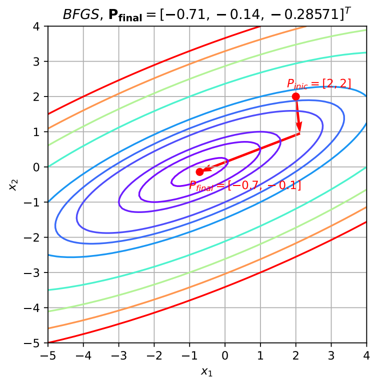
(c) Steepest Descent



(d) Fletcher-Reeves



(e) Newton-Raphson



(f) BFGS

Figura 2: Resumo gráfico dos passos dados em cada método